סיכום ונוסחאון במבוא למדעי המחשב ושפת Java

קורס 20441 · עודכן ב-2026ג׳

(נכתב על ידי דניאל וונג – www.wongi.io)

כל סוגי השאלות במבחנים של הקורס, כל אחד עם תבנית פתרון בצעדים, שרטוט ודוגמה פתורה ממבחן אמיתי. כל קטע קוד ופלט בסיכום הורץ ונבדק ב-Java.

תוכן העניינים

נושא	עמוד	נושא	עמוד
יסודות Java, מחלקות ואובייקטים	1	ירושה ופולימורפיזם	28
רקורסיה	7	רשימות מקושרות	37
יעילות וסיבוכיות	13	מחסניות ותורים	43
עצים בינאריים	21		

ביבליוגרפיה

האוניברסיטה הפתוחה. *מבוא למדעי המחשב ושפת Java – חוברת מצגות : יחידות 1–6* (20441-5087). רעננה : האוניברסיטה הפתוחה.

האוניברסיטה הפתוחה. *מבוא למדעי המחשב ושפת Java – חוברת מצגות : יחידות 7–12* (20441-5095). רעננה : האוניברסיטה הפתוחה.

1 יסודות Java, מחלקות ואובייקטים

1.1 טיפוסים, המרות וביטויים

טיפוס	byte	short	int	long	float	double	char	boolean
ביטים	8	16	32	64	32	64	16	–
טווח	$-128..127$	$-32768..32767$	$-2^{31}..2^{31}-1$	$-2^{63}..2^{63}-1$	$\pm 3.4 \cdot 10^{38}$	$\pm 1.7 \cdot 10^{308}$	0..65535	false/true
ברירת מחדל	0	0	0	0	0.0	0.0	'\u0000'	false

טווח ה-`int`: בערך מינוס עד פלוס 2.1 מיליארד. ברירת המחדל היא של שדה ושל תא במערך. **משתנה מקומי** (בתוך שיטה) לא מקבל ערך לבד: קריאה שלו לפני השמה – שגיאת קומפילציה. **`final`** – קבוע: השמה נוספת – שגיאת קומפילציה. (מצגות 1–6, עמ' 24–28)

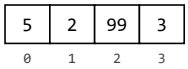
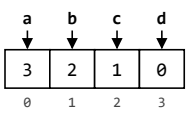
ניסוח	קוד
הירכיית הטיפוסים: `byte` ← `short` ← `int` ← `long` ← `float` ← `double`, וגם `char` ← `int`. מהצר לרחב Java ממירה לבד: **המרת השמה** (`double d = 5` נותן `5.0`) **וקידום** – בפעולה בין `int` ל-`double`, ה-`int` הופך ל-`double`. (עמ' 34–35)	`double d = 5;` `// 5.0` `5.5 / 4` `// 1.375` `7 / 2 * 2.0` `// 3 * 2.0 = 6.0`
casting – מהרחב לצר רק בהמרה מפורשת. `(int)` **קוצץ** את השבר (לכיוון 0, לא מעגל). ה-casting חל רק על מה שצמוד אליו (הוא קודם לכפל ולחילוק). בלי casting – שגיאת קומפילציה *possible lossy conversion*. `+=` ודומיו עושים casting לבד. (עמ' 35)	`(int) 3.9` `// 3` `(int) -3.9` `// -3` `(int) 7.9 / 2` `// 3 (7 / 2)` `(double) 7 / 2` `// 3.5` `(double) (7 / 2)` `// 3.0` `int x = 3.5;` `// שגיאת קומפילציה` `int y = 5; y += 2.7;` `// 7`
חילוק שלמים – כששני האופרנדים שלמים, `/` זורק את השבר (קיצוץ לכיוון 0). מספיק אופרנד ממשי אחד לחילוק ממשי. חילוק שלם ב-0 – `ArithmeticException`; ממשי – `Infinity`. **שארית `%`** – הסימן של התוצאה הוא הסימן של **המספר השמאלי**. `10 % n` – הספרה האחרונה, `10 / n` – המספר בלעדיה. אי-זוגי: `n % 2 != 0` (ולא `== 1`, שנכשל בשליליים). (עמ' 29)	`7 / 2 // 3` `-7 / 2 // -3` `1 / 2 * 4.0` `// 0.0` `10 % 3 // 1` `6 % 7 // 6` `-7 % 3 // -1` `7 % -3 // 1` `1234 % 10 // 4` `1234 / 10 // 123` `-3 % 2 == 1` `// false`
`char` הוא מספר (קוד Unicode): `'a' = 97, 'A' = 65, '0' = 48` =, והאותיות רצופות. חשבון על `char` נותן `int`; חזרה לתו ב-`(char)`. `'a' - c` – מקום האות ב-abc (0–25), האינדקס של המונה שלה. `'0' - '7'` – הספרה כמספר. `>` בין אותיות קטנות = סדר ה-abc; כל אות גדולה קטנה מכל אות קטנה.	`'a' + 1` `// 98` `(char) ('a' + 1)` `// 'b'` `(char) ('a' + i)` `// האות ה-i` `'z' - 'a'` `// 25` `'a' + 'b'` `// 195` `"" + 'a' + 'b'` `// "ab"` `c++; c += 2;` `// char עובר, נשאר` `c = c + 1;` `// שגיאת קומפילציה`
`++` / `--` – אחרי המשתנה (`x++`): משתמשים בערך **הישן** ואז מוסיפים 1. לפני (`++x`): קודם מוסיפים ואז משתמשים. בשורה לבד – אין הבדל. (עמ' 30–32)	`x = 8; y = x++;` `// x = 9 y = 8` `x = 8; y = ++x;` `// x = 9 y = 9` `n = 5; x = n++; y = ++n;` `// x = 5 y = 7` `k = 3; k = k++;` `// k נשאר 3`
`&&` ו-`||` מקצרים: אם הצד השמאלי כבר קובע את התוצאה, הימני **לא מחושב** (`... && false`, `... || true`). לכן את בדיקת הגבול או ה-`null` כותבים **משמאל**; בסדר הפוך – חריגה.	`i < s.length() - 1 && s.charAt(i) == s.charAt(i + 1)` `// בטוח` `s.charAt(i) == s.charAt(i + 1) && i < s.length() - 1` `// חריגה בסוף` `p != null && p.equals("x")` `// בטוח גם כש-p == null`
קדימות, מהגבוהה לנמוכה: `()` ← אוניריים `(int) ! -- ++` ← `% / *` ← `+ -` ← `=> =< > <` ← `== !=` ← `&&` ← `||` ← `?:` ← `+= =`. באותה רמה – משמאל לימין. (עמ' 32, 54) **`else` שייך ל-`if` הקרוב** שאין לו `else`, בלי קשר להזחה: עם `a = 5, b = 0` מודפס `B`, ועם `a = 1` כלום. (עמ' 47)	`4 * (16 - 8 % 2)` `// 64` `10 - 4 - 3` `// 3` `-7 / 2 * 2` `// -6` `!(3 > 2) || 2 > 1 && 1 > 2` `// false` `if (a > 3)` `  if (b > 3) System.out.println("A");` `else System.out.println("B");` `// של ה-if הפנימי`

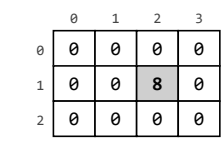
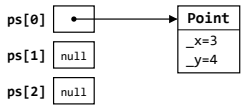
ניסוח	קוד
Math – מחזירות את הטיפוס שקיבלו: `abs`, `max`, `min`. `pow` ו-`sqrt` תמיד `double`. `Math.random()` – ממשי ב-$[0, 1)$. **גלישה**: `int` שעובר את `Integer.MAX_VALUE` מתגלגל לשלילי בלי הודעה. `Integer.MAX_VALUE` משמש כ"אינסוף" בתחילת חיפוש מינימום. (עמ' 39)	`Math.pow(2, 10)` // 1024.0 `Math.sqrt(16)` // 4.0 `Math.round(2.5)` // 3 `Math.round(-2.5)` // -2 `(int) (Math.random() * 6) + 1` // 1 עד 6 `Integer.MAX_VALUE + 1` // -2147483648

1.2 מחרוזות (String)

ניסוח	קוד
`s.length()` – האורך (שיטה, עם סוגריים; במערך `arr.length` בלי). **`s.charAt(i)`** – התו במקום `i`, מ-0 עד `length() - 1`; מחוץ לטווח – `StringIndexOutOfBoundsException`.	`String s = "abcdef";` `s.length()` // 6 `s.charAt(2)` // 'c' `s.charAt(s.length())` // חריגה
`s.substring(i)` – מ-`i` עד הסוף. **`s.substring(i, j)`** – מ-`i` עד `j` **לא כולל**, באורך `j - i`. `substring(length())` ו-`substring(i, i)` חוקיים ומחזירים `""`. **`s.indexOf(x)`** – המקום הראשון של `x` או מחרוזת, `-1` אם אין; `indexOf(x, from)` מחפש מ-`from`; `lastIndexOf` – האחרון.	`s.substring(2)` // "cdef" `s.substring(2, 4)` // "cd" `s.substring(6)` // "" `s.indexOf("de")` // 3 `s.indexOf('z')` // -1 `"banana".indexOf('a', 2)` // 3
`s1.equals(s2)` – אותו תוכן, תו אחרי תו (`equalsIgnoreCase` – בלי הבדל בין גדולות לקטנות). **`==` בין מחרוזות משווה הפניות**, לא תוכן. (עמ' 60) **`s1.compareTo(s2)`** – שלילי אם `s1` קודמת במילון, 0 אם שוות, חיובי אם אחריה: ההפרש בין התווים הראשונים השונים, ואם אחת היא רישא של השנייה – הפרש האורכים.	`String str1 = "test";` `String str2 = new String("test");` `str1 == str2` // false `str1.equals(str2)` // true `"apple".compareTo("arrow")` // -2 ('p' - 'r') `"ab".compareTo("abc")` // -1
שרשור `+` – מחושב משמאל לימין. כל עוד אין מחרוזת, `+` הוא חיבור מספרים (ותווים נספרים כמספרים); מהרגע שיש מחרוזת – הכול מודבק כטקסט. * קודם ל-`+`. (עמ' 56)	`"" + 1 + 2` // "12" `1 + 2 + ""` // "3" `1 + 2 + "x" + 1 + 2` // "3x12" `"x" + (1 + 2)` // "x3" `'a' + 1 + ""` // "98"
מחרוזת לא ניתנת לשינוי (immutable). כל שיטה (`toUpperCase`, `substring`, `+`) מחזירה מחרוזת **חדשה**; אם לא שומרים אותה – לא קרה כלום. לכן שיטה שמקבלת `String` לא משנה את המחרוזת של מי שקרא לה.	`s.toUpperCase();` // s לא השתנתה `s = s.toUpperCase();` // עכשיו כן `res += ch;` // מוסיף תו בסוף `res = s.charAt(i) + res;` // בונה את s הפוכה

1.3 מערכים

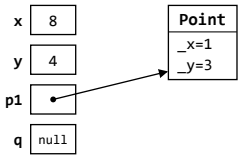
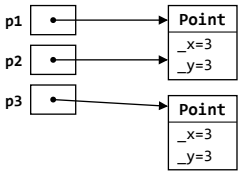
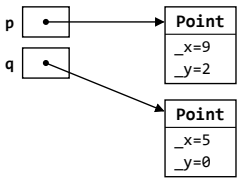
ניסוח	שרטוט	קוד
יצירה – `new int[n]`: הגודל קבוע. התאים מקבלים **ברירת מחדל** (`0`, `0.0`, `false`, `null`). `{…}` – רק בשורת ההצהרה. **`arr.length`** – תכונה, בלי סוגריים. אינדקסים `0` עד `length - 1`; מחוץ להם – `ArrayIndexOutOfBoundsException`. `println(arr)` מדפיס כתובת (`…@I]`), לא איברים. (עמ' 183–187)		`int[] arr = {5, 2, 99, 3};` `arr = {1, 2};` // שגיאת קומפילציה `for (int i = 0; i <= arr.length; i++)` `  sum += arr[i];` // חריגה ב-i = 4
מערך מונים – האינדקס הוא **ערך**, והתא סופר כמה פעמים הערך הופיע. לאותיות: 26 מונים ו-`c[ch - 'a']++`. למספרים אי-שליליים: `new int[max + 1]` ו-`temp[a[i]]++`. מעבר על המונים **לפי הסדר** נותן את הערכים ממוינים.		`int[] c = new int[26];` // s = "aacbab" `for (int i = 0; i < s.length(); i++)` `  c[s.charAt(i) - 'a']++;`

ניסוח	שרטוט	קוד
מערך דו-ממדי – מערך של שורות. `new int[3][4]`: 3 שורות, 4 עמודות. `m.length` – מספר **השורות**, `m[0].length` – מספר **העמודות** (`m[i].length` – אורך שורה `i`). `m[i][j]` – שורה `i`, עמודה `j`. (עמ' 202)		`int[][] m = new int[3][4];` `m[1][2] = 8;` `for (int i = 0; i < m.length; i++)` `  for (int j = 0; j < m[i].length; j++)` `  ... m[i][j] ...`
מערך של אובייקטים – `new Point[3]` יוצר 3 **הפניות**, כולן `null`, ואף `Point`. כל איבר נוצר לבד ב-`new`. קריאה לשיטה דרך תא ריק – `NullPointerException`. (עמ' 197)		`Point[] ps = new Point[3];` `ps[0] = new Point(3, 4);` `ps[1].getX();  // NullPointerException`

1.4 מחלקות ואובייקטים

ניסוח	קוד
מחלקה – **משתני מופע** (`private`, בקורס `x_`), **בנאים** ו**שיטות**. **אובייקט** נוצר ב-`new`, שמקצה לו זיכרון ומחזיר הפניה אליו. (עמ' 91, 104) **בנאי** – בשם המחלקה, **בלי ערך החזרה**. אם לא כתבנו אף בנאי, Java מוסיפה **בנאי ריק**; כשכתבנו בנאי כלשהו הוא **נעלם**, ו-`new Point()` – שגיאת קומפילציה. **בנאי העתקה** `Point(Point other)` – אובייקט חדש עם אותם ערכים. (עמ' 90, 120)	`public class Point {` `  private int _x, _y;  // משתני מופע` `  public Point(int x, int y) { _x = x; _y = y; }` `  public Point(Point other) { _x = other._x; _y = other._y; }` `  public int getX() { return _x; }` `}` `Point p1 = new Point();  // שגיאה: אין בנאי ריק`
this – הפניה לאובייקט שהפעיל את השיטה. כשפרמטר מסתיר שדה באותו שם, `this.x` הוא השדה; `x = x` משים את הפרמטר לעצמו, והשדה נשאר 0. (עמ' 91)	`public Bad(int x)  { x = x; }  // השדה נשאר 0` `public Good(int x) { this.x = x; }  // נכון`
העמסה (overloading) – כמה שיטות או בנאים באותו שם עם **רשימת פרמטרים שונה**: במספר, בטיפוסים או בסדר. ערך החזרה שונה לבד – שגיאת קומפילציה. הבחירה לפי הטיפוסים **בזמן קומפילציה**: התאמה מדויקת, אחרת בהמרה מרחיבה (`char` ← `int` ← `double`); שתי התאמות שקולות – שגיאת *ambiguous*. (עמ' 95–96)	`void print(int i, double d)  // הראשונה` `void print(double d, int i)  // השנייה` `print(10, 10.0);  // הראשונה` `print(10.0, 10);  // השנייה` `print(10, 10);  // שגיאה: ambiguous` `// show(int), show(double): show('a') -> show(int)`
כימוס (encapsulation) – השדות `private`: נגישים רק **בתוך המחלקה** (גם של אובייקט אחר מאותה מחלקה: `other._x`). מבחוץ – רק דרך **שיטה מאחזרת** (`getX`) ו**שיטה קובעת** (`setX`), שיכולה לסנן ערכים לא חוקיים. `p1._x = -7` ממחלקה אחרת – שגיאת קומפילציה *has private access*. (עמ' 99)	`public void setX(int x) {` `  if (x >= 0)  // ערך לא חוקי לא נכנס` `  _x = x;` `}`
`static` – **משתנה של המחלקה**: עותק אחד שמשותף לכל האובייקטים (למשל מונה האובייקטים שנוצרו). **שיטה סטטית** שייכת למחלקה ונקראת בשמה (`Point.getCounter()`); אין לה `this`, ולכן גישה ממנה לשדה מופע – שגיאת קומפילציה *non-static … from a static context*. קבוע – `static final` משותף. (עמ' 146–148)	`private static int counter = 0;` `public Point(int x, int y) { ...; counter++; }` `public static int getCounter() { return counter; }`

1.5 זיכרון: הפניות, הצבעה כפולה, העברת פרמטרים

ניסוח	שרטוט	קוד
פרימיטיבי מול הפניה – משתנה פרימיטיבי מחזיק את **הערך** (במחסנית). משתנה מטיפוס מחלקה או מערך מחזיק **הפניה** לאובייקט שבזיכרון הערימה. `x = y` בפרימיטיבי מעתיק ערך: שינוי x אחר כך לא משנה את y. **`null`** – הפניה שלא מצביעה לאף אובייקט (ברירת המחדל של שדה או תא מטיפוס מחלקה); קריאה לשיטה דרכה – `NullPointerException`, ולכן בודקים קודם `q != null &&`. (עמ' 104–105, 118)		`int x = 4;` `int y = x;` `x = 8;` // y נשאר 4 `Point q = null;` `q.getX();` // *NullPointerException*
הצבעה כפולה (aliasing) – `p2 = p1` מעתיק את **ההפניה**, לא את האובייקט: שני משתנים, **אובייקט אחד**, ושינוי דרך אחד נראה דרך השני. כך גם במערכים (`arr2 = arr1`). לעותק אמיתי – בנאי העתקה, או מערך חדש ולולאה. (עמ' 119–120, 187–188) **`==` מול `equals`** – בין הפניות, `==` בודק אם זה **אותו אובייקט**. השוואת תוכן – רק ב-`equals` שהמחלקה כתבה; בלי `equals` משלה, `equals` של `Object` הוא בדיוק `==`. (עמ' 106)		`Point p1 = new Point(1, 3);` `Point p2 = p1;` `p2.setX(3);` // *p1.getX() == 3* `Point p3 = new Point(3, 3);` `p1 == p2` // *true* `p1 == p3` // *false* `p1.equals(p3)` // *true* `public boolean equals(Point other) {` `  return _x == other._x && _y == other._y;` `}`
העברת פרמטרים – תמיד עותק של הערך. פרמטר פרימיטיבי: השיטה לא משנה את המשתנה של הקורא (`swap(int a, int b)`). פרמטר הפניה: עותק של **ההפניה** – השיטה **כן** משנה את האובייקט או המערך עצמו, אבל השמה חדשה לפרמטר (`q = new …`) לא מזיזה את המשתנה של הקורא. בשרטוט: המצב בסוף `change(p)`. (עמ' 188)		`static void change(Point q) {` `  q.setX(9);` // משנה את האובייקט של p `  q = new Point(0, 0);` // רק q זז `  q.setX(5);` `}` `Point p = new Point(1, 2);` `change(p);` // *p.getX() == 9* `swap(int[] a, int i, int j)` // כן מחליף במערך

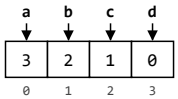
1.6 "מה מבצעת השיטה" – מחרוזות ומערכים

תבנית פתרון – מה השיטה עושה (מה, לא איך)

1. **קוראים את התנאים** – מה מובטח על הקלט (רק אותיות קטנות? מספרים אי-שליליים? שונים זה מזה?). שיטה שקוראת לשיטה אחרת – מתחילים מהפנימית.
2. **נותנים שם לכל משתנה** – מה הוא שומר בכל רגע (`c[p]` – כמה פעמים הופיעה האות ה-p", "`temp` – אורך הרצף הנוכחי, `c` – הארוך עד עכשיו").
3. **עוקבים על הדוגמה שבשאלה** בטבלה: עמודה לכל משתנה, שורה לכל סיבוב. זו גם התשובה לסעיף "מה יוחזר".
4. **מכלילים** – מריצים בראש 2–3 קלטים קטנים משלנו (הכול זהה, הכול שונה, כבר ממוין) ומחפשים את המשמעות.
5. **כותבים מה, לא איך** – משפט אחד על **הערך המוחזר** (או על השינוי במערך), בלי "הלולאה עוברת...".
6. **מקרי קצה** – קלט ריק (מה מוחזר, או איזו חריגה), איבר אחד, תיקו (< שומר את **הראשון**, =< את **האחרון**), ערך שלא נמצא, תו מחוץ לתנאי.

דוגמה – (2017ב, שאלון 467, מועד 84, שאלה 6)

```
public static String what(String s) {
    int[] c = new int[26];
    String res = "";
    for (int i = 0; i < s.length(); i++)
        c[s.charAt(i) - 'a']++;
    for (int i = 0; i < c.length; i++)
        if (c[i] != 0)
            res += (char) ('a' + i);
    return res;
}
public static boolean secret(String s1, String s2) {
    return what(s1).equals(what(s2));
}
```



מעקב על "aacbab": הלולאה הראשונה סופרת: a – 3, b – 2, c – 1. השנייה עוברת על המונים **לפי הסדר** ומוסיפה כל אות שהמונה שלה לא 0, **פעם אחת**: "abc". **הכללה:** "banana" ← "abn", "zzz" ← "z".

what: מחזירה מחרוזת של **האותיות השונות** שב-s, כל אחת פעם אחת, **לפי סדר ה-abc**. למחרוזת ריקה – "" (לא `null` ולא חריגה).

secret: מחזירה `true` אם `s1` ו-`s2` בנויות מאותה **קבוצת אותיות**, בלי קשר לסדר ולמספר החזרות ("aab", "abb" ← `true`; "abc", "abcd" ← `false`; שתי ריקות ← `true`), ואחרת `false`.

שימו לב: `secret` **אינה** בודקת אנגרמה: `what` זורקת את הכמויות. ולהשוואה – `equals`, לא ==.

שאלות נוספות מאותו סוג: בטבלת הדפוסים למטה.

דפוסים שחוזרים בשאלות – ומה הם אומרים

ניסוח	קוד
מערך מונים – כמה פעמים הופיע כל ערך. מעבר על המונים לפי הסדר = הערכים ממוינים; צבירה עד k = האיבר ה-k בגודלו (k = 1..10: {71,5,2,12,84,71,85,192,71,85}, 192 85 85 84 71 71 71 12 5 2). (2017ב, שאלון 467, מועד 85, שאלה 6) (2024א, שאלון 462, מועד 63, שאלה 5)	`temp[a[i]]++;` `for (i = 0; i < temp.length && total < k; i++)` `  total += temp[i];` `return i - 1;`
אינדקס המינימום / המקסימום – < (או >) בהשוואה שומר את המופע **הראשון**, =< עובר ל**אחרון**. למשל fun1("banana", 3) מחזירה 4, המקום הראשון של n מ-3 והלאה. (2016א, שאלון 481, מועד 87, שאלה 6)	`if (a[i] > a[m]) m = i;  // {3,9,1,9,2}: 1` `if (a[i] >= a[m]) m = i;  // {3,9,1,9,2}: 3`
הרצף הנוכחי מול הטוב ביותר – `temp` גדל כל עוד הרצף נמשך ומתאפס כשהוא נשבר; `c` שומר את השיא. התשובה: הרצף / הקטע הארוך (או הכבד) ביותר, ובתיקו – הראשון. "aaabbbbaaccde" ← 'b'; הסכום הגדול ביותר של קטע לא יורד ב-{1,11,15,-3,5,7,4,2,13,27} ← 42. (2017ב, שאלון 467, מועד 93, שאלה 6) (2016א, שאלון 481, מועד 92, שאלה 6)	`if (i < s.length() - 1 && s.charAt(i) == s.charAt(i + 1))` `  temp++;` `else {` `  if (temp > c) { c = temp; res = s.charAt(i); }` `  temp = 1;` `}`

ניסוח	קוד
שתי לולאות `i`, `j` על כל הקטעים – בודקות כל תת-מחרוזת / תת-מערך רציף ושומרות את הארוך ביותר שמקיים תנאי: הפלינדרום הארוך ("abb" ← "bb", "ab" ← "a"); תת-המערך הארוך של מספרים עוקבים (`max - min == j - i` כשכולם שונים). (2024ב, שאלון 468, מועד 87, שאלה 5) (2024א, שאלון 462, מועד 69, שאלה 5)	`for (int i = 0; i < s.length(); i++)` `  for (int j = i + 1; j <= s.length(); j++)` `  ... s.substring(i, j) ...`
"המקום האחרון שבו ראינו" – `temp1` / `temp2` זוכרים את המופע האחרון של `x` ושל `y`, ובכל מפגש מודדים את המרחק: המרחק הקטן ביותר בין מופע של `x` למופע של `y`. (2025א, שאלון 463, מועד 66, שאלה 5)	`if (a[i] == x) { temp1 = i; ... }` `else if (a[i] == y) { temp2 = i; ... }`
שיטות עזר שמרכיבות מיון – מוצאים מינימום / מקסימום ומעבירים אותו לקצה (החלפה, היפוך רישא, הזזה). בודקים את **הכיוון** על 3 איברים: "cadb" ← "dcba" (יורד); {7,9,2,5,8,1,4} ← 1 2 4 5 7 8 9 (עולה, משני הקצוות). (2016א, שאלון 481, מועד 83, שאלה 6) (2016א, שאלון 481, מועד 87, שאלה 6) (2025ב, שאלון 473, מועד 78, שאלה 5)	`int m = g(s, i);` // מקום הקטנה מבין 0..i `s = f(s, m);` // היפוך 0..m: היא בהתחלה `s = f(s, i);` // היפוך 0..i: היא במקום i

שימו לב: מלכודות שחוזרות בשאלות האלה:

- **קלט ריק:** `s.charAt(0)` או `a[0]` בשורה הראשונה – חריגה, `StringIndexOutOfBoundsException` או `ArrayIndexOutOfBoundsException`. לולאה שלא רצה – מוחזר הערך ההתחלתי: "", 0, ואפילו 1 למערך ריק ב-2024א מועד 69.
- **`else if`:** אותו תא לא נספר פעמיים – כש-`x == y` אף מרחק לא נמדד, ומוחזר `Integer.MAX_VALUE`.
- **תו מחוץ לתנאי:** אות גדולה או רווח נותנים `'a' - ch` שלילי – `ArrayIndexOutOfBoundsException`; לכן הבדיקה `'a' <= ch && ch <= 'z'`, ואות גדולה לא נמחקת ("Every Day" בלי "ey" ← "Evr Da").
- **הכיוון:** הקטן שנשלח **לסוף** נותן מיון **יורד** ("abc" ← "cba").
- **`s = f(s)`** משנה רק את המשתנה המקומי; שיטה `void` שמקבלת מערך כן משנה את המערך של הקורא.

2 רקורסיה

2.1 התבנית הכללית – שאלה 1, ״כתבו שיטה רקורסיבית״

תבנית פתרון – שיטה רקורסיבית בלי לולאות

1. **מזהים את הדפוס** לפי מה שמחזירים: ״האם קיים״ – בדיקה, ״כמה״ – צבירה, ״הכי קטן / גדול״ – מיטוב (הטבלה למטה).
2. **החתימה מהשאלה + שיטת עזר.** את החתימה אסור לשנות, ולכן היא רק קוראת לשיטת עזר private באותו שם (העמסת-יתר) עם הפרמטרים הנוספים: אינדקס i, תא (row, col), הסכום שנצבר, מונה, המסלול כ-String. (מצגות 7–12, עמ׳ 86)
3. **תנאי כישלון** – מצב שכבר לא יכול להצליח: יצאנו מהמערך, התא כבר במסלול, עברנו את הסכום, נגמרו הצעדים.
4. **תנאי הצלחה** – הגענו למטרה: הסכום 0, הגענו לתא האחרון, כל האיברים חולקו והתנאי מתקיים.
5. **הענפים** – קריאה אחת לכל בחירה: לקחת / לדלג, ארבעה כיוונים, צעד קטן / גדול. כל קריאה מתקרבת לתנאי עצירה, למשל i + 1 או k - 1. (עמ׳ 68)
6. **הצירוף** – || ל״האם קיים״, && ל״כולם״, + ל״כמה״, Math.max / Math.min ל״הכי״.
7. **ביטול (Un-choose)** – אם סימנו משהו במערך משותף, מחזירים אותו לקדמותו לפני שחוזרים. פרמטר int או String חוזר לבד.

```
public static boolean sumNum(int[] a, int num) {
    return sumNum(a, num, 0);            // החתימה רק מתחילה
}
private static boolean sumNum(int[] a, int num, int i) {
    if (i == a.length) return false;     // כישלון
    if (a[i] == num) return true;        // הצלחה
    return sumNum(a, num - a[i], i + 1)  // a[i] לוקחים את
        || sumNum(a, num, i + 1);        // a[i] מדלגים על
}
```

התבנית על sumNum מהמצגות – האם יש איברים שסכומם num. למשל, sumNum({9,2,3,7,5}, 17) מחזירה true, כי $17 = 5 + 3 + 9$. (מצגות 7–12, עמ׳ 93–94)

שימו לב: בלי לולאות בכלל, גם לא בשיטות העזר. כל לולאה הופכת לשיטת עזר עם אינדקס (2.6). השאלה תמיד מתירה העמסת-יתר.

שימו לב: סדר הבדיקות. בדיקת הגבולות באה לפני כל קריאה של a[i] או mat[row][col]. הצלחה שיכולה לקרות בדיוק כשהקלט נגמר באה לפני הכישלון ״הקלט נגמר״ (2.7).

ניסוח	קוד
בדיקה – ״האם קיים / האם אפשר״. בערך 15 שאלות. כישלון false, הצלחה true. האופרטור || נעצר בענף הראשון שהצליח.	`return f(.., a) || f(.., b);`
צבירה – ״כמה דרכים / כמה מסלולים״. בערך 15 שאלות. כישלון 0, הצלחה 1, מחברים את כל הענפים.	`return f(.., a) + f(.., b);`
מיטוב – ״הכי זול / ארוך / כבד״. בערך 33 שאלות, הדפוס הנפוץ. כישלון הוא **זקיף**: Integer.MAX_VALUE במינימום, Integer.MIN_VALUE או -1 במקסימום. הצלחה – הערך.	`return Math.min(f(.., a), f(.., b));`
הדפסת כל הפתרונות – פרמטר String path שגדל בכל קריאה. מדפיסים רק בתנאי ההצלחה. String לא משתנה, ולכן אין מה לבטל. (עמ׳ 81)	`f(.., path + i + " ");`

2.2 מסלול במטריצה – סימון וחזרה לאחור

תבנית פתרון – מסלול במטריצה (backtracking). 23 מתוך 66 השאלות

1. **הפרמטרים** – `(row, col)` של התא הנוכחי, ומה שנשאר: הסכום, `k`, האות הבאה במילה. אם סופרים פניות – גם הכיוון הקודם.
2. **כישלון, בסדר הזה** – מחוץ לגבולות, אחר כך התא כבר במסלול (מסומן), ורק אז התא אסור או שעברנו את הסכום.
3. **הצלחה** – הסכום 0, הגענו לתא היעד, נבנתה כל המילה.
4. **בחירה (Choose)** – מסמנים את התא. אם מותר לשנות את המערך: שומרים את `int value = mat[row][col]`, ושמים בתא ערך שלא מופיע במערך, למשל `-1` או `'!'`. אם אסור: מסמנים במערך אחר, כמו `path`.
5. **חקירה (Explore)** – ארבע קריאות: `row - 1`, `row + 1`, `col - 1`, `col + 1`, מצורפות לפי הדפוס.
6. **ביטול (Un-choose)** – מחזירים את התא לקדמותו לפני ה-`return`. רק כשהשאלה רוצה את המסלול מסומן בסוף, משאירים את הסימון בהצלחה.
7. **המסלול מתחיל בכל תא?** שיטת עזר שסורקת את `(row, col)` במקום לולאה כפולה: בסוף שורה עוברים ל-`(row + 1, 0)`.

דוגמה – findSum, מסלול שסכומו sum, מסומן ב-path (2018ב, שאלון 475, מועד 82, שאלה 1)

אסור לשנות את `mat`, ולכן `path` משמש גם כסימון "כבר במסלול".

```
private static boolean findSum(int[][] mat, int sum,
                               int[][] path, int row, int col) {
    if (row < 0 || row >= mat.length || col < 0
        || col >= mat[0].length || path[row][col] == 1)
        return false;                          // כישלון
    sum -= mat[row][col];
    if (sum < 0) return false;                 // הערכים חיוביים
    path[row][col] = 1;                        // בחירה
    if (sum == 0) return true;                 // הצלחה, הסימון נשאר
    if (findSum(mat, sum, path, row - 1, col)
     || findSum(mat, sum, path, row + 1, col)
     || findSum(mat, sum, path, row, col - 1)
     || findSum(mat, sum, path, row, col + 1))
        return true;                           // חקירה
    path[row][col] = 0;                        // ביטול
    return false;
}
// הסריקה של תאי ההתחלה, במקום לולאה כפולה
private static boolean start(int[][] mat, int sum,
                             int[][] path, int row, int col) {
    if (row == mat.length) return false;
    if (col == mat[0].length)
        return start(mat, sum, path, row + 1, 0);
    return findSum(mat, sum, path, row, col)
        || start(mat, sum, path, row, col + 1);
}
```

	0	1	2	3
0	**2**	41	3	14
1	**2**	1	24	7
2	2	15	10	54
3	63	22	2	4

עם `sum = 4` מוחזר `true`, ו-`path` מסמן את `[0][0]` ואת `[1][0]` (המסומנים). עם `sum = 9` מוחזר `false`, ו-`path` נשאר כולו אפסים.

שימו לב: שוכחים לבטל. בלי `path[row][col] = 0` הניסיונות שנכשלו נשארים מסומנים: עם `sum = 9` מוחזר `false`, אבל `path` לא כולו אפסים. גרוע מזה, תא שנשאר מסומן חוסם ניסיון אחר: ב-`{{2,4},{2,2}}` עם `sum = 6` מוחזר `false`, אף שיש מסלול $4+2$.

שימו לב: הגבולות קודם. אם בודקים את `path[row][col] == 1` לפני `row < 0`, הקריאה עם `row = -1` זורקת `ArrayIndexOutOfBoundsException`. האופרטור `||` מפסיק באמת הראשונה, ולכן הגבולות חייבים להיות ראשונים.

שימו לב: רק ימינה / למטה – אי אפשר לחזור לתא, ולכן לא צריך לסמן ולא לבטל.

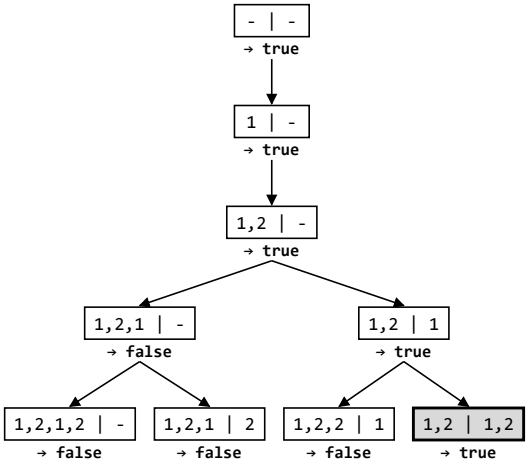
שאלות נוספות מאותו סוג: `canFind` (2026א, שאלון 462, מועד 83, שאלה 1); `maxPathAve` (2026א, שאלון 462, מועד 97, שאלה 1); `cntTrueReg` (2017ב, שאלון 467, מועד 93, שאלה 1); `prince` (2018ב, שאלון 472, מועד 83, שאלה 1).

2.3 תת-קבוצות, חלוקה וסכום – לקחת או לדלג

תבנית פתרון – בכל איבר מחליטים לאן הוא הולך. 16 שאלות

1. **הפרמטרים** – `i` (האיבר הבא), ולכל קבוצה מה שצריך לבדוק בסוף: הסכום, מספר האיברים. אין מערך עזר: הקבוצות חיות בפרמטרים.
2. **כישלון מוקדם** (כשהערכים חיוביים) – הסכום שנשאר שלילי: עוצרים.
3. **סוף המערך** `i == arr.length` – בודקים את **כל** התנאים (סכום וגם גודל) ומחזירים לפי הדפוס.
4. **הענפים** – `arr[i]` לקבוצה 1 או לקבוצה 2 (או: לוקחים / מדלגים). כל ענף מתקדם ל-`i + 1`.
5. **איבר שאפשר לקחת שוב** (כמה מקטעים מאותו אורך) – בענף "לוקחים" נשארים על `i`, ורק "מדלגים" מתקדם.
6. **הצירוף** – `||` ל"אפשר", `+` ל"כמה חלוקות", `Math.min` ל"ההפרש הקטן".

דוגמה – equalSplit, שתי קבוצות שוות בגודל ובסכום (2021ב, שאלון 477, מועד 62, שאלה 1)




```java
public static boolean equalSplit(int[] arr) {
    return equalSplit(arr, 0, 0, 0, 0, 0);
}
private static boolean equalSplit(int[] arr, int i,
        int sum1, int sum2, int cnt1, int cnt2) {
    if (i == arr.length)                    // כולם חולקו
        return sum1 == sum2 && cnt1 == cnt2;
    return equalSplit(arr, i + 1, sum1 + arr[i], sum2,
                      cnt1 + 1, cnt2)       // לקבוצה 1
        || equalSplit(arr, i + 1, sum1, sum2 + arr[i],
                      cnt1, cnt2 + 1);      // לקבוצה 2
}
```

העץ של {1, 2, 1, 2} (קבוצה 1 | קבוצה 2): האופרטור `||` עוצר ב-`true` הראשון, ולכן הענף "לקבוצה 2" של השורש לא נקרא. בשאלה: המערך {-3,5,12,14,-9,13} מחזיר `true`; המערך {-3,5,-12,14,-9,13} מחזיר `false` (הסכומים שווים, הגדלים לא); המערך {-3,5,-12,14,-9} מחזיר `false`.

שימו לב: "אסור לשנות את המערך ואסור מערך עזר" – לכן לא מוחקים איברים ולא בונים קבוצות. הבחירה נשמרת רק בסכומים ובמונים שבפרמטרים, ואין מה לבטל.

שאלות נוספות מאותו סוג: `isSum` (2019ב, שאלון 466, מועד 93, שאלה 1); `split3` (2021ב, שאלון 477, מועד 60, שאלה 1); `knapSack` (2017א, שאלון 455, מועד 90, שאלה 1); `makeSum` (2020ב, שאלון 478, מועד 81, שאלה 1).

2.4 הכי זול / הכי ארוך – דפוס המיטוב

תבנית פתרון – מינימום / מקסימום עם זקיף. כ-33 שאלות, במטריצה, במערך ובמחרוזות

1. **כישלון – זקיף.** ערך שאף פתרון אמיתי לא מגיע אליו: `Integer.MAX_VALUE` במינימום, כדי ש-`Math.min` לא יבחר בו. במקסימום: `Integer.MIN_VALUE`, או `-1` / `0` כשכל הערכים חיוביים.
2. **הצלחה** – מחזירים את הערך של הפתרון שנבנה.
3. **העלות – בפרמטר או בחזרה.** בפרמטר: מוסיפים את `st[i]` ל-`sum` לפני הקריאה, והזקיף אף פעם לא מחובר. בחזרה: `st[i] + best`, ולפני החיבור בודקים אם `best` הוא הזקיף.
4. **הצירוף** – `Math.min(a, b)` או `Math.max(a, b)`, על כל הענפים.
5. **בשיטה החיצונית** – מתרגמים את הזקיף למה שהשאלה מבקשת ("אם אין מסלול החזירו `-1`").

דוגמה – cheapRt, המסלול הזול עם step1, ו-step2 לכל היותר limit פעמים (2022ב, שאלון 477, מועד 91, שאלה 1)




```
0 2 4 6 8 = 34
0 2 5 8 = 13
0 3 5 8 = 8
0 3 6 8 = 19
```

```
private static int cheapRt(int[] st, int step1, int step2,
        int limit, int i, int sum, String path) {
    if (i >= st.length || limit < 0)          // כישלון
        return Integer.MAX_VALUE;
    sum += st[i];                             // משלמים על i
    if (i == st.length - 1) {                 // הצלחה
        System.out.println(path + i + " = " + sum);
        return sum;
    }
    path += i + " ";
    int a = cheapRt(st, step1, step2, limit - 1,
                    i + step2, sum, path);    // צעד step2
    int b = cheapRt(st, step1, step2, limit,
                    i + step1, sum, path);    // צעד step1
    return Math.min(a, b);
}
```

הקריאה cheapRt(st, 3, 2, 4) מדפיסה את ארבעת המסלולים ומחזירה 8 (המסומנים). עם step2 = 6 היא לא מדפיסה כלום ומחזירה 2147483647.

שימו לב: הזקיף גולש. החיבור Integer.MAX_VALUE + 2 נותן 2147483647-. מי שכותב return st[i] + Math.min(a, b); מחבר לזקיף בכל פעם ששני הענפים נכשלו, והמספר השלילי "מנצח" ב-Math.min: באותה דוגמה מוחזר 2147483640- במקום 8. גם Integer.MIN_VALUE + (-5) נותן 2147483643. לכן מחזירים את הזקיף כמו שהוא, או צוברים בפרמטר:

```
int best = Math.min(a, b);
if (best == Integer.MAX_VALUE) return Integer.MAX_VALUE;   // אין מסלול מכאן
return st[i] + best;
```

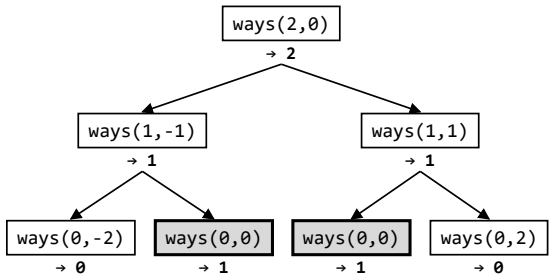
שאלות נוספות מאותו סוג: cheapestRoute (2018א, שאלון 483, מועד 91, שאלה 1); minPrice (2021א, מועד 85, שאלה 1); heaviestPath (2024ב, שאלון 465, מועד 99, שאלה 1); oneFiveSeven (2017ב, שאלון 467, מועד 85, שאלה 1).

2.5 כמה דרכים – דפוס הצבירה

תבנית פתרון – ספירה. כ-15 שאלות: דרכים, מסלולים, חלוקות, סדרות

1. **הצלחה** – נבנתה דרך שלמה וחוקית: return 1.
2. **כישלון** – הדרך לא חוקית או שנגמרו הצעדים: return 0.
3. **הענפים** – קריאה לכל צעד אפשרי, **ומחברים** את כולם. אסור לעצור בענף הראשון (כמו ב-||).
4. **תנאי העצירה כולל את כל התנאים** – "נגמרו הצעדים" מחזיר 1 רק אם גם הגענו ליעד.
5. **שימו לב לאפס** – "0 צעדים ואני כבר ביעד" היא דרך אחת.

דוגמה – ways, להגיע ב-k צעדים בדיוק לנקודה במרחק n ימינה (2017ב, שאלון 467, מועד 84, שאלה 1)




```
public static int ways(int k, int n) {
    if (k == 0)                          // נגמרו הצעדים
        return n == 0 ? 1 : 0;           // ביעד: דרך אחת
    return ways(k - 1, n - 1)            // צעד ימינה
         + ways(k - 1, n + 1);           // צעד שמאלה
}
```

בשאלה: ways(0, 0) מחזירה 1, ways(4, 2) מחזירה 4, ways(4, 3) מחזירה 0.

שאלות נוספות מאותו סוג: howManySorted (2018א, שאלון 486, מועד 87, שאלה 1); countPairs (2023א, שאלון 471, מועד 65, שאלה 1); totalWays (2020א, שאלון 470, מועד 87, שאלה 1); howManyPaths (2019ב, שאלון 466, מועד 86, שאלה 1).

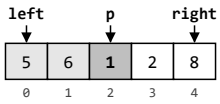
2.6 מעבר על מערך בלי לולאה – אינדקס כפרמטר

תבנית פתרון – כל לולאה היא שיטת עזר. 10 שאלות: מערך מאוזן, מטריצה בתוך מטריצה, מיון ברקורסיה

1. **האינדקס של הלולאה הוא פרמטר**: הלולאה `for (k = lo; k <= hi; k++)` הופכת לשיטה `f(.., k)`.
2. **תנאי הלולאה הוא תנאי העצירה** – `if (k > hi) return` הערך של ״הלולאה נגמרה״ (`0` לסכום, `-1` ל״לא נמצא״).
3. **גוף הלולאה** – מטפלים ב-`a[k]`, וממשיכים ב-`return f(.., k + 1)`.
4. **לולאה כפולה** – פרמטרים `(row, col)`: בסוף שורה (`col == mat[0].length`) ממשיכים ל-`(row + 1, 0)`; בסוף המטריצה עוצרים.
5. **חלק של מערך** – מעבירים `left, right` במקום ליצור מערך חדש (מערך עזר אסור).

דוגמה – isBalanced / where, מערך k-מאוזן (2025ב, שאלון 473, מועד 78, שאלה 1)

מערך מאוזן סביב `p` אם סכום `vec[0..p-1]` שווה לסכום `vec[p..]`. מערך הוא k-מאוזן אם הוא מאוזן, ושני החלקים (k−1)-מאוזנים.




```
static int sum(int[] vec, int lo, int hi) {   // לולאת הסכום
    if (lo > hi) return 0;
    return vec[lo] + sum(vec, lo + 1, hi);
}
static int where(int[] vec, int left, int right, int p) {
    if (p > right) return -1;                 // כל p נוסה
    if (sum(vec, left, p - 1) == sum(vec, p, right))
        return p;
    return where(vec, left, right, p + 1);    // ה-p הבא
}
static boolean isBalanced(int[] vec, int k,
                          int left, int right) {
    if (k == 0) return true;
    int p = where(vec, left, right, left);
    if (p == -1) return false;
    return isBalanced(vec, k - 1, left, p - 1)   // שני החלקים
        && isBalanced(vec, k - 1, p, right);
}
```

כאן `where` מחזירה 2, כי $5 + 6 = 1 + 2 + 8$. בשאלה: המערך {1,2,2,1,3,2,1} הוא 2-מאוזן; המערך {2,1,3,5,1,1,2,3,1,2,2,1} הוא 2-מאוזן, אבל לא 3-מאוזן.

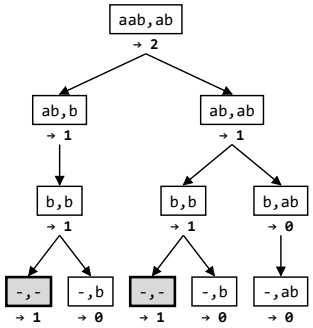
שאלות נוספות מאותו סוג: `isBalanced` (2016ב, שאלון 490, מועד 82, שאלה 1); `matInMat` (2025ב, שאלון 470, מועד 91, שאלה 1); `bubbleSort` (2026ב, שאלון 468, מועד 98, שאלה 1); `fixSort` (2026ב, שאלון 468, מועד 91, שאלה 1).

2.7 מחרוזות – שני אינדקסים

תבנית פתרון – מחרוזת עם אינדקסים. 8 שאלות: תת-סדרה, פלינדרום, מרחק עריכה

1. **אינדקס לכל מחרוזת** – `i` ב-`str` ו-`j` ב-`pattern` (בדרך כלל מותר רק `charAt` ו-`length`). בפלינדרום: `i` מההתחלה ו-`j` מהסוף, והם מתקרבים.
2. **ההצלחה קודם** – `j == pattern.length()`: כל התבנית נמצאה. רק אחריה הכישלון `i == str.length()`.
3. **הענפים** – כשהתווים שווים מתקדמים בשניהם, `(i + 1, j + 1)`. תמיד אפשר גם לדלג על תו של `str`: `(i + 1, j)`. בעריכה, הוספה ומחיקה הן עוד ענפים.
4. **הצירוף** – `+` לספירה, `Math.max` לאורך, `Math.min` למספר פעולות.

דוגמה – count, כמה פעמים pattern מופיעה כתת-סדרה של str (2019א, שאלון 478, מועד 93, שאלה 1)




```
private static int count(String str, String pattern,
                         int i, int j) {
    if (j == pattern.length()) return 1;   // הצלחה – קודם
    if (i == str.length()) return 0;       // כישלון
    int use = 0;
    if (str.charAt(i) == pattern.charAt(j))
        use = count(str, pattern, i + 1, j + 1);  // משתמשים
    return use + count(str, pattern, i + 1, j);   // מדלגים
}
```

בשאלה, count("subsequence", "sue") מחזירה 7. בעץ, של count("aab", "ab"), כל קופסה היא מה שנשאר משתי המחרוזות.

שימו לב: הסדר ההפוך – אם i == str.length() נבדק ראשון, התאמה שנגמרת בתו האחרון של str הולכת לאיבוד: מוחזר 4 במקום 7.

ניסוח	קוד
פלינדרום – i ו-j מתקרבים. כשנשאר תו אחד או אף תו: true.	`if (i >= j) return true;` `if (s.charAt(i) != s.charAt(j)) return false;` `return isPal(s, i + 1, j - 1);`

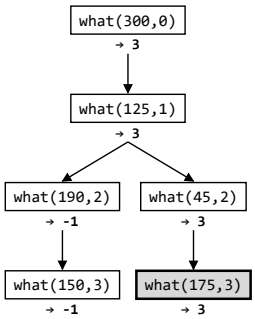
שאלות נוספות מאותו סוג: edit (2017א, שאלון 455, מועד 86, שאלה 1); minimalSt (2023ב, שאלון 468, מועד 83, שאלה 1); lcs (2025א, שאלון 463, מועד 66, שאלה 1); longestPalindrome (2019א, שאלון 478, מועד 85, שאלה 1).

2.8 "מה עושה השיטה" – מעקב אחרי רקורסיה

תבנית פתרון – עץ קריאות

1. **מציירים עץ קריאות** – כל קריאה היא קופסה עם הפרמטרים שלה, והקריאות שהיא עושה מתחתיה, משמאל לימין לפי סדר הקוד.
2. **יורדים עד תנאי עצירה**, וכותבים מה הוא מחזיר.
3. **עולים** – כל קריאה מצרפת את מה שקיבלה (+, ||, max, "אם עוד לא נמצא"), וכותבים את הערך מתחת לקופסה.
4. **מריצים 2–3 קלטים קטנים**, מוצאים את החוק ועונים **מה** השיטה עושה (לא איך), כולל מקרי קצה: ריק, לא נמצא, איבר אחד.

דוגמה – what(root, c) על עץ בינרי (2016ב, שאלון 490, מועד 87, שאלה 3)




```
public static int what(Node root, int c, int x) {
    int ans = -1;
    if (root.getNumber() == c)
        ans = x;
    else {
        if (root.getLeftSon() != null)
            ans = what(root.getLeftSon(), c, x + 1);
        if (ans == -1 && root.getRightSon() != null)
            ans = what(root.getRightSon(), c, x + 1);
    }
    return ans;
}
```

הקריאה what(root, 175) מחזירה 3. אל 400 לא יורדים, כי ans כבר לא 1-. **התשובה:** השיטה מחזירה את העומק של הצומת שבו c (השורש בעומק 0), ו-1- אם c לא בעץ או שהעץ ריק.

שאלות נוספות: מעקבים על עצים ועל רשימות, בפרקים 4 ו-6.

2.9 לפני שמגישים

- **לולאות** – אין for או while באף שיטה, גם לא בשיטות העזר.
- **החתימה** – בדיוק כמו בשאלה. הפרמטרים הנוספים – בשיטת עזר באותו שם.
- **המערך** – לא שונה, או הוחזר לקדמותו בכל return שאחרי סימון.
- **זקיף** – לא מחברים ל-MAX_VALUE או ל-MIN_VALUE. השיטה החיצונית מחזירה את מה שהשאלה ביקשה.
- **התקדמות ותיעוד** – כל קריאה מתקרבת לתנאי עצירה, בלי קריאות מיותרות. הערה לכל שיטה: מה היא מחזירה ומה התפקיד של כל פרמטר.

3 יעילות וסיבוכיות

3.1 סיבוכיות – מה סופרים

ניסוח	קוד
סיבוכיות זמן – כמה פעולות בסיסיות האלגוריתם מבצע, כפונקציה של **גודל הקלט** n. סופרים תמיד את **המקרה הגרוע ביותר**: זו הבטחה שהאלגוריתם לא **יעבור**. (מצגות 7–12, עמ' 47, 49)	$4+1+3n+1=3n+5 \Rightarrow O(n)$
סדר גודל O – מוחקים מקדמים וקבועים, ומשאירים רק את האיבר שגדל הכי מהר. (עמ' 49, 57)	$14n \to O(n)$ $\quad 100 \to O(1)$ $\frac{n^2-n}{2} \to O(n^2)$ $\quad n+\log n \to O(n)$
הסדר – מהמהיר לאיטי. $\log_2 n$ הוא כמה פעמים אפשר לחצות את n עד שנשאר 1: לולאה שחוצה (או מכפילה) רצה n או $\log n$ פעמים. (עמ' 51–52, 63)	$1 < \log n < \sqrt{n} < n < n\log n < n^2 < 2^n$ $n^2 = 10^6$, $n\log n \approx 10^4$, $\log n \approx 10 : n = 1000$
סיבוכיות מקום – רק הזיכרון **הנוסף** שהשיטה מקצה. מספר קבוע של משתנים – $O(1)$. מערך עזר בגודל m – $O(m)$. המערך שהתקבל כפרמטר לא נספר. רקורסיה – עומק הקריאות. (עמ' 65)	`int i, n;` `// O(1)` `int[] h = new int[m];` `// O(m)`

3.2 קריאת הסיבוכיות מהקוד

לולאות **מקוננות** – **כופלים**, לולאות **אחת אחרי השנייה** – **מחברים**, ומשאירים רק את האיבר הגדול. (עמ' 55, 57, 59)

צורת הקוד	כמה פעמים רץ הגוף	סיבוכיות
`for (int j = 0; j < n; j++)`	n	$O(n)$
`for (int j = 0; j < 100; j++)`	100 – קבוע, גם כשהוא גדול	$O(1)$
`for (i < n) ...` ואחריה `for (j < m) ...`	$n+m$	$O(n+m)$
`for (i < n)` ובתוכה `for (int j = i + 1; j < n; j++)`	$(n-1)+\cdots+1=\frac{n(n-1)}{2}$	$O(n^2)$
`for (int k = n; k > 0; k /= 2)` או `i *= 2`	$\log_2 n$	$O(\log n)$
`for (i < n)` ובתוכה `binarySearch(a, num - a[i])`	$n \cdot \log n$	$O(n\log n)$
`while (a[j] - a[i] > k) i++;` ובתוכה `for (j < n)`	`i` רק מתקדם: לכל היותר n **בכל הריצה**	$O(n)$, לא $O(n^2)$
`for (j < s)` n `s = s * 2` פעמים, ואחר כך	$s=2^n$ – מחשבים קודם את הגבול	$O(2^n)$

תבנית – "מה סיבוכיות הזמן והמקום של השיטות f1...f5"

1. **לכל לולאה – כמה פעמים היא רצה**, כפונקציה של n: המשתנה גדל ב-1 – n; מוכפל או נחצה – $\log$ של הגבול; עד קבוע – 1.
2. **מקוננות – כופלים, אחת אחרי השנייה – מחברים**, ומשאירים רק את האיבר הגדול.
3. **הגבול הוא משתנה?** מחשבים קודם את הערך שלו (s הוא 2^n, k הוא $\frac{n(n+1)}{2}$).
4. **מקום** – מערכים ש-new מקצה (לפי הגודל שלהם) ועומק רקורסיה. משתנים בודדים – $O(1)$.

דוגמה – (2016ב, שאלון 490, מועד 91, שאלה 6)

```
public static int f4(int n) {
    int k = 0;
    while (n > 0) { k += n; n--; }
    int res = 0;
    while (k > 0) { k /= 2; res++; }
    return res;
}
```

```
public static void f1(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            for (int k = n; k > 0; k = k / 2)
                System.out.println('*');
}
```

f4: הלולאה הראשונה רצה n פעמים, ובסופה $k=\frac{n(n+1)}{2}$. השנייה חוצה את k: $\log k \approx 2\log n$ פעמים. זמן $O(n+\log n)=O(n)$, מקום $O(1)$. ל-$n=1000$: $k=500500$, והשנייה רצה 19 פעמים.

f1: $\frac{n(n-1)}{2}$ זוגות i,j, ולכל זוג k נחצה – $O(n^2\log n)$, מקום $O(1)$. ל-$n=8$: $28\cdot 4=112$ כוכביות. באותה שאלה: מערך מונים `new int[m]` ושתי לולאות – זמן $O(n+m)$, מקום $O(m)$.

שימו לב: k גדול ($\approx n^2/2$), אבל לולאה שחוצה אותו רצה רק $\log(n^2)=2\log n$ פעמים. ולהפך: לולאה רגילה עד s כש-s הוא 2^n היא $O(2^n)$, לא $O(n)$.

מה המיון נותן בחינם – (2016ב, שאלון 490, מועד 82, שאלה 6)

ההגבלות בשאלה: אין מערך עזר, מערך ממוין נשאר ממוין, ואסור להשאיר חורים.

הפעולה	מערך ממוין	מערך לא ממוין
הכנסת איבר	$O(n)$ – מזיזים איברים	$O(1)$ – בסוף
האיבר ה-k בגודלו	$O(1)$ – בתא $k-1$	$O(n)$
האם x במערך	$O(\log n)$ – חיפוש בינרי	$O(n)$
הוצאת המינימום	$O(n)$ – סוגרים את החור	$O(n)$
זוג שסכומו x	$O(n)$ – שני מצביעים	$O(n^2)$
ההפרש הגדול ביותר בין שני איברים	$O(1)$ – אחרון פחות ראשון	$O(n)$ – מקסימום ומינימום

שאלות נוספות מאותו סוג: רשימה ממוינת מול לא ממוינת (2016ב, שאלון 490, מועד 87, שאלה 6)· מיון בחירה עם הכנסה (2025ב, שאלון 470, מועד 67, שאלה 5).

3.3 האלגוריתמים של הקורס

האלגוריתם	זמן – גרוע	זמן – הכי טוב	מקום	מצגות 7–12
חיפוש סדרתי	$O(n)$	$O(1)$	$O(1)$	עמ׳ 49
חיפוש בינרי (מערך ממוין)	$O(\log n)$: לכל היותר $1+\log_2 n$ סיבובים	$O(1)$	$O(1)$	עמ׳ 50–52
מיון בועות	$O(n^2)$	$O(n)$ – בגרסה עם swap, על מערך ממוין	$O(1)$	עמ׳ 53–55
מיון בחירה	$O(n^2)$	$O(n^2)$ – תמיד סורק את כל השאר	$O(1)$	עמ׳ 55–56
מיון הכנסה	$O(n^2)$	$O(n)$ – על מערך ממוין	$O(1)$	עמ׳ 56
דגל בשני צבעים (0 ו-1)	$O(n)$ – מעבר אחד	$O(n)$	$O(1)$	עמ׳ 57–58
מיון מהיר (quicksort)	$O(n^2)$	$O(n\log n)$	רקורסיה	עמ׳ 88–90
מיון מיזוג (רשימות)	$O(n\log n)$	$O(n\log n)$	רקורסיה	עמ׳ 126–131

שימו לב: מיון בתוך הפתרון עולה לפחות $O(n\log n)$ (ובמיונים של הקורס $O(n^2)$). ״קודם ממיינים״ משתלם רק כשהיעד עצמו לא קטן מזה.

3.4 תבנית – ״כתבו שיטה יעילה ככל הניתן״

כמעט תמיד שאלה 2 (25 נק׳). ״תשובה שאינה יעילה מספיק תקבל מעט נקודות בלבד״.

תבנית פתרון – נתון ← יעד ← דפוס ← סיבוכיות

1. **שלב ראשון – קוראים מה נתון:** ממוין? ערכים בטווח ידוע? אי-שליליים? מבנה מיוחד (סדרה חשבונית, מערך טווחים, מטריצה, ask)? מותר לשנות את המערך? k קבוע?
2. **קובעים את היעד – החסם התחתון:** אין סדר וכל איבר יכול לשנות את התשובה – $O(n)$; ממוין ומחפשים מקום אחד – $O(\log n)$. לפעמים השאלה אומרת (״רמז: ריבועית״).
3. **כותבים בראש פתרון ישיר ואת הסיבוכיות שלו** – מה הוא מחשב שוב ושוב? מה הוא לא מנצל?
4. **בוחרים דפוס** מהטבלה, וכותבים קוד מתועד.
5. **זמן וגם מקום, כל אחד עם נימוק:** ״זמן $O(n)$: כל מצביע רק מתקדם – יחד לכל היותר $2n$ צעדים, בכל צעד מספר קבוע של פעולות. מקום $O(1)$: רק `i`, `j`, `sum`, בלי מערך עזר ובלי רקורסיה״.

שימו לב: מערך עזר בגודל n הוא מקום $O(n)$, וכשהשאלה דורשת $O(1)$ זה פתרון לא יעיל. גם רקורסיה תופסת מקום.

מה נתון / מה מבקשים	היעד	הדפוס	סעיף	שאלות
ממוין (או מונוטוני), מחפשים ערך או גבול	$O(\log n)$	חיפוש בינרי	3.5	27
לא ממוין, התשובה תלויה בכל איבר	$O(n)$	מעבר אחד עם משתני מצב	3.6	17
ממוין, זוג / שלשה עם סכום או הפרש	$O(n^2)$ / $O(n)$	שני מצביעים	3.7	10
אי-שליליים, תת-מערך רציף	$O(n)$	חלון זז	3.8	8
שני מערכים ממוינים	$O(n+m)$	מצביע בכל מערך	3.9	4
לסדר לפי קבוצות, בלי מערך עזר	$O(n)$	חלוקה (דגל)	3.10	3
ערכים בטווח ידוע	$O(n)$	מערך מונים / סימון במערך	3.11	2
מטריצה ממוינת בשורות ובעמודות	$O(n+m)$	הליכת מדרגות	3.12	2

מה נתון / מה מבקשים	היעד	הדפוס	סעיף	שאלות
קוד נתון: "מה הוא עושה" ושפרו אותו	לפי הבזבוז	מעקב ← "מה" ← דפוס	3.13	11

3.5 חיפוש בינרי וגרסאותיו

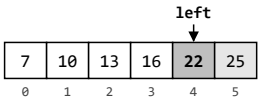
תבנית פתרון – מוצאים תכונה שמחלקת את המערך לשני צדדים

1. **שלב ראשון – מה ממוין, או מה מונוטוני?** לא תמיד הערכים: סכומי שכנים, "התא במקומו" ($a[i] = a[0] + i \cdot d$), ההפרש $a[i] - b[i]$, הריבוע x^2 מול n.
2. **מנסחים שאלה על `mid` שהתשובה עליה "לא... לא, כן... כן"**, ומחפשים את ה"כן" הראשון – הגבול.
3. **בכל סיבוב זורקים חצי:** `low = mid + 1` או `high = mid - 1` (או `high = mid` כש-`mid` עוד מועמד). הטווח חייב לקטון בכל סיבוב. בסוף `low` הוא הגבול – או ששומרים תשובה (`res = mid`) וממשיכים.
4. **סיבוכיות:** כל סיבוב חוצה את הטווח `low..high`, ובכל סיבוב מספר קבוע של פעולות – זמן $O(\log n)$, מקום $O(1)$. (עמ' 50–52)

דוגמה – האיבר החסר בסדרה חשבונית (2019א, שאלון 478, מועד 85, שאלה 2)

```
public static int binarySearch(int[] a, int x) {
    int low = 0, high = a.length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (a[mid] == x)
            return mid;
        if (a[mid] < x)
            low = mid + 1;     // מימין x
        else
            high = mid - 1;    // משמאל x
    }
    return -1;
}
```

```
int d = (arr[size - 1] - arr[0]) / size;
int left = 0, right = size - 1;
while (left <= right) {
    int mid = (left + right) / 2;
    if (arr[mid] == arr[0] + mid * d)
        left = mid + 1;   // במקום: החור מימין
    else
        right = mid - 1;  // זז: החור כאן או משמאל
}
return arr[0] + left * d;
```



מימין החיפוש הבינרי של המצגות, משמאל הגרסה של הגבול. חסר 19. ההפרש: $(25 - 7)/6 = 3$ – מחלקים ב-`size`, כי בסדרה המלאה יש `size + 1` איברים. התאים 0–3 במקומם, 4–5 זזו. `left` נעצר בתא 4: מחזירים $7 + 4 \cdot 3 = 19$.

הגרסה	השאלה על `mid` ← לאן הולכים	מבחנים
מופע ראשון / אחרון של x	ראשון: `a[mid] >= x` ← שמאלה (כולל `mid`). אחרון: `a[mid] <= x` ← ימינה. מספר המופעים = אחרון − ראשון + 1	2024א מועד 75 ש' 5
נקודת מפנה / פסגה	`a[mid] < a[mid+1]` ← ימינה, אחרת שמאלה (כולל `mid`)	2023א מועד 72; מצגות עמ' 62
איבר חסר בסדרה	התא "במקומו" ← ימינה	2019א 85, 2022א 67, 2023ב 99
איבר בודד בין זוגות	`mid` זוגי: `a[mid] == a[mid+1]` ← `low = mid + 2`, אחרת `high = mid`	2018ב מועד 83
שני מערכים מונוטוניים נפגשים	`a[mid] > b[mid]` ← ימינה; שוויון ← שומרים ושמאלה	2019ב 86, 2026א 89
חיפוש על התשובה (אין מערך)	שורש: `mid * mid <= n` ← ימינה (long!); `ask(...)` ← לפי התשובה	2026א 83, 2020ב 84, 2016א 92, 2021א 95
מערך טווחים Range	משווים ל-`getSmall()` / `getBig()` של `mid`	2017ב 93, 2020ב 96, 2026א 97
כמעט ממוין / אפסים באמצע	בודקים `mid-1..mid+1` וקופצים 2; מדלגים על אפסים: $O(K \log n)$	2018ב 91, 2022ב 96, 2023א 57
"ממוין לסירוגין"	מחפשים על `a[mid] + a[mid+1]` – הסכומים ממוינים	2016א מועד 83

שימו לב: `k = i + 1` במקום `i = k + 1`: כשנשארים שני תאים, `k` שווה ל-`i`, הטווח לא קטן, והלולאה לא נגמרת (2017א, שאלון 455, מועד 86, שאלה 6). וחיפוש בינרי שבכל סיבוב מדלג עד K תאים הוא $O(K \log n)$ – כותבים את זה, ואז "K קבוע ולכן $O(\log n)$".

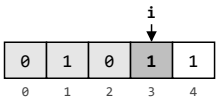
שאלות נוספות מאותו סוג: חציון של שני מערכים ממוינים (2024ב, שאלון 465, מועד 61, שאלה 5)· מערך חלקלק (2026ב, שאלון 468, מועד 98, שאלה 2)· דרך מערך הסכומים (2024א, שאלון 462, מועד 63, שאלה 2)· `k` הקרובים: חיפוש בינרי ואז שני מצביעים החוצה (2025ב, שאלון 470, מועד 91, שאלה 2).

3.6 מעבר אחד עם משתני מצב

תבנית פתרון – מה צריך לזכור כדי לא לחזור אחורה?

1. **שלב ראשון – הפתרון הישיר הוא שתי לולאות**: לכל j עוברים על כל מה שלפניו. **מה מכל זה באמת חשוב?** בדרך כלל ערך אחד או כמה: המינימום, שלושת הגדולים, מספר האפסים, אורך הרצף, הסכום עד כאן.
2. **שומרים אותו במשתנה ומעדכנים בכל צעד** – קודם משתמשים בו בשביל `a[i]`, ורק אחר כך מעדכנים אותו עם `a[i]`. אתחול: `a[0]`, `0` או `Integer.MIN_VALUE` – לפי מה שמוחזר כשאין תשובה.
3. **סיבוכיות:** לולאה אחת – $O(n)$, מספר קבוע של משתנים – $O(1)$. אי אפשר פחות: איבר שלא נקרא יכול לשנות את התשובה.

דוגמה – זוגות מכוניות מצטלבות (2016א, שאלון 481, מועד 87, שאלה 2)




```
int east = 0, pairs = 0;      // east: כמה 0 ראינו
for (int i = 0; i < a.length; i++) {
    if (a[i] == 0)
        east++;
    else
        pairs += east;  // 1 מצטלב עם כל 0 שלפניו
}
return pairs;
```

זוג = 0 שנמצא לפני 1. ל-1 שבתא 3 יש לפניו שני אפסים, ולכן הוא מוסיף 2. סך הכול $1+2+2=5$. במקום $O(n^2)$ – זמן $O(n)$, מקום $O(1)$.

מה זוכרים	מה זה פותר	מבחנים
המינימום עד עכשיו	ההפרש הגדול ביותר $a[j]-a[i]$, $j>i$ (3.13)	2024ב מועד 61
שלושת הגדולים ושני הקטנים	שלשה עם מכפלה מקסימלית / מינימלית – בלי מיון	2019א 93, 2022ב 91, 2025ב 67
אורך הרצף הנוכחי	תתי-מערכים עולים (מוסיפים len), סימנים מתחלפים, כולם $\leq$ num	2020א 85, 2022א 89, 2025א 95
הסכום הכולל, ואיבר אחד עובר צד בכל צעד	חלוקה לשני חלקים, מעבר בין שני כבישים, נקודת איזון	2019א 84, 2019ב 93, 2026ב 91
המספר הקטן שעוד אי אפשר להרכיב	מערך ממוין: res += a[i] כל עוד $a[i] \leq res$	2017א מועד 90
הערך המוחלט הקטן ביותר	איבר שמחלק את כולם – שני מעברים	2025א מועד 66

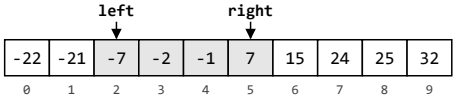
3.7 שני מצביעים במערך ממוין

תבנית פתרון – מצביע מכל קצה, ובכל צעד אחד זז

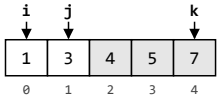
1. **שלב ראשון – `left = 0`, `right = n - 1`.** הסכום **גדול מדי** – `right--`: `a[right]` גדול מדי גם לכל `left` שאחרי. **קטן מדי** – `left++`. **שווה** – מצאנו. עוצרים כש-`left == right`.
2. **שלשה:** קובעים את האיבר הראשון בלולאה, ועל השאר מריצים שני מצביעים – $O(n^2)$ במקום $O(n^3)$.
3. **סיבוכיות:** בכל צעד מצביע אחד זז, ואף אחד לא חוזר – לכל היותר $n-1$ צעדים: זמן $O(n)$, מקום $O(1)$. (עמ' 60–61)

דוגמה – שלשות שסכומן קטן מ-num (2017א, שאלון 455, מועד 86, שאלה 2)

```
int left = 0, right = a.length - 1;
while (left < right) {
    int s = a[left] + a[right];
    if (s == num) return true;
    if (s > num) right--;    // גדול מדי
    else left++;             // קטן מדי
}
return false;
```




```
for (int i = 0; i < n - 2; i++)
    for (int j = i + 1, k = n - 1; j < k; )
        if (arr[i] + arr[j] + arr[k] >= num)
            k--;
        else {
            count += k - j;  // גם j+1..k מתאימים
            j++;
        }
```



מימין זוג שסכומו num (המצגות): ב-`0` = num המצביעים מגיעים ל-$0+7=7$. משמאל השלשות, num = 12: $1+3+7=11<12$, ולכן גם עם 4 ועם 5 – שלוש שלשות בבת אחת (`k - j` = 3). התשובה: 4. זמן $O(n^2)$, מקום $O(1)$.

שימו לב: סופרים את כל הזוגות בבת אחת (`count += k - j`), לא אחד-אחד – אחרת חוזרים ל-$O(n^3)$.

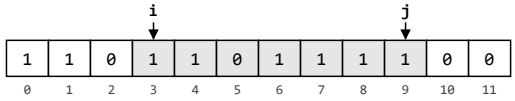
שאלות נוספות מאותו סוג: **באותו כיוון** – זוגות עם הפרש $\leq k$, `count += j - i` (2026ב, שאלון 468, מועד 89, שאלה 2)· הזזה מעגלית – מנקודת השבירה, עם `%` n (2017ב, שאלון 467, מועד 84, שאלה 2)· ממוצע x – מורידים איבר מהקצה (2016ב, שאלון 490, מועד 91, שאלה 2)· מי גשם – מטפלים בצד הנמוך (2018ב, שאלון 475, מועד 82, שאלה 2). עוד: שלשה פיתגורית (2017א, שאלון 455, מועד 87, שאלה 2)· שלשות שמכפלתן `num` (2020א, שאלון 470, מועד 87, שאלה 2)· זוג בסכום x במערך טווחים (2016ב, שאלון 490, מועד 82, שאלה 2).

3.8 חלון זז – תת-מערך רציף

תבנית פתרון – מרחיבים מימין, מכווצים משמאל

1. **שלב ראשון – כל המספרים אי-שליליים?** (או שסופרים משהו, כמו אפסים.) רק אז הוספת תא מגדילה את הסכום והוצאת תא מקטינה אותו.
2. **`j` מכניס תא בכל איטרציה, ו-`i` מוציא תאים משמאל כל עוד החלון לא חוקי.** (בחלון הקצר ביותר – מכווצים כל עוד הוא חוקי.)
   גודל קבוע k: `sum += a[i] - a[i - k]`.
3. **סיבוכיות:** כל תא נכנס פעם אחת ויוצא לכל היותר פעם אחת – $2n$ צעדים: זמן $O(n)$ (`while` בתוך `for`, אבל `i` רק מתקדם), מקום $O(1)$.

דוגמה – הרצף הארוך ביותר עם לכל היותר k אפסים (2021ב, שאלון 477, מועד 92, שאלה 2)




```
int i = 0, zeros = 0, best = 0;
for (int j = 0; j < a.length; j++) {
    if (a[j] == 0) zeros++;          // נכנס a[j]
    while (zeros > k) {              // יותר מדי אפסים
        if (a[i] == 0) zeros--;
        i++;                         // יוצא a[i]
    }
    best = Math.max(best, j - i + 1);
}
return best;
```

`k = 1`: החלון `a[3..9]` עם אפס אחד – אורך 7. כש-`j = 10` נכנס אפס שני, ו-`i` קופץ אל אחרי האפס שבתא 5. הפתרון הישיר (כל התחלה בנפרד) – $O(n^2)$.

שאלות נוספות מאותו סוג: חלון בגודל k (2020ב, שאלון 478, מועד 81, שאלה 2)· הקצר ביותר שסכומו גדול מ-k (2021ב, שאלון 477, מועד 60, שאלה 2)· סכום `target` (2023ב, שאלון 468, מועד 81, שאלה 2)· "מה עושה `secret`" (2025א, שאלון 463, מועד 72, שאלה 5).

3.9 שני מערכים ממוינים – מצביע בכל מערך

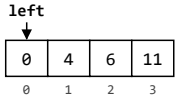
תבנית פתרון – בכל צעד מקדמים את המצביע שכבר סיים את תפקידו

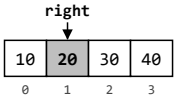
1. **שלב ראשון – מצביע על כל מערך.** מיזוג – שניהם מההתחלה. סכום קרוב ל-x – מתחילת a ומסוף b.
2. **מקדמים את המצביע שלא יעזור יותר:** במיזוג – של הקטן ; בסכום – גדול מ-x ← `right--`, אחרת `left++` ; בשלושה מערכים – של הקטן מבין השלושה.
3. **סיבוכיות:** בכל צעד מצביע אחד זז – זמן $O(n+m)$. מקום $O(1)$, ובמיזוג למערך חדש – $O(n+m)$.

דוגמה – הזוג שסכומו הכי קרוב ל-x (2017ב, שאלון 467, מועד 85, שאלה 2)

```
int[] c = new int[a.length + b.length];
int i = 0, j = 0, k = 0;
while (i < a.length && j < b.length)
    if (a[i] <= b[j])
        c[k++] = a[i++];
    else
        c[k++] = b[j++];
while (i < a.length) c[k++] = a[i++];
while (j < b.length) c[k++] = b[j++];
```

```
int left = 0, right = b.length - 1;
int diff = Integer.MAX_VALUE, ra = 0, rb = 0;
while (left < a.length && right >= 0) {
    int s = a[left] + b[right];
    if (Math.abs(s - x) < diff) {   // קרוב יותר
        diff = Math.abs(s - x);
        ra = a[left]; rb = b[right];
    }
    if (s > x) right--;    // סיים b[right]
    else left++;           // סיים a[left]
}
```





מימין מיזוג, משמאל הדוגמה: x = 18 – $0+40$, $0+30$, $0+20$ … הזוג 0 ו-20. זמן $O(n+m)$ במקום $O(n \cdot m)$, מקום $O(1)$.

שאלות נוספות מאותו סוג: (2025ב, שאלון 473, מועד 78, שאלה 2) · ההפרש הקטן ביותר בין שני מערכים: ממיינים ומקדמים את הקטן (2024ב, שאלון 465, מועד 99, שאלה 2) · השלשה הקרובה משלושה מערכים (2025א, שאלון 463, מועד 95, שאלה 5).

3.10 חלוקה – דגל בשניים או בשלושה צבעים

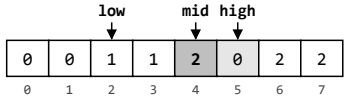
תבנית פתרון – אזורים מסודרים ומצביע לסוף כל אזור

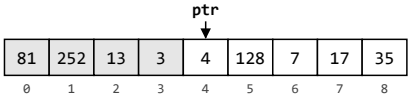
1. **שלב ראשון – מגדירים אזורים:** לפני `low` – אפסים, בין `low` ל-`mid` – אחדים, אחרי `high` – שתיים, ובאמצע (`mid..high`) עוד לא ידוע.
2. **בכל צעד בודקים את `a[mid]` ומחליפים אותו לאזור שלו:** 0 ← עם `low`, ו-`low++`, `mid++`; 1 ← `mid++`; 2 ← עם `high`, ו-`high--` (בלי `mid++`: התא שהגיע עוד לא נבדק). בשני צבעים – בלי האזור האמצעי.
3. **k קבוצות (k קבוע):** מעבר לכל קבוצה, עם `ptr` לסוף החלק המסודר – $O(k \cdot n) = O(n)$.
4. **סיבוכיות:** כל צעד מקטין את האזור הלא ידוע – זמן $O(n)$; רק החלפות במקום – מקום $O(1)$. (עמ' 57–58)

דוגמה – מיון לפי שארית k (2018א, שאלון 483, מועד 91, שאלה 2)

```
int low = 0, mid = 0, high = a.length - 1;
while (mid <= high) {
    if (a[mid] == 0)
        swap(a, low++, mid++);
    else if (a[mid] == 1)
        mid++;
    else
        swap(a, mid, high--);
}
```

```
int ptr = 0;                          // סוף החלק המסודר
for (int i = 0; i < k; i++)           // שארית 0, 1, ..., k-1
    for (int j = ptr; j < a.length; j++)
        if (a[j] % k == i) {
            int temp = a[ptr];
            a[ptr] = a[j];
            a[j] = temp;
            ptr++;
        }
```





מימין דגל בשלושה צבעים: 0–1 אפסים, 2–3 אחדים, 4–5 עוד לא נבדקו, 6–7 שתיים. משמאל `k = 10`,
{81, 3, 7, 128, 4, 252, 13, 17, 35} אחרי המעבר של שארית 3: התאים 0–3 מסודרים ו-`ptr = 4`. בסוף:
{128, 17, 7, 35, 81, 252, 13, 3, 4}. k מעברים של $O(n)$ ו-k קבוע – זמן $O(n)$, מקום $O(1)$. מערך מונים בגודל k לא מספיק: צריך להזיז את המספרים עצמם.

שאלות נוספות מאותו סוג: (2022ב, שאלון 477, מועד 86, שאלה 2) · מערך גלי – מעבר על האינדקסים האי-זוגיים והחלפה עם שכן (2024ב, שאלון 468, מועד 87, שאלה 2).

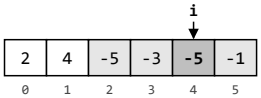
3.11 ערכים בטווח ידוע – מערך מונים, או המערך עצמו כטבלה

תבנית פתרון – הערך הוא אינדקס

1. **שלב ראשון – הערכים בין 0 ל-$m-1$?** מערך מונים: `++h[a[j]]`. זמן $O(n+m)$, מקום $O(m)$ – ו-$O(1)$ רק כש-m קבוע.
2. **נדרש מקום $O(1)$ והערכים בין 1 ל-n?** משתמשים **במערך עצמו**: לכל ערך v יש תא משלו. מסמנים אותו בסימן מינוס (וקוראים ערכים עם `Math.abs`), או מעבירים את v ל"ביתי" שלו, התא $v-1$, בהחלפות.
3. **סיבוכיות:** כל תא מסומן (או מוחלף למקומו) פעם אחת – זמן $O(n)$, מקום $O(1)$.

דוגמה – מספר כפול, n תאים וערכים 1 עד n–1 (2022א, שאלון 476, מועד 74, שאלה 2)

```
for (int i = 0; i < a.length; i++) {
    int v = Math.abs(a[i]);
    if (a[v] < 0)          // התא v כבר מסומן
        return v;
    a[v] = -a[v];          // "ראינו את v"
}
return -1;
```



{1, 5, 3, 5, 4, 2}: מסמנים את התאים 2, 4, 5, 3. ב-`i = 4` הערך `a[5]` – 5 כבר שלילי: מחזירים 5.

שאלות נוספות מאותו סוג: החיובי הקטן ביותר שחסר – כל v לתא $v-1$ (2024א, שאלון 462, מועד 75, שאלה 2) · מערך מונים `new int[m]` בטבלת סיבוכיות (2016ב, שאלון 490, מועד 91, שאלה 6).

3.12 מטריצה ממוינת – הליכת מדרגות

תבנית פתרון – מתחילים בפינה שבה שני הכיוונים "מתנגדים"

1. **שלב ראשון – פינה ימנית עליונה** (שורות ועמודות עולות): שמאלה קטן, למטה גדל. (במטריצה יורדת – פינה שמאלית תחתונה.)
2. **התא שלילי (או קטן מ-x)** – כל השורה משמאלו שלילית: סופרים `j + 1` בבת אחת ו-`i++`. **לא שלילי** – כל העמודה מתחתיו לא שלילית: `j--`.
3. **סיבוכיות:** כל צעד מוחק שורה או עמודה – לכל היותר $n+m$ צעדים: זמן $O(n+m)$, מקום $O(1)$.

דוגמה – כמה מספרים שליליים (2019ב, שאלון 466, מועד 83, שאלה 2)

```
int i = 0, j = arr[0].length - 1, count = 0;
while (i < arr.length && j >= 0) {
    if (arr[i][j] < 0) {
        count += j + 1;    // שליליים arr[i][0..j]
        i++;
    }
    else
        j--;               // לא שליליים arr[i..][j]
}
return count;
```

	0	1	2	3	4
0	-9	-7	-4	**-1**	**2**
1	-8	-5	**-2**	**3**	6
2	-6	**-3**	**1**	4	8
3	**-2**	**0**	5	7	9

המסלול (באפור): $10 = 1+2+3+4$ שליליים ב-8 צעדים ($9 = n+m \geq$). מעבר על כל התאים: $O(n \cdot m)$.

שאלות נוספות מאותו סוג: "מה עושה `find`" – אותה הליכה מהפינה השמאלית התחתונה (2024ב, שאלון 465, מועד 99, שאלה 5).

3.13 תבנית – "מה עושה השיטה, מה הסיבוכיות, כתבו שיטה יעילה יותר"

תבנית פתרון – מעקב ← "מה" ← סיבוכיות ← הבזבוז

1. **שלב ראשון – מריצים על דוגמה קטנה** בטבלת מעקב (`i`, `j`, המשתנים), עד שרואים מה נשמר.
2. **מנסחים מה השיטה עושה – לא איך.** "השיטה מחזירה את...", במונחים של המערך, בלי לולאות ומשתנים. מוסיפים **מקרי קצה**: מערך ריק או `null`, איבר אחד, אין תשובה (מה מוחזר?), גבולות (`high` נכלל או לא), מספרים שליליים, שוויון.
3. **סיבוכיות הקוד הנתון** – לפי 3.2, זמן ומקום.
4. **איפה הבזבוז?** מה מחושב מחדש בכל איטרציה – והוא הדפוס (סכום רץ, מינימום עד עכשיו, חלון).
5. **השיטה החדשה מחזירה בדיוק אותו ערך**, גם במקרי הקצה.

דוגמה – (2024ב, שאלון 465, מועד 61, שאלה 2)

```
public static int what(int[] a) {
    if ((a == null) || (a.length == 0))
        return 0;
    int m = 0;
    for (int i = 0; i < a.length; i++)
        for (int j = i + 1; j < a.length; j++)
            if (a[j] - a[i] > m)
                m = a[j] - a[i];
    return m;
}
```

```
public static int whatEff(int[] a) {
    if (a == null || a.length == 0)
        return 0;
    int min = a[0];    // הקטן ביותר עד עכשיו
    int m = 0;         // ההפרש הטוב ביותר
    for (int i = 1; i < a.length; i++) {
        if (a[i] - min > m)
            m = a[i] - min;
        if (a[i] < min)
            min = a[i];
    }
    return m;
}
```

מה: השיטה מחזירה את ההפרש הגדול ביותר $a[j]-a[i]$ כאשר $j>i$ (איבר מאוחר פחות איבר מוקדם). אם אין הפרש חיובי (מערך יורד, איבר אחד, מערך ריק או `null`) – 0. הסדר חשוב: ב-{10, 20, 1, 2} היא מחזירה 10, לא 19.

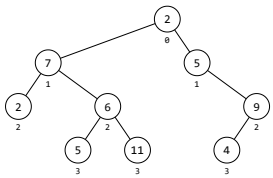
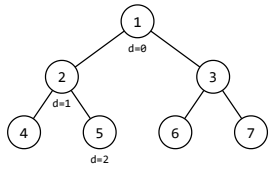
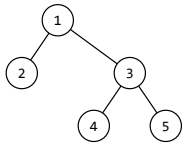
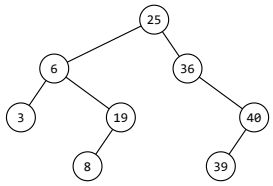
סיבוכיות: $O(n^2)$, מקום $O(1)$. **הבזבוז:** לכל j רק בן זוג אחד חשוב – המינימום שלפניו. **שיפור:** זמן $O(n)$, מקום $O(1)$.

שימו לב: `m` מתחיל ב-0 כמו ב-`what`. אם מתחילים ב-`Integer.MIN_VALUE`, מערך יורד יחזיר מספר שלילי, ו-`what` מחזירה עליו 0.

שאלות נוספות מאותו סוג: (2018א, שאלון 483, מועד 85, שאלה 2) · (2018א, שאלון 486, מועד 87, שאלה 2) · (2025א, שאלון 463, מועד 72, שאלה 5) · (2024א, שאלון 462, מועד 75, שאלה 5).

4 עצים בינאריים

4.1 מושגים וספירות

ניסוח	שרטוט	קוד
הצומת – המחלקה Node: מספר, בן שמאלי ובן ימני. בן שחסר הוא null. **עץ בינרי** הוא ריק, או שורש עם תת-עץ שמאלי ותת-עץ ימני, וכל אחד מהם עץ בינרי. לכן כמעט כל שיטה על עץ היא רקורסיה. (מצגות 7–12, עמ׳ 144–145)		`public class Node {` `  private int _number;` `  private Node _leftSon, _rightSon;` `  public int getNumber()  {...}` `  public Node getLeftSon()  {...}` `  public Node getRightSon() {...}` `  // setNumber, setLeftSon, setRightSon` `}`
שורש – הצומת היחיד בלי אב (2 העליון). **עלה** – צומת בלי בנים (2, 5, 11, 4). **בן / אב / אחים** – 7 ו-5 אחים, בנים של השורש. **צאצא / אב קדמון** – לאורך מסלול למטה / למעלה (11 צאצא של 7). **תת-עץ** – צומת עם **כל** הצאצאים שלו. (עמ׳ 143–144) **רמה (עומק)** של צומת – מספר הקשתות מהשורש אליו (המספר מתחת לצומת; השורש ברמה 0). **גובה** העץ – מספר הקשתות במסלול הארוך ביותר מהשורש לעלה; כאן 3.		
ספירות – ברמה d יש לכל היותר 2^d צמתים. לכן בעץ בגובה h יש $h+1 \le n \le 2^{h+1}-1$ צמתים, והגובה של עץ עם n צמתים הוא בין $\lfloor \log_2 n \rfloor$ (מאוזן) ל-$n-1$ (שרשרת).		$1+2+4=2^3-1=7$
עץ מלא – לכל צומת 0 או 2 בנים (אין בן יחיד), כמו בציור. אם k צמתים עם 2 בנים, אז $n=2k+1$ (אי-זוגי) ויש $k+1$ עלים. **עץ מושלם** – מלא, וכל העלים באותה רמה (העץ בשורה הקודמת): $n=2^{h+1}-1$, כלומר 1, 3, 7, 15... **עץ שלם** – כל הרמות מלאות חוץ מהאחרונה, שמתמלאת משמאל.		
עץ חיפוש בינרי – בכל צומת: **כל** תת-העץ השמאלי קטן ממנו, **וכל** תת-העץ הימני גדול ממנו (ערך שווה – ימינה). המינימום הוא הצומת השמאלי ביותר, המקסימום הימני ביותר. (עמ׳ 149, 152–153)		

שימו לב: יש מבחנים שסופרים את השורש ברמה **1** ואת הגובה **בצמתים** (null מחזיר 0, לא -1). קוראים מה השיטה מחזירה, לא את המילה ״גובה״.

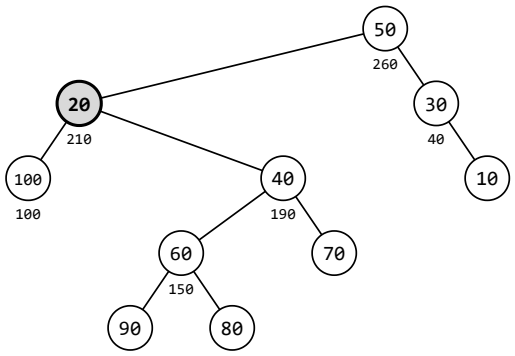
4.2 "מה עושה השיטה" – מעקב על העץ והכללה

תבנית פתרון – מעקב מלמטה למעלה, ואז משפט אחד

1. **קוראים את הקוד ומוציאים נוסחה:** מה מחזיר `null`, מה מחזיר עלה, ואיך צומת מחשב מהתשובות של הבנים. למשל `f(t) = t + max(f(L), f(R))`. שיטה שקוראת לשיטה אחרת – מתחילים מהפנימית.
2. **מעתיקים את העץ** בדיוק (צדדים, בנים חסרים), וכותבים ליד כל `null` את מה שהוא מחזיר.
3. **מטפסים מלמטה למעלה:** כותבים ליד כל עלה את מה שהוא מחזיר, ואז ליד כל צומת ששני הבנים שלו כבר כתובים. כך לא מחזיקים מחסנית קריאות בראש.
4. **ערך שיורד למטה** (פרמטר כמו `d + 1`, `st + "0"`, `sum + v`) – כותבים מלמעלה למטה. **הדפסה** – לפי מקום ההדפסה: לפני הקריאות = תחילי, ביניהן = תוכי, אחריהן = סופי.
5. **התשובה** – מה שכתוב ליד השורש. אבל אם השיטה לוקחת max / min על כל הצמתים, התשובה היא הגדול ביותר **בכל העץ**.
6. **מכלילים:** קוראים את הנוסחה במילים ומחפשים תבנית מוכרת (4.3). כותבים **מה** השיטה מחזירה, לא איך, ומקרי קצה: עץ ריק, צומת אחד, בן יחיד, ערך שלא בעץ.

דוגמה – (2026א, שאלון 462, מועד 97, שאלה 3)

```
public static int something(Node node) {
    if (node == null) return 0;
    int left  = something(node.getLeftSon());
    int right = something(node.getRightSon());
    return node.getNumber() + Math.max(left, right);
}
public static int secret(Node root) {
    if (root == null) return 0;
    int current = root.getNumber()
                + something(root.getLeftSon())
                + something(root.getRightSon());
    int left  = secret(root.getLeftSon());
    int right = secret(root.getRightSon());
    return Math.max(current, Math.max(left, right));
}
```



- **something:** הנוסחה `v + max(left, right)` ו-`null` מחזיר 0. התגים בציור, מהעלים למעלה: $60 + 90 = 150$, $40 + 150 = 190$, $20 + 190 = 210$, ובשורש $50 + 210 = 260$.
- **secret:** בכל צומת `current = v + something(L) + something(R)`. ב-20: $20 + 100 + 190 = 310$, ובשורש רק $50 + 210 + 40 = 300$. מודפס 310.
- **מה כותבים:** `something` מחזירה את הסכום הגדול ביותר של מסלול מהשורש עד עלה (עץ ריק – 0, צומת אחד – הערך שלו). `secret` מחזירה את סכום המסלול הכבד ביותר בעץ, בין שני צמתים כלשהם (עץ ריק – 0).

שימו לב: התשובה לא תמיד בשורש: ב-`secret` היא ב-20 (310), והשורש נותן רק 300.

שימו לב: הערך של `null`: 0 מול `Integer.MIN_VALUE` מול -1. עם ערכים שליליים, 0 "קוטע" מסלול בצומת שאינו עלה. וערך כמו 0 או -1 שמסמן גם "לא נמצא" או "לא מאוזן" מתנגש בערך אמיתי (2020ב, שאלון 478, מועד 81, שאלה 3).

שימו לב: בדיקה מול הבנים בלבד אינה בדיקה מול כל תת-העץ. ו-`t.getLeftSon().getNumber()` בלי בדיקת `null` נופל ב-`NullPointerException` (2016ב, שאלון 490, מועד 91, שאלה 3).

שאלות נוספות מאותו סוג: (2017א, שאלון 455, מועד 86, שאלה 3), (2017ב, שאלון 467, מועד 93, שאלה 3), (2018א, שאלון 486, מועד 87, שאלה 3), (2024ב, שאלון 465, מועד 99, שאלה 3).

4.3 כתיבת שיטה רקורסיבית – והתבניות שחוזרות במבחנים

תבנית – שיטה רקורסיבית על עץ

1. **`null` קודם:** מה התשובה לעץ ריק? ספירה וסכום – `0`, גובה – -1, מקסימום – `Integer.MIN_VALUE`, "כולם מקיימים" – `true`, "קיים" – `false`.
2. **עלה** – רק אם הוא מקרה מיוחד (ספירת עלים, מסלול שנגמר בעלה).
3. **קוראים לשני הבנים ומחברים** עם הצומת עצמו: `+`, `Math.max`, `&&`, `||`.
4. **מידע שיורד למטה** (עומק, סכום המסלול, גבולות) – פרמטר נוסף, ושיטה ציבורית שקוראת עם הערך ההתחלתי.
5. **בודקים** על עץ ריק, צומת אחד ושרשרת.

ניסוח	קוד
מספר הצמתים (ובאותה צורה **סכום**: `t.getNumber() + ...`) (2016א, שאלון 481, מועד 83, שאלה 3)	`if (t == null) return 0;` `return 1 + count(t.getLeftSon()) + count(t.getRightSon());`
מספר העלים (עמ' 148)	`if (t == null) return 0;` `if (t.getLeftSon() == null && t.getRightSon() == null) return 1;` `return leaves(t.getLeftSon()) + leaves(t.getRightSon());`
גובה בקשתות. עם `return 0` ל-`null` – מספר הרמות (2017ב, שאלון 467, מועד 85, שאלה 3)	`if (t == null) return -1;` `return 1 + Math.max(height(t.getLeftSon()), height(t.getRightSon()));`
הערך הגדול בעץ (לא עץ חיפוש: בודקים את כולו) (עמ' 152)	`if (t == null) return Integer.MIN_VALUE;` `return Math.max(t.getNumber(),` `Math.max(max(t.getLeftSon()), max(t.getRightSon())));`
האם x בעץ – `||` עוצר במציאה הראשונה, משמאל לפני ימין (עמ' 146)	`if (t == null) return false;` `return t.getNumber() == x || contains(t.getLeftSon(), x)` `|| contains(t.getRightSon(), x);`
העומק של x – הפרמטר d יורד, -1 = לא נמצא (2016ב, שאלון 490, מועד 87, שאלה 3)	`if (t == null) return -1;` `if (t.getNumber() == x) return d;` `int left = depth(t.getLeftSon(), x, d + 1);` `if (left != -1) return left;` `return depth(t.getRightSon(), x, d + 1);  // depth(root, x, 0)`
המסלול הכבד מהשורש לעלה – הצומת ועוד הבן הטוב יותר (ערכים חיוביים) (2017א, שאלון 455, מועד 86, שאלה 3)	`if (t == null) return 0;` `return t.getNumber() + Math.max(path(t.getLeftSon()),` `path(t.getRightSon()));`
האם עץ חיפוש – הגבולות יורדים וזוכרים את **כל** האבות (2025ב, שאלון 473, מועד 78, שאלה 3)	`if (t == null) return true;  // isBST(root, MIN_VALUE, MAX_VALUE)` `int v = t.getNumber();` `return v >= a && v < b && isBST(t.getLeftSon(), a, v)` `&& isBST(t.getRightSon(), v, b);`
השוואת שני עצים – אותו מבנה ואותם ערכים	`if (a == null || b == null) return a == b;` `return a.getNumber() == b.getNumber()` `&& same(a.getLeftSon(), b.getLeftSon())` `&& same(a.getRightSon(), b.getRightSon());`
תת-העץ הגדול ביותר שמקיים תכונה – אם כל תת-העץ מקיים, סופרים אותו; אחרת, הטוב משני הצדדים (2021א, שאלון 0, מועד 70, שאלה 3)	`if (allEven(t)) return count(t);  // allEven(null) = true` `return Math.max(best(t.getLeftSon()), best(t.getRightSon()));`

שימו לב: מסלול **מצומת לצומת** (לא דרך השורש): בכל צומת `v + path(L) + path(R)`, ולוקחים את המקסימום על כל הצמתים – כמו `secret` ב-4.2.

4.4 "שנו את העץ / ציירו עץ ש..."

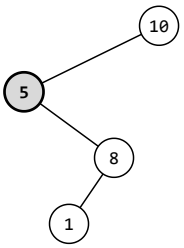
תבנית פתרון – לבנות עץ לפי מה שהשיטה צריכה להחזיר

1. **קודם המשמעות:** משפט אחד על מה השיטה מחזירה (4.2). בלעדיו רק מנחשים.
2. **"שינוי מינימלי":** מוצאים את הצומת שקובע את התוצאה (הבדיקה שנכשלה, או הצומת שנתן את המקסימום) ומשנים שם ערך אחד. סופרים פעולות: הוספה, מחיקה (עם כל הבנים שנמחקים), שינוי ערך. בודקים שלא נשברה בדיקה אחרת, ושעץ חיפוש נשאר עץ חיפוש.
3. **"השלימו ערכים":** ממלאים מלמטה למעלה, מצמתים שכל הבנים שלהם ידועים ועד השורש.
4. **"האם קיים עץ ש-f כן ו-g לא?":** בודקים אם אחת גוררת את השנייה. אם כל עץ ש-g מאשרת גם f מאשרת, השילוב "g כן, f לא" **לא קיים**, ומסבירים למה. לשאר בונים את העץ הקטן ביותר (שרשרת, כשמבקשים עומק).
5. **מריצים שוב** את השיטה על העץ שציירתם.

דוגמה – (2025ב, שאלון 473, מועד 78, שאלה 3)

f – כל צומת מול **הבנים** שלו: שמאלי קטן ממש, ימני גדול או שווה.

g – הגבולות של "האם עץ חיפוש" (4.3).



f כן, g לא: השורש 10, בן שמאלי 5, ל-5 בן ימני 8, ול-8 בן שמאלי 1. כל זוג אב-בן תקין ($5 < 10$, $8 \leq 5$, $1 < 8$), אבל 1 בתת-העץ **הימני** של 5 וקטן ממנו.

f לא, g כן – לא קיים: עץ חיפוש מקיים את הכלל גם מול הבנים, ולכן כש-g מחזירה true גם f מחזירה true.

שימו לב: "מינימלי" נספר בפעולות: מחיקת צומת עם בנים היא כמה פעולות. ושינוי ערך בעץ חיפוש חייב לשמור על הסדר.

שאלות נוספות מאותו סוג: (2016ב, שאלון 490, מועד 91, שאלה 3), (2020ב, שאלון 478, מועד 81, שאלה 3), (2023ב, שאלון 468, מועד 83, שאלה 3), (2025ב, שאלון 470, מועד 91, שאלה 3).

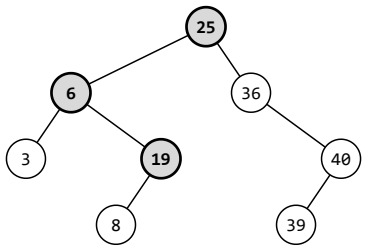
4.5 עץ חיפוש בינרי

ניסוח	קוד
חיפוש – משווים לצומת ויורדים לצד **אחד** (עמ' 150)	`if (root == null) return false;` `if (root.getNumber() == num) return true;` `if (root.getNumber() > num)` `  return searchInBST(root.getLeftSon(), num);` `return searchInBST(root.getRightSon(), num);`
הכנסה – יורדים כמו בחיפוש, ונכנסים במקום ה-null הראשון. ערך שווה – ימינה. סדר ההכנסה קובע את צורת העץ: ממוין ← שרשרת. (עמ' 150–151)	`if (x < bst.getNumber()) {` `  if (bst.getLeftSon() == null) bst.setLeftSon(new Node(x));` `  else insert(bst.getLeftSon(), x);` `}` `else ...  // אותו דבר, ימינה`
מינימום / מקסימום – הולכים שמאלה / ימינה עד הסוף (עמ' 153)	`if (root.getRightSon() == null) return root.getNumber();` `return maxInSearchTree(root.getRightSon());`
מחיקה – עלה: מנתקים. בן אחד: הבן עולה למקומו. שני בנים: מחליפים בעוקב (המינימום של תת-העץ הימני) ומוחקים אותו.	
סיבוכיות – חיפוש, הכנסה, מחיקה, מינימום: מסלול אחד, $O(h)$. מאוזן: $O(\log n)$. שרשרת: $O(n)$. מעבר על כל העץ: $O(n)$. (עמ' 151)	

תבנית פתרון – שיטה על עץ חיפוש: מסלול אחד למטה

1. **משווים לצומת:** אם הוא קטן מ-x, אז **כל** תת-העץ השמאלי שלו קטן מ-x – אפשר לטפל בו בבת אחת (לספור, לדלג) ולהמשיך ימינה. אם גדול – כל הימני בחוץ, ממשיכים שמאלה.
2. **מסמנים את המסלול** בעץ, ובכל פנייה ימינה שואלים: מה נוסף כאן?
3. **סיבוכיות:** מסלול אחד – $O(h)$; שיטה שקוראת לספירה של תת-עץ מוסיפה את גודלו.
4. **מה כותבים:** משמעות במונחי הסדר – "כמה ערכים קטנים מ-x", "הגדול ביותר שקטן מ-num", "הקודם בסדר התוכי".

דוגמה – (2016א, שאלון 481, מועד 83, שאלה 3)



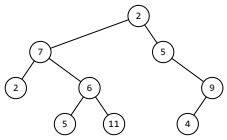

```
public static int g(Node t, int x) {
    if (t == null) return 0;
    if (t.getNumber() > x)
        return g(t.getLeftSon(), x);
    if (t.getNumber() < x)
        return f(t.getLeftSon()) + 1 + g(t.getRightSon(), x);
    return f(t.getLeftSon());       // f = מספר הצמתים
}
```

g(root, 20): ב-25 שמאלה. ב-6: $f(3)+1=2$, ימינה. ב-19: $f(8)+1=2$. סה"כ **4** (3, 6, 8, 19). **מה כותבים:** g מחזירה את מספר הצמתים שערכם קטן ממש מ-x (עץ ריק – 0).

שימו לב: "מינימום" בעץ חיפוש הוא השמאלי ביותר – לא בהכרח עלה. **ועץ חיפוש אינו מאוזן** מעצמו: $O(\log n)$ רק אם הוא מאוזן.

שאלות נוספות מאותו סוג: (2024א, שאלון 462, מועד 69, שאלה 3), (2019ב, שאלון 466, מועד 86, שאלה 3), (2019א, שאלון 478, מועד 93, שאלה 3), (2026ב, שאלון 468, מועד 98, שאלה 3).

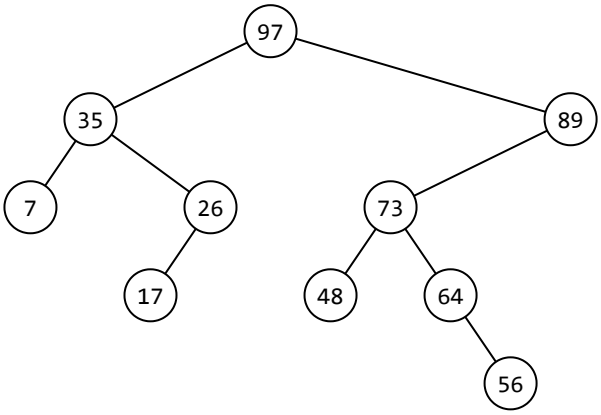
4.6 סריקות ושחזור עץ

ניסוח	שרטוט	קוד
תחילי (PreOrder) – שורש, שמאל, ימין. השורש **ראשון**. (עמ' 146–148)		`if (t != null) {  // preOrder(Node t)` `  System.out.print(t.getNumber() + " ");` `  preOrder(t.getLeftSon());` `  preOrder(t.getRightSon());` `}` `2 7 2 6 5 11 5 9 4`
תוכי (InOrder) – שמאל, שורש, ימין: ההדפסה **בין** שתי הקריאות. השורש מפריד בין השמאלי לימני. בעץ חיפוש – ממוין.		`2 7 5 6 11 2 5 4 9`
סופי (PostOrder) – שמאל, ימין, שורש: ההדפסה **אחרי** שתי הקריאות. השורש **אחרון**.		`2 5 11 6 7 4 9 5 2`
לפי רמות – רמה אחרי רמה, משמאל לימין. במבחנים: שיטה שמדפיסה רמה x (השורש ברמה 1), ולולאה על כל הרמות (2019ב, שאלון 466, מועד 83, שאלה 3).		`if (node == null) return;  // what(Node node, int x)` `if (x == 1) System.out.print(node.getNumber() + " ");` `else {` `  what(node.getLeftSon(), x - 1);` `  what(node.getRightSon(), x - 1);` `}  // for (x = 1; x <= levels; x++) what(root, x);` `2 7 5 2 6 9 5 11 4`

תבנית פתרון – שחזור עץ משתי סריקות (אחת מהן תוכית)

1. **השורש:** הראשון בסדר תחילי, או האחרון בסדר סופי.
2. **מוצאים אותו בסדר התוכי:** מה שמשמאלו – תת-העץ השמאלי (k ערכים), מה שמימינו – הימני.
3. **מחלקים גם את הסריקה השנייה:** k הערכים הראשונים (בתחילי – אחרי השורש) הם השמאלי, והשאר הימני.
4. **חוזרים** על כל חלק עד ערך בודד. חלק ריק = אין בן בצד הזה.
5. **בודקים:** סורקים את העץ שקיבלתם, ומקבלים בדיוק את שתי הסריקות שבשאלה.

דוגמה – (2021א, שאלון 0, מועד 85, שאלה 3)



סופי: 97 89 73 64 56 48 35 26 17 7

תוכי: 89 56 64 73 48 | 97 | 26 17 35 7

השורש **97**. משמאל 4 ערכים: בסופי 35 26 17 7, השורש שלהם **35** (האחרון), ובתוכי 26 17 | 35 | 7. מימין: 89 73 64 56 48, השורש **89**, ובתוכי אין כלום מימינו – אין לו בן ימני. וכך הלאה.

תחילי: 56 64 48 73 89 17 26 7 35 97

- **עץ חיפוש + סריקה אחת:** התוכי הוא הערכים ממוינים, ולכן מספיק סופי (או תחילי): האחרון הוא השורש, הקטנים ממנו משמאל והגדולים מימין (2021ב, שאלון 477, מועד 92, שאלה 3).
- **מחרוזת סוגריים:** (5(50)) – מה שלפני המספר הוא תת-העץ **השמאלי**; בן ימני היה ((5(50)) (2022ב, שאלון 477, מועד 86, שאלה 3).
- **כתובות 0/1:** כל שורה היא צומת – 0 שמאלה, 1 ימינה, והשורש היא השורה הריקה (2018ב, שאלון 472, מועד 83, שאלה 3).

שימו לב: תחילי + סופי **לא** קובעים עץ אחד: בן יחיד יכול להיות משמאל או מימין. וסופי ממוין אינו אומר שזה עץ חיפוש – בונים רק מהסריקות.

שאלות נוספות מאותו סוג: (2018ב, שאלון 475, מועד 82, שאלה 3), (2025ב, שאלון 470, מועד 67, שאלה 3), (2018ב, שאלון 472, מועד 91, שאלה 3).

4.7 טענות נכון / לא נכון

ניסוח
"בסריקה תוכית השורש לא יכול להיות ראשון ולא אחרון" – **לא.** שורש בלי בן שמאלי הוא ראשון: 1 עם בן ימני 2 נותן 2 1. (2021א, שאלון 0, מועד 85, שאלה 3)
"צומת שמאלי יותר נסרק לפני צומת ימני יותר, בכל סריקה" – **לא**, רק בתוכי. לשורש 1 בנים 2 ו-3, ל-2 בן ימני 5, ול-3 בן שמאלי 6. בתחילי 6 3 5 2 1 1: 3 נסרק לפני 6, שמשמאלו. (שם)
"חיפוש בעץ חיפוש עם N צמתים דורש לכל היותר $\log_2 N$ השוואות" – **לא**, $O(h)$. הכנסה של 1…8 בסדר עולה נותנת שרשרת: 8 השוואות, ו-$\log_2 8 = 3$. (שם)
"בעץ בלי בן יחיד מספר הצמתים אי-זוגי" – **כן.** כל צומת חוץ מהשורש הוא בן, ול-k צמתים יש 2 בנים: $n - 1 = 2k$. (שם)
"תחילי + סופי קובעים עץ אחד" – **לא.** בן יחיד יכול להיות משמאל או מימין: 1 עם בן שמאלי 2, ו-1 עם בן ימני 2 – בשניהם תחילי 2 1 וסופי 1 2. תחילי + תוכי (ערכים שונים) – **כן**.
"הסריקה התוכית של עץ חיפוש ממוינת" – **כן**, בסדר עולה (לא-יורד אם יש ערכים שווים). (2022ב, שאלון 477, מועד 86, שאלה 3)
"מספיק לבדוק כל צומת מול הבנים שלו כדי לדעת שזה עץ חיפוש" – **לא.** בעץ של 4.4 כל זוג אב-בן תקין, ו-1 בתת-העץ הימני של 5. (2025ב, שאלון 473, מועד 78, שאלה 3)

ניסוח
״ערך כפול שמוכנס לעץ חיפוש נהיה בן ימני של העותק״ – **לא תמיד.** רק אם לעותק אין בן ימני. בעץ 3, ובניו 2 (ולו בן שמאלי 1) ו-4 (ולו בן ימני 5), ה-3 החדש נכנס כבן **שמאלי** של 4. (2024א, שאלון 462, מועד 75, שאלה 3)
״לאותם ערכים יש עץ חיפוש אחד״ – **לא.** ל-3 ערכים יש 5 עצים, ול-n ערכים הגובה בין $\lceil \log_2(n+1) \rceil$ רמות (מאוזן) ל-n רמות (שרשרת). (שם)

5 ירושה ופולימורפיזם

5.1 "מה יקרה בכל שורה" – מציירים קודם, ואז שתי שאלות

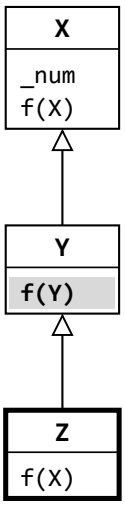
תבנית פתרון – מגדל ← קומפילציה ← ריצה ← פלט

1. **מציירים את המגדל.** קופסה לכל מחלקה, ובה השדות והשיטות עם **סוגי הפרמטרים**. חץ מכל מחלקה אל מחלקת האב. ליד כל שיטה כותבים:
   - **דריסה** – אותו שם ואותם פרמטרים בדיוק כמו שיטה שמעליה. היא מחליפה אותה.
   - **העמסה** – אותו שם עם פרמטרים אחרים. זו שיטה חדשה, והיא לא מחליפה כלום.

   את `Object` מציירים למעלה רק כשהחיפוש מגיע אליו (`toString`, או `equals` שאף מחלקה לא דרסה).
2. **לכל שורה – שני טיפוסים. הטיפוס המוצהר** הוא מה שכתוב לפני שם המשתנה. **הטיפוס בפועל** הוא מה שאחרי `new`, ואותו מסמנים במגדל.
3. **קומפילציה – לפי הטיפוס המוצהר.** מחפשים מהטיפוס המוצהר **ומעלה בלבד** שיטה בשם הזה, שמקבלת את הטיפוס **המוצהר** של כל ארגומנט. אם אין כזו, זו שגיאת קומפילציה. אם יש כמה, נבחרת הספציפית ביותר (הקרובה). **החתימה ננעלת.** casting מותר רק בין מחלקות באותו ענף (למעלה או למטה).
4. **ריצה – לפי הטיפוס בפועל.** casting לטיפוס שהאובייקט אינו – `ClassCastException`. מתחילים מהקומה התחתונה ועולים עד הקומה הראשונה שיש בה **בדיוק החתימה שננעלה**.
5. **הפלט.** מריצים את השיטה שנמצאה. קריאה בתוכה (`g()` היא `this.g()`) עוברת שוב את שלבים 3–4. הטיפוס המוצהר של `this` הוא המחלקה שבה הקוד כתוב.
6. **החריגים לחיפוש:**
   - **שדות** ושיטות **static** לא נדרסים. הם נקבעים לפי הטיפוס המוצהר (ובתוך שיטה, לפי המחלקה שבה הקוד כתוב).
   - **private** לא עוברת בירושה ולא נדרסת.
   - **`super.f()`** מתחילה לחפש קומה אחת מעל המחלקה שבה הקוד כתוב, ולא חוזרת למטה. אבל קריאה **בתוך** הגרסה של האב שוב מתחילה מלמטה.
   - **בנאים:** `super(...)` רץ ראשון, ולכן ההדפסות של הבנאים יוצאות מלמעלה למטה. שיטה דרוסה שנקראת מתוך הבנאי של האב מריצה את הגרסה של הבן (5.4).
   - **protected** נגיש לבן ולכל מחלקה באותה חבילה. בלי מילת הרשאה – רק בחבילה (5.2).

דוגמה – (2026א, שאלון 462, מועד 97, שאלה 5)

```
public class X {                // _num
  public int f(X x) {
    System.out.print("in X ");
    _num = x._num;  return _num;
  }
}
public class Y extends X {
  public int f(Y y) {           // העמסה
    System.out.print("in Y ");
    setNum(y.getNum() + 3);  return getNum();
  }
}
public class Z extends Y {
  public int f(X x) {           // דריסה
    System.out.print("in Z ");
    if (x instanceof Z) return x.getNum() * 10;
    return super.f(x);
  }
}
X xy = new Y(5);   X xz = new Z(3);
Y yy = new Y(4);   Y yz = new Z(2);   Z zz = new Z(6);
```



החיפוש של `zz.f(zz)` במגדל: ל-Z יש **שתי** שיטות f – `f(X)` שלה ו-`f(Y)` שירשה. שתיהן מקבלות Z, והקרובה, `f(Y)`, ננעלת. בזמן ריצה מתחילים ב-Z. אין בה `f(Y)`, ולכן עולים ל-Y.

הפקודה	קומפילציה	ריצה	פלט
`zz.f(zz)`	`f(X)` ו-`f(Y)` מתאימות ← `f(Y)`	`Y.f(Y)`: 6+3	`in Y 9`
`xz.f(yy)`	`f(X)` ← מוצהר X	`X.f(X)` = `super.f(x)` ← Z אינו yy .`Z.f(X)`	`in Z in X 4`
`yy.f(xy)`	`X.f(X)` ← X לא מקבלת `f(Y)`	`X.f(X)`	`in X 5`
`yz.f((Z)xy)`	casting באותו ענף, תקין	ל-xy (שהוא Y) אין קומה Z	שגיאת ריצה: `ClassCastException`
`zz.f(xz)`	`Z.f(X)` ← X מוצהר xz	Z הוא xz .`Z.f(X)`	`in Z 30`

שימו לב: זמן הריצה **לא בוחר חתימה מחדש**. ב-`zz.f(zz)` הוא לא עובר ל-`f(X)` של Z, אף שהיא "נראית" נכונה. התשובה הנפוצה `in Z 60` שגויה.

שימו לב: כש-casting בארגומנט נכשל, הארגומנט מחושב **לפני** הקריאה, ולכן לא מודפס כלום מהשיטה.

שאלות נוספות מאותו סוג: (2022א, שאלון 473, מועד 89, שאלה 4), (2022ב, שאלון 477, מועד 86, שאלה 4), (2022ב, שאלון 477, מועד 96, שאלה 4), (2021ב, שאלון 477, מועד 62, שאלה 4).

5.2 הכללים בטבלה

ניסוח	קוד
דריסה (overriding) – שיטה בבן עם אותו שם ו**אותם פרמטרים בדיוק**. סוג ההחזרה זהה, וההרשאה לא יכולה להיות מחמירה יותר. אי אפשר לדרוס שיטה `final`. (מצגות 7–12, עמ' 8) **העמסה (overloading)** – אותו שם עם פרמטרים אחרים: שיטה חדשה. הבחירה בין העמסות נעשית **בקומפילציה**, לפי הטיפוסים המוצהרים של הארגומנטים.	`class A { void f(int x) {...} }` `class B extends A {` `  void f(int x) {...}  // דריסה` `  void f(double x) {...} // העמסה` `}` `class C extends A {` `  int f(int x) {...}  // שגיאת קומפילציה` `}`
טיפוס מוצהר וטיפוס בפועל – הקומפיילר רואה רק את הטיפוס המוצהר, וזמן הריצה מתחיל מהטיפוס בפועל. השמה `X v = new Y()` עוברת רק אם Y היא X או צאצא שלה. (מצגות 7–12, עמ' 17–18) **casting** – כלפי מעלה תמיד תקין. כלפי מטה עובר קומפילציה ונבדק בריצה: אם לאובייקט אין את הקומה, `ClassCastException`. בין שני ענפים – שגיאת קומפילציה. אל **ממשק** – עובר קומפילציה (אם המחלקה לא `final`) ונבדק בריצה. (מצגות 7–12, עמ' 25–26) **instanceof** – בודק את האובייקט בפועל: `true` אם יש במגדל שלו הקומה. `null instanceof X` הוא `false`. בין שני ענפים – שגיאת קומפילציה. (מצגות 7–12, עמ' 27)	`class Q extends P {...}` `class R extends P {...}` `P p = new Q();` `Q q1 = p;  // שגיאת קומפילציה` `Q q = (Q) p;  // תקין` `R r = (R) p;  // שגיאת ריצה` `R s = (R) q;  // שגיאת קומפילציה` `p instanceof Q  // true` `p instanceof R  // false` `p instanceof Object  // true`
בנאים – השורה הראשונה היא `super(...)` או `this(...)`. אם לא כתוב כלום, Java מוסיפה `super()`, ואז לאב חייב להיות בנאי בלי פרמטרים. מחלקה בלי בנאי מקבלת בנאי ריק. בנאים לא עוברים בירושה. (מצגות 7–12, עמ' 5–6)	`class A { A(int x) {...} }` `class B extends A {` `  B() { }  // שגיאת קומפילציה` `}`
מחלקה אבסטרקטית – אי אפשר ליצור ממנה מופע (new), אבל יש לה שדות, בנאים ושיטות רגילות. מחלקה לא אבסטרקטית שיורשת ממנה **חייבת** לממש כל שיטה אבסטרקטית, באותה חתימה. (מצגות 7–12, עמ' 21–23) **ממשק (interface)** – רק קבועים ושיטות אבסטרקטיות, כולן `public`. מחלקה יורשת ממחלקה אחת וממשת כמה ממשקים. השיטה הממשת חייבת להיות `public`. (מצגות 7–12, עמ' 30)	`abstract class A1 { abstract void f(int y); }` `class B1 extends A1 { }  // שגיאת קומפילציה` `A1 a = new A1();  // שגיאת קומפילציה` `class B4 extends A1 { void f(int y) {...} }` `A1 b = new B4();  // תקין`
`Object` – האב של כל מחלקה. `toString()` שלו מחזירה את שם המחלקה, @ ומספר (B6@1b6d3586). `equals` שלו משווה כתובות, כמו `==`. `println(x)` ו-`x + "s"` קוראים ל-`x.toString()`. (מצגות 7–12, עמ' 38–41)	`public boolean equals(Object other) {  // דריסה` `  return (other != null) && (other instanceof C)` `  && (_x == ((C) other)._x);` `}`

הרשאות גישה (מצגות 7–12, עמ׳ 13–14)

מי ניגש לחבר של המחלקה	`private`	בלי הרשאה (package)	`protected`	`public`
המחלקה עצמה	כן	כן	כן	כן
מחלקה באותה חבילה	לא	כן	כן	כן
מחלקה יורשת בחבילה אחרת	לא	לא	כן (בירושה)	כן
מחלקה אחרת בחבילה אחרת	לא	לא	לא	כן

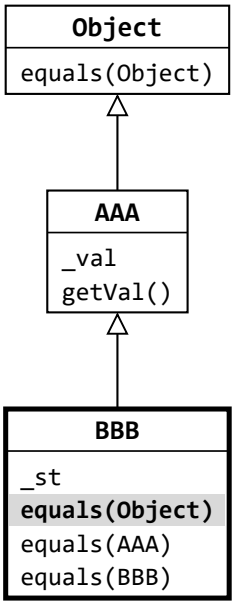
שימו לב: במבחנים כל המחלקות באותה חבילה. לכן גם `Driver` ניגש ישירות לשדה `protected`, ושדה `private` של האב סגור גם בפני הבן (`super.x` הוא שגיאת קומפילציה).

5.3 equals – דריסה מול העמסה

תבנית פתרון – Object למעלה, ואז מי נעול ומי רץ

1. **מציירים את המגדל עם Object למעלה**, ועליו `equals(Object)`. בכל מחלקה כותבים את כל ה-`equals` שלה עם סוג הפרמטר.
   - פרמטר `Object` – **דריסה** של `Object`.
   - כל פרמטר אחר (`equals(A)`, `equals(B)`) – **העמסה**: שיטה חדשה שלא מחליפה כלום.
2. **קומפילציה:** לפי הטיפוס המוצהר של מי שקוראים לו, רושמים את כל ה-`equals` שהוא רואה (ממנו ומעלה, כולל `Object`). מתוכן בוחרים את הספציפית ביותר לטיפוס **המוצהר** של הארגומנט. החתימה ננעלת.
3. **ריצה:** מהטיפוס בפועל ומעלה, עד הגרסה הראשונה של החתימה הנעולה. אם הגענו עד `Object`, התוצאה `true` רק אם זה **אותו אובייקט**.
4. **הפלט:** בתוך השיטה, `instanceof` ו-casting בודקים את הטיפוס בפועל. גם `super.equals(obj)` בוחרת חתימה לפי הטיפוס **המוצהר** של `obj` בתוך השיטה.

דוגמה – (2018א, שאלון 486, מועד 87, שאלה 4)




```
public class BBB extends AAA {      // _st = "bbb"
  public boolean equals(Object other) {
    System.out.print("Object ");
    if (other instanceof BBB) return ...;  // משווה שדות
    return false;
  }
  public boolean equals(AAA other) {
    System.out.print("AAA ");  ...          // אותו גוף
  }
  public boolean equals(BBB other) {
    System.out.print("BBB ");  return ...; // משווה שדות
  }
}
AAA a1 = new AAA(5), a2 = new AAA(5), ab = new BBB();
BBB b1 = new BBB(), b2 = new BBB();
```

המגדל מראה את `ab.equals(a1)`: `ab` מוצהר `AAA`, ומשם רואים רק את `equals(Object)`. בפועל `ab` הוא `BBB`, ולכן רצה הדריסה של `BBB`.

השורה (if (…) println(n))	ננעלת	רצה	פלט
`a1.equals(b1)`	`equals(Object)`	של `Object`: אובייקטים שונים	(כלום)
`b1.equals(a1)`	`equals(AAA)`	של `BBB`. `a1` אינו `BBB`	`AAA`
`ab.equals(a1)`	`equals(Object)`	של `BBB`	`Object`
`b1.equals(ab)`	`equals(AAA)`	של `BBB`. `ab` הוא `BBB` עם אותם ערכים	`AAA 5`
`ab.equals(b1)`	`equals(Object)`	של `BBB`	`Object 6`
`b1.equals(b2)`	`equals(BBB)`	של `BBB`	`BBB 8`

שימו לב: `equals(B other)` היא **העמסה**. משתנה שמוצהר מטיפוס האב לא רואה אותה בכלל, ולכן דרכו רצה `Object.equals`, כלומר ==.

שימו לב: `print` ולא `println`: השם של הגרסה שרצה (`"AAA "`) והמספר שאחריו מודפסים **באותה שורה**.

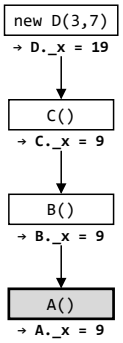
שאלות נוספות מאותו סוג: (2016ב, שאלון 490, מועד 87, שאלה 4), (2017ב, שאלון 467, מועד 84, שאלה 4), (2018א, שאלון 483, מועד 85, שאלה 4), (2023ב, שאלון 468, מועד 99, שאלה 4).

5.4 בנאים וסדר ההדפסות

תבנית פתרון – מפת בנאים, ועולים עד למעלה

1. **בודקים את ההשמה קודם.** `C c3 = new B(6)` כש-B מעל C היא שגיאת קומפילציה, ושום דבר לא מודפס.
2. **מציירים את שרשרת הבנאים.** לכל בנאי, לאן הוא קורא בשורה הראשונה: `super(...)`, `this(...)`, או `super()` מובלע כשלא כתוב כלום.
3. **עולים בשרשרת עד הבנאי העליון.** ההדפסות יוצאות בדרך חזרה: קודם הבנאי של המחלקה העליונה, ואחרון הבנאי שאחרי new.
4. **רושמים את השדות בכל קומה.** שדה שאיש לא נתן לו ערך הוא 0, `null` או `false`. שדה **static** הוא אחד לכל האובייקטים, והוא גדל בכל ריצה של הבנאי, גם דרך `super(...)`.

דוגמה – (2021ב, שאלון 477, מועד 60, שאלה 4)




```
class A {           A() { _x = 9; }
                    A(int x) { _x = x; } }
class B extends A { B() { }
                    B(int x) { super(x); } }
class C extends B { C() { super(); }
                    C(int x) { super(x); _x = _x * x; } }
class D extends C { D() { super(); _x++; }
                    D(int x, int y) { _x = _x + x + y; } }
class E extends C { E() { } }
```

כל בנאי מסתיים בהדפסה `System.out.println("A._x = " + _x)` עם שם המחלקה שלו. `new D(3,7)`: הבנאי לא קורא ל-super בעצמו, ולכן רץ `C()`, ממנו `B()`, ממנו `A()`. `A()` מדפיס ראשון.

הפקודה	מה קורה
`A a1 = new B();`	`A._x = 9 / B._x = 9`
`C c4 = new C(5);`	`A._x = 5 / B._x = 5 / C._x = 25`
`C c5 = new D();`	`A._x = 9 / B._x = 9 / C._x = 9 / D._x = 10`
`A a7 = new D(3,7);`	`A._x = 9 / B._x = 9 / C._x = 9 / D._x = 19`
`C c3 = new B(6);`	שגיאת קומפילציה: B אינה C
`D d8 = new E();`	שגיאת קומפילציה: E ו-D בשני ענפים

שימו לב: שיטה דרוסה בתוך הבנאי של האב רצה בגרסה של הבן, לפני שהבנאי של הבן נתן ערכים לשדות שלו. ב-(2017א, שאלון 455, מועד 86, שאלה 4) הבנאי `A(int x)` מדפיס `toString()`, ו-B דורסת אותה. לכן `new B()` (שקורא ל-`super(4)`) מדפיס `X: 4 Y: 0`: y עוד 0.

שימו לב: מונה static (2017ב, שאלון 467, מועד 93, שאלה 4): `new B(1)` מריץ קודם את `A(1)` כולו (כולל ה-`new A(0)` שבתוכו), ורק אחר כך את הלולאה של B. הפלט: `n = 1 k = 1 / n = 0 k = 2 / n = 0 k = 3` (שורה לכל אחד).

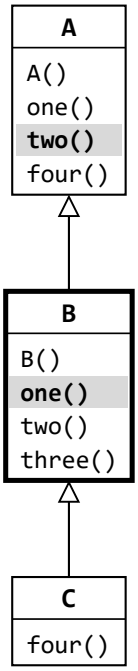
שאלות נוספות מאותו סוג: (2018ב, שאלון 472, מועד 83, שאלה 4), (2020א, שאלון 470, מועד 85, שאלה 4), (2020ב, שאלון 478, מועד 81, שאלה 4), (2026ב, שאלון 468, מועד 89, שאלה 5).

5.5 casting, instanceof ו-super

תבנית פתרון – casting מזיז את החץ

1. **קומפילציה של `v (T)`:** T והטיפוס המוצהר של v באותו ענף? אם לא, זו שגיאת קומפילציה. אחרי ה-casting **החץ זז ל-T**, והשיטה צריכה להיות ב-T או מעליו.
2. **ריצה של `v (T)`:** האם במגדל של האובייקט יש קומה T? אם לא – `ClassCastException`. מה שהודפס לפני כן (למשל מהבנאים) נשאר בפלט.
3. **`instanceof`** בודק את האובייקט בפועל, בדיוק כמו casting בזמן ריצה.
4. **`super.two()`** מריצה את הגרסה של האב. אבל `this.one()` שבתוכה מתחילה שוב **מהקומה התחתונה**.

דוגמה – (2025ב, שאלון 470, מועד 91, שאלה 6)




```
// println = System.out.println
class A {
  A() { println("A"); }
  void one() { println("one of A"); }
  void two() { println("two of A"); this.one(); }
  void four() { println("four of A"); this.two(); }
}
class B extends A {
  B() { println("B"); }
  void one() { println("one of B"); }
  void two() { println("two of B"); }
  void three() { println("three of B"); super.two(); }
}
class C extends B {          // בלי בנאי
  void four() { println("four of C"); }
}
```

ל-C אין בנאי, ולכן הקומפיילר מוסיף `C() { super(); }`. המגדל מראה את `((B) a3).three()`: three של B קוראת ל-`super.two()`, כלומר two של A. בתוכה `this.one()` מתחילה שוב מלמטה, ורצה one של B.

הקוד	התשובה
`A a1 = new A();` `((B) a1).two();`	מודפס A, ואז שגיאת ריצה: `ClassCastException`. a1 הוא A, ואין לו קומה B
`B b1 = new B();` `((A) b1).three();`	שגיאת קומפילציה: אחרי ה-casting החץ על A, ול-A אין three
`A a3 = new B();` `a3.two(); ((B) a3).two();` `((B) a3).three();`	`A / B / two of B / two of B /` `three of B / two of A / one of B`
`B b3 = new C(); b3.four();` `((B) new C()).four();`	`A / B / four of C / A / B / four of C`

שימו לב: `B b3 = (A)(new B());` היא שגיאת קומפילציה: התוצאה של ה-casting היא A, ואותה אי אפשר לשים במשתנה B. וגם `B b2 = (B)(new A(6));` עוברת קומפילציה, אבל הבנאי מדפיס קודם, ורק אחר כך יש `ClassCastException` (2017א, שאלון 455, מועד 86, שאלה 4).

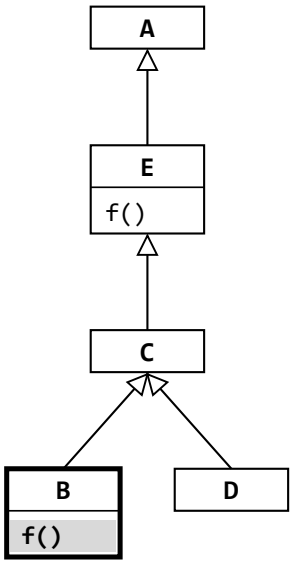
שאלות נוספות מאותו סוג: (2019א, שאלון 478, מועד 84, שאלה 4), (2019א, שאלון 478, מועד 85, שאלה 4), (2019א, שאלון 478, מועד 93, שאלה 4), (2023א, שאלון 468, מועד 57, שאלה 3).

5.6 משחזרים את עץ הירושה מהתוצאות

תבנית פתרון – כל שורה היא רמז

1. **מתרגמים כל שורה לעובדה על העץ:**
   - **השמה** `X v = new Y();` – Y תקינה (לא בהכרח ישירות). שגיאת קומפילציה בהשמה – Y **לא** מתחת ל-X.
   - **casting:** שגיאת קומפילציה – שני הטיפוסים בענפים שונים. שגיאת ריצה – אותו ענף, אבל לאובייקט בפועל אין את הקומה.
   - **קריאה:** שגיאת קומפילציה – לטיפוס המוצהר ולמעליו אין את השיטה. פלט – הגרסה שרצה כתובה בקומה הראשונה, מהטיפוס בפועל ומעלה, שיש בה את השיטה.
2. **מציירים שרשרת מההשמות**, מפצלים לענפים לפי השגיאות, וממקמים את השיטות לפי הפלטים.
3. **בודקים שוב כל שורה על העץ שבנינו**, גם את השורות עם השגיאות.

דוגמה – (2019ב, שאלון 466, מועד 86, שאלה 4)



השורה	מה היא אומרת
`A e1 = new E();`, `E c1 = new C();` תקינות	E מתחת ל-A, ו-C מתחת ל-E
`C b1 = new B();`, `C d1 = new D();` תקינות	B ו-D מתחת ל-C
`B d2 = new D();` – שגיאת קומפילציה	D לא מתחת ל-B
`a1.f();` – שגיאת קומפילציה	ב-A אין f
`((E)e1).f();` מדפיס bye bye	f כתובה ב-E
`((B)b1).f();` מדפיס hello	B דורסת את f
`((D)b1).f();` – שגיאת ריצה	C ו-D באותו ענף, אבל b1 הוא B: אין לו קומה D

שאלות נוספות מאותו סוג: (2019ב, שאלון 466, מועד 83, שאלה 4), (2019ב, שאלון 466, מועד 93, שאלה 4), (2025א, שאלון 463, מועד 72, שאלה 6).

5.7 האם המחלקה עוברת קומפילציה? טענות "נכון תמיד"

תבנית פתרון – רשימת בדיקה למחלקה

1. **שיטות אבסטרקטיות:** מחלקה **לא** אבסטרקטית מממשת כל אחת, באותה חתימה בדיוק. `f(double)` לא מממשת את `f(int)`, כי היא העמסה. `new` של מחלקה אבסטרקטית או של ממשק – שגיאת קומפילציה.
2. **בנאי:** קורא (במפורש או ב-`super()` המובלע) לבנאי שבאמת קיים באב. גם מחלקה בלי בנאי מקבלת `super()` מובלע.
3. **גישה ודריסה:** `private` של האב לא נגיש בבן (גם לא `super.x`). דריסה – אותו סוג החזרה, הרשאה לא מחמירה, לא `final`.
4. **טענה עם "תמיד" או "בהכרח":** מספיקה **דוגמה נגדית אחת** שמתאימה לעץ.

דוגמה – (2017א, שאלון 455, מועד 90, שאלה 4)

```
public abstract class A1 {  private int x;
  public A1(int x) { this.x = x; }   public abstract void f(int y);  }
```

המחלקה	עוברת?	למה
`class B1 extends A1 { B1(int x){super(x);} }`	לא	B1 לא אבסטרקטית ולא מממשת את `f(int)`
`abstract class B2 extends A1 { B2(int x){super(x);} }`	כן	מחלקה אבסטרקטית לא חייבת לממש
`abstract class B3 extends A1 { }`	לא	`super()` מובלע, ול-A1 אין בנאי בלי פרמטרים
`class B4 … { B4(){super(2);} void f(int x){x = 7;} }`	כן	מממשת את `f(int)`
`abstract class B5 … { void f(double x){super.x = 8.5;} }`	לא	`x` הוא `private` ב-A1
`class B6 … { void f(int x){ A1 a1 = new B6(); … } }`	כן	`new B6()` מותר. רק `new A1()` אסור

שימו לב: אם מחלקה יכולה להיות גם ממשק, בודקים גם את המקרה הזה: casting ממחלקה לממשק **עובר קומפילציה** ונבדק בריצה (2026א, שאלון 462, מועד 83, שאלה 5).

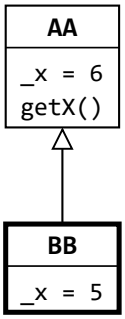
שאלות נוספות מאותו סוג: (2016א, שאלון 481, מועד 83, שאלה 4), (2026א, שאלון 462, מועד 89, שאלה 5), (2026א, שאלון 462, מועד 83, שאלה 5).

5.8 שדות מוסתרים ו-super.x

תבנית פתרון – שני שדות באותו אובייקט

1. **מציירים במגדל את השדה בכל קומה שהגדירה אותו.** לאובייקט של הבן יש **שני** שדות x_.
2. **ממלאים אותם לפי שרשרת הבנאים** (5.4). בתוך שיטה, x_ הוא השדה של המחלקה שבה הקוד כתוב. super.getX() ו-super._x מגיעים לשדה של האב.
3. **v._x מבחוץ:** לוקחים את ה-x_ של **הטיפוס המוצהר** של v, ולא של הטיפוס בפועל.
4. **אם ה-x_ שנמצא הוא private,** זו שגיאת קומפילציה. הקומפיילר לא ממשיך לחפש x_ אחר.

דוגמה – (2016ב, שאלון 490, מועד 91, שאלה 4)




```
public class AA {
  public int _x;
  public AA() { this(5); }
  public AA(int k) { _x = k*2; }
  public int getX() { return _x; }
}
public class BB extends AA {
  public int _x;
  public BB() { this(2); }
  public BB(int k) { super(k+1);  _x = super.getX() - 1; }
}
```

new BB() ← BB(2) ← AA(3): ה-x_ של AA הוא 6, וה-x_ של BB הוא 1–6 = 5.

השורה	שני השדות public	x_ של BB הוא private	x_ של AA הוא private
AA a = new AA(); a._x	10	10	שגיאת קומפילציה
BB b = new BB(); b._x	5	שגיאת קומפילציה	5
a = new BB(); a._x	6 (מוצהר AA)	6	שגיאת קומפילציה
b = (BB)a; b._x	5	שגיאת קומפילציה	5

שאלות נוספות מאותו סוג: (2016א, שאלון 481, מועד 87, שאלה 4), (2026ב, שאלון 468, מועד 91, שאלה 5), (2026ב, שאלון 468, מועד 98, שאלה 5).

5.9 משלימים שיטה שמזהה מאיזו מחלקה האובייקט

תבנית פתרון – טבלת ערכים לכל מחלקה

1. **בונים טבלה:** שורה לכל מחלקה, ועמודה לכל שיטה מותרת. בכל תא כותבים מה השיטה מחזירה **בגרסה שרצה** במחלקה הזו (הגרסה הכי נמוכה במגדל).
2. **מוצאים הפרשים** שמבדילים בין השורות (כאן a - f). אם שתי מחלקות זהות, מוסיפים שיטה שמפרידה ביניהן.
3. **casting:** קוראים לשיטה דרך המחלקה **הגבוהה ביותר** שיש בה את השיטה ((One) obj). ל-obj (Three) עושים casting רק אחרי שידוע שהאובייקט הוא Three או מתחתיה, אחרת תהיה ClassCastException.

דוגמה – (2020ב, שאלון 475, מועד 84, שאלה 4)

```
int a = ((One) obj).getA();
int f = ((One) obj).f();
if (a == f)      System.out.println("One");
if (a + 1 == f)  System.out.println("Two");
if (a - 1 == f)  System.out.println("Four");
if (a == f - 2) {
  int g = ((Three) obj).g();
  if (g == a) System.out.println("Three");
  else        System.out.println("Five");
}
```

המחלקה	f() - getA()	g()
One	0	אין
Two	1	אין
Three	2	a
Four (יורשת מ-Three)	-1	a
Five (יורשת מ-Three)	2	a + 1

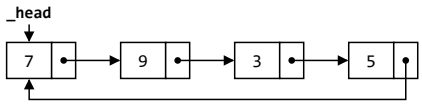
שאלות נוספות מאותו סוג: (2020ב, שאלון 475, מועד 96, שאלה 4).

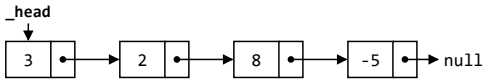
6 רשימות מקושרות

6.1 הצומת והרשימה – כמו שהמבחנים נותנים אותם

ניסוח	קוד
צומת – IntNode. ערך _value ומצביע _next לצומת הבא. null אומר שאין צומת הבא. (מצגות 7–12, עמ' 108)	`IntNode(int val, IntNode n)` `int getValue()` `IntNode getNext()` `void setValue(int v)` `void setNext(IntNode node)`
רשימה – IntList. שדה אחד: private IntNode _head. רשימה ריקה: _head == null. כל שיטה מתחילה ב-_head ומתקדמת עם getNext(). (עמ' 108–109)	`public class IntList {` `  private IntNode _head;` `  public IntList() { _head = null; }` `  public IntList(IntNode node) { _head = node; }` `}`
רשימה דו-סטרית – IntNodeTwo. _num, _next ו-_prev (getNum, getPrev, setPrev...). ב-IntListTwo יש _head ו-_tail. כל שינוי מעדכן **שני** חצים, וגם את _tail כשמוסיפים בסוף. (עמ' 112–114)	`// הוספת temp אחרי curr` `temp.setNext(curr.getNext());` `temp.setPrev(curr);` `curr.setNext(temp);` `if (temp.getNext() == null)` `  _tail = temp;` `else` `  temp.getNext().setPrev(temp);`

- **הכתיב בשאלה.** {3, 2, 8, -5} היא הרשימה 3 -> 2 -> 8 -> -5 -> null. ברוב המבחנים **המספר השמאלי הוא הראש**, אבל בודקים מה כתוב בשאלה.
- **return new IntList(_head)** בונה אובייקט IntList חדש על **אותם צמתים**. זו לא העתקה: שינוי בתוצאה הוא שינוי ברשימה המקורית.
- **רשימה מעגלית** (בציור משמאל): האחרון מצביע על הראשון, ואין null. לולאה עוצרת כשחוזרים לצומת שממנו התחלנו. (עמ' 116–118)

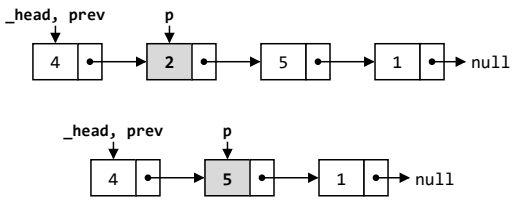




6.2 מעקב: מצביעים כמדבקות

תבנית – מעקב על קוד של רשימה

1. **מציירים את הרשימה:** קופסה לכל צומת, חץ לכל next, ו-null בסוף.
2. **מדבקה לכל משתנה מצביע** (p, prev, curr, _head...), מעל הצומת שהוא מצביע עליו.
3. **שורה אחר שורה:**
   - השמה למשתנה (p = p.getNext()) **מזיזה מדבקה**.
   - setNext **משנה חץ**. מוחקים את החץ הישן מיד.
   - setValue משנה רק את המספר שבקופסה. החצים לא זזים.
   - צומת שאף חץ ואף מדבקה לא מגיעים אליו **יצא מהרשימה**.
4. **בסוף כל סיבוב בלולאה** כותבים שורה בטבלת מעקב: כל המצביעים והמשתנים.
5. **בסוף קוראים את התוצאה מהמצביע שמוחזר (או מ-_head) לפי החצים**, ולא לפי סדר הקופסאות בדף.



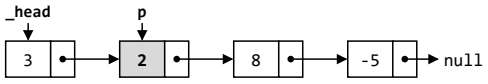

```
prev.setNext(p.getNext()); // חץ: prev עוקף את 2
p = p.getNext();           // מדבקה: p זז ל-5
```

שימו לב: אחרי היפוך הקופסאות בדף עדיין בסדר הישן. רק החצים קובעים מה הסדר.

6.3 התבניות הבסיסיות – שלד ושרטוט

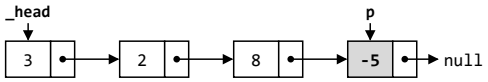
כמעט כל שיטה במבחן בנויה מהתבניות האלה. מזהים אותן בקוד, ונותנים להן שם בתשובה.

מעבר על כל הרשימה. p מתחיל בראש ומתקדם עד null: אורך, סכום, חיפוש, מקסימום. $O(n)$. (עמ' 111)



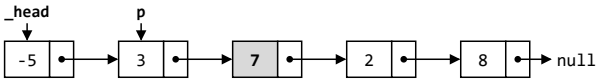

```
IntNode p = _head;
while (p != null) {
    ... p.getValue() ...
    p = p.getNext();
}
```

עצירה בצומת האחרון. התנאי הוא על p.getNext(), ולכן צריך לוודא _head != null לפני הלולאה.



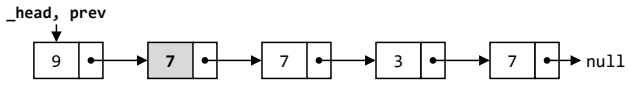

```
IntNode p = _head;     // _head != null
while (p.getNext() != null)
    p = p.getNext();
```

הוספה בראש / אחרי p. קודם מחברים את הצומת החדש להמשך, ורק אחר כך מחברים אליו את p (או את _head). בראש: $O(1)$. בסוף: $O(n)$, כי צריך להגיע לאחרון. בציור: 7 נוסף אחרי 3. (עמ' 109)



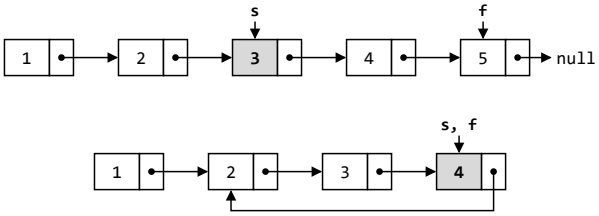

```
_head = new IntNode(x, _head);          // בראש
p.setNext(new IntNode(x, p.getNext())); // אחרי p
```

מחיקה עם prev. אין דרך לחזור אחורה, ולכן עוצרים בצומת **שלפני** זה שמוחקים ומדלגים עליו. הראש הוא מקרה נפרד (_head = _head.getNext()). כשמוחקים את כל המופעים, אחרי מחיקה prev **נשאר במקומו**, כי הצומת החדש שאחריו עוד לא נבדק: {7, 7, 9, 7, 7, 3, 7} בלי 7 ← {9, 3}. למחיקה של מופע אחד בלבד מוסיפים return אחרי המחיקה. (עמ' 110–111; 2016ב, שאלון 490, מועד 87, שאלה 5)




```
while (_head != null && _head.getValue() == num)
    _head = _head.getNext();          // הראש
if (_head == null) return;
IntNode prev = _head;
while (prev.getNext() != null) {
    if (prev.getNext().getValue() == num)
        prev.setNext(prev.getNext().getNext());
    else
        prev = prev.getNext();
}
```

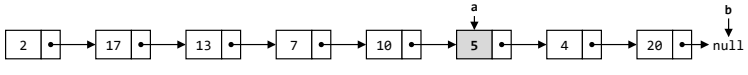
איטי ומהיר: האמצע, ומעגל. f זז שני צעדים ו-s צעד אחד. כש-f מגיע לסוף, s באמצע. באורך זוגי s על **השני** משני האמצעיים ({1, 2, 3, 4} ← 3). **מעגל:** אותה לולאה, ובסוף כל סיבוב בודקים s == f. אם יש מעגל, המהיר מקיף את האיטי ופוגש אותו; אם אין, f מגיע ל-null. מעבר אחד, $O(n)$. (2017ב, שאלון 467, מועד 93, שאלה 5; 2024א, שאלון 462, מועד 63, שאלה 4)




```
IntNode s = _head, f = _head;
while (f != null && f.getNext() != null) {
    f = f.getNext().getNext();
    s = s.getNext();
    if (s == f) return true;   // רק לבדיקת מעגל
}
```

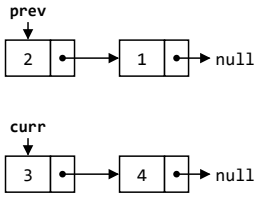
פער של k: הצומת ה-k מהסוף. b יוצא k צעדים לפני a, ואז שניהם זזים יחד. כש-b מגיע ל-null, a על הצומת ה-k מהסוף. בציור k = 3.

(2019א, שאלון 478, מועד 85, שאלה 5)



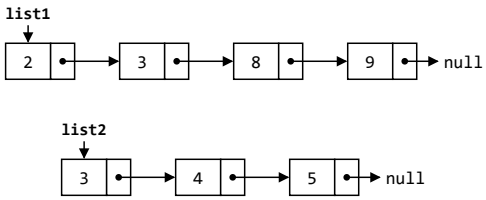

```
IntNode a = _head, b = _head;
for (int i = 0; i < k; i++) {
    if (b == null) return null; // פחות מ-k צמתים
    b = b.getNext();
}
while (b != null) {
    a = a.getNext();
    b = b.getNext();
}
```

היפוך. שלושה מצביעים. את `next` שומרים **לפני** שמשנים את החץ, כי זו הדרך היחידה להגיע להמשך. בציור: אחרי שני סיבובים על {1, 2, 3, 4}. $O(n)$ זמן, $O(1)$ מקום. (2026א, שאלון 462, מועד 97, שאלה 4)



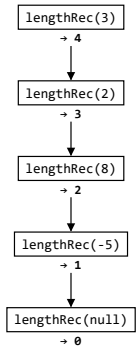

```
IntNode prev = null, curr = _head, next;
while (curr != null) {
    next = curr.getNext();
    curr.setNext(prev);
    prev = curr;
    curr = next;
}
_head = prev;
```

מיזוג שתי רשימות ממוינות. בכל צעד לוקחים את הקטן משני הראשים: {2, 3, 8, 9} ו-{3, 4, 5} ← {2, 3, 3, 4, 5, 8, 9}, ב-$O(n+m)$. אותו רעיון בכל שאלה על שתי רשימות ממוינות (חיתוך, קישור ערכים שווים, ההפרש הקטן): מקדמים את המצביע שעל הערך הקטן. (עמ' 126–129)



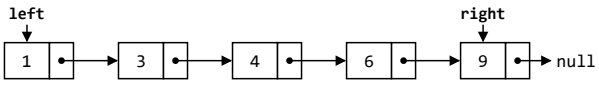

```
IntNode merge(IntNode list1, IntNode list2) {
    if (list1 == null) return list2;
    if (list2 == null) return list1;
    if (list1.getValue() < list2.getValue()) {
        list1.setNext(merge(list1.getNext(), list2));
        return list1;
    }
    list2.setNext(merge(list1, list2.getNext()));
    return list2;
}
```

רקורסיה על רשימה. תנאי עצירה: `p == null`. שיטה פרטית מקבלת צומת ו**מחזירה את הראש החדש**, והציבורית עושה `_head = f(_head, …)`. כך מכניסים למקום הנכון ברשימה ממוינת: {1, 3, 5} ו-f(4) ← {1, 3, 4, 5}. בציור: האורך ברקורסיה על {3, 2, 8, -5}. (עמ' 119–125; 2026ב, שאלון 468, מועד 98, שאלה 4)




```
public void f(int num) { _head = f(_head, num); }
private IntNode f(IntNode p, int num) {
    if (p == null) return new IntNode(num);
    if (p.getValue() > num) return new IntNode(num, p);
    p.setNext(f(p.getNext(), num));
    return p;
}
```

שני קצוות ברשימה דו-סטרית. `left` מתחיל בראש ו-`right` בזנב. בכל סיבוב זז **רק אחד** מהם, ולכן הם נפגשים ולא עוברים זה את זה. ברשימה ממוינת: סכום קטן מדי ← `left` קדימה, גדול מדי ← `right` אחורה. $O(n)$. בציור החצים של `prev` לא מצוירים. (עמ' 115; 2022ב, שאלון 477, מועד 96, שאלה 5)




```
IntNodeTwo left = _head, right = _tail;
while (left != right) {
    int s = left.getNum() + right.getNum();
    if (s == x) return true;
    if (s < x) left = left.getNext();
    else right = right.getPrev();
}
return false;
```

חלון: שני מצביעים באותו כיוון. הקדמי מרחיב את הקטע ומוסיף לסכום, האחורי מצמצם ומחסיר. כל אחד עובר על הרשימה פעם אחת, ולכן $O(n)$ ולא $O(n^2)$. (2018א, שאלון 483, מועד 91; 2021ב, שאלון 477, מועד 62; 2022ב, שאלון 477, מועד 86)

6.4 "מה עושה השיטה?" – מעקב, הכללה, מקרי קצה

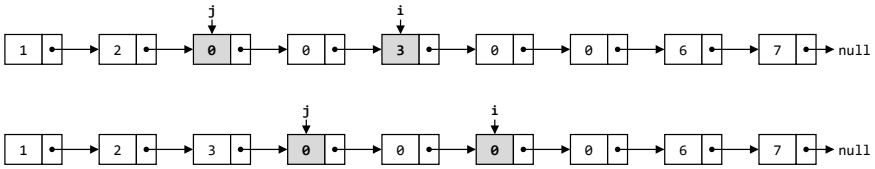
זה הסוג של כמעט כל שאלת רשימות (כ-58 מתוך 61 השאלות): מעקב על רשימה נתונה, ואז "מה השיטה עושה באופן כללי? התייחסו למקרי קצה".

תבנית פתרון – מעקב ← הכללה ← מקרי קצה ← סיבוכיות

1. **קוראים את הקוד:** מה כל מצביע מסמן ("j = המקום הבא לאיבר שאינו אפס"), ומה כל לולאה עושה. האם השיטה משנה חצים (setNext) או רק ערכים (setValue)?
2. **מעקב על הרשימה מהשאלה** בשיטת המדבקות (6.2): שרטוט בכל שינוי, ושורה בטבלה לכל סיבוב.
3. **מכלילים:** משווים את הקלט לפלט. מה נשאר, מה זז, מה נמחק, ולמה? בודקים את ההשערה על עוד דוגמה קטנה.
4. **כותבים מה, לא איך:** משפט אחד על **התוצאה** (מה מוחזר, מה מודפס, איך נראית הרשימה), לא על הלולאות.
5. **מקרי קצה:** רשימה ריקה, צומת אחד, ערכים שווים (> מול =>), אין אף איבר מתאים, ערכים שליליים ואפס.
6. **תופעות לוואי:** האם הרשימה המקורית השתנתה? (setValue או setNext על הצמתים שלה, _head = null, return new IntList(_head).)
7. **סיבוכיות** אם נשאלת: זמן ומקום, ולמה (כמה פעמים כל מצביע עובר על הרשימה).

דוגמה – (2025ב, שאלון 473, מועד 78, שאלה 4): something על {0, 1, 2, 0, 3, 0, 0, 6, 7}

```
IntNode i = _head, j = _head;
while (i != null) {
    if (i.getValue() != 0) {
        int temp = i.getValue();
        i.setValue(j.getValue());
        j.setValue(temp);
        j = j.getNext();
    }
    i = i.getNext();
}
return new IntList(_head);
```



שלב 1 – הקוד: i עובר על כל הצמתים. j מסמן את המקום הבא לאיבר שאינו אפס. כש-i על ערך שאינו אפס, מחליפים את הערכים של i ו-j, ו-j מתקדם. רק setValue: החצים לא משתנים.

שלב 2 – המעקב (מקומות מ-0; בציור: לפני ההחלפה של 3 ואחריה):

i במקום (ערך)	מה קורה	j אחרי	הרשימה אחרי
0 (0)	אפס: רק i זז	0	{0, 1, 2, 0, 3, 0, 0, 6, 7}
1 (1), 2 (2)	החלפה, j זז	2	{1, 2, 0, 0, 3, 0, 0, 6, 7}
3 (0)	אפס	2	ללא שינוי
4 (3)	החלפה עם מקום 2	3	{1, 2, 3, 0, 0, 0, 0, 6, 7}
5 (0), 6 (0)	אפס	3	ללא שינוי
7 (6), 8 (7)	החלפה, j זז	5	{1, 2, 3, 6, 7, 0, 0, 0, 0}

שלב 3–4 – מה: something מזיזה את כל האפסים לסוף הרשימה, והאיברים שאינם אפס נשארים באותו סדר יחסי.

שלב 5 – מקרי קצה: רשימה ריקה נשארת ריקה. רשימה בלי אפסים, רק עם אפסים או עם איבר אחד לא משתנה. שליליים לא מפריעים ({-2, 4, 0} ← {-2, 0, 4}).

שלב 6 – תופעת לוואי: הרשימה המקורית משתנה. מוחזר IntList חדש, אבל על אותם צמתים.

שלב 7 – סיבוכיות: $O(n)$ זמן, כי i ו-j רק מתקדמים, ו-$O(1)$ מקום. secret מאותה שאלה נותנת אותה תוצאה ב-$O(n^2)$: לכל אפס היא סורקת קדימה עד הערך הבא שאינו אפס, וברשימה של n אפסים זה $\frac{n(n-1)}{2}$ צעדים.

שימו לב: "האם יש הבדל בין השיטות?" לא שואלת רק על הפלט. כשהתוצאה זהה עונים גם על היעילות.

שימו לב: שיטה שרק מחליפה ערכים לא משנה את האורך. סופרים איברים לפני ואחרי.

שאלות נוספות מאותו סוג: (2016א, שאלון 481, מועד 87, שאלה 5), (2019ב, שאלון 466, מועד 86, שאלה 5), (2024ב, שאלון 468, מועד 87, שאלה 4), (2026א, שאלון 462, מועד 97, שאלה 4).

6.5 "האם קיימת רשימה ש...?" / "תנו קלט ש..."

כ-20 שאלות: האם יש רשימה ששתי שיטות נותנות עליה אותה תוצאה, מתי הרשימה לא משתנה, איזה טווח של x מחזיר ערך מסוים.

תבנית פתרון – מהצורה של התוצאה אל התנאי

1. **קודם יודעים מה כל שיטה עושה** (6.4), במילים של התוצאה.
2. **מתרגמים את הדרישה לתנאי על הרשימה.** למשל "`something` לא מוחקת כלום" ⇔ "אחרי אף איבר אין איבר גדול ממנו".
3. **הצורה של התוצאה מכריעה:** תמיד ממוינת? תמיד לא עולה? אותו אורך?
4. **אם קיימת:** דוגמה שעומדת בכל התנאים של השאלה (אורך, איברים שונים), והרצה של שני הצדדים עליה.
5. **אם לא קיימת:** נימוק מהצורה (סתירה), ולא "ניסיתי ולא מצאתי".
6. **טווח של `x`:** מה מוחזר כפונקציה של `x`, ומשם התחום. שימו לב לקצוות (`<` מול `=<`).

דוגמה – (2026א, שאלון 462, מועד 97, שאלה 4, סעיף ה)

`mystery` הופכת את הרשימה. `something` משאירה, באותו סדר, רק את האיברים שאין אחריהם איבר גדול מהם (`{9, 10, -12, 18, 13, -11, 2}` ← `{18, 13, 2}`). האם יש רשימה של 5 איברים **שונים** ש-`mystery` נותנת עליה את ההפוך של מה ש-`something` נותנת?

- `mystery(L)` היא ההפוכה של `L`, ולכן צריך ש-`something(L) = L`, כלומר ש-`something` לא תמחק כלום.
- היא לא מוחקת כלום ⇔ כל איבר גדול או שווה לכל מה שאחריו. עם איברים שונים: **הרשימה יורדת ממש**.
- **כן:** `{5, 4, 3, 2, 1}` – שני הצדדים `{1, 2, 3, 4, 5}`. (ב-`{5, 1, 4, 2, 3}` לא: `{3, 2, 4, 1, 5}` מול `{3, 4, 5}`.)

שימו לב: בודקים את התנאים של השאלה על הדוגמה: אורך, "שונים זה מזה", "לא כולם זהים".

שאלות נוספות מאותו סוג: (2023א, שאלון 471, מועד 65, שאלה 4), (2024ב, שאלון 465, מועד 61, שאלה 4), (2025א, שאלון 463, מועד 95, שאלה 4), (2026א, שאלון 462, מועד 89, שאלה 4).

6.6 "כתבו את השיטה היעילה ביותר"

כ-10 שאלות שואלות על סיבוכיות, ובחלקן צריך לכתוב (או להשלים) גרסה יעילה.

תבנית פתרון – מה לא נוצל ← מעבר אחד ← סיבוכיות

1. **מה הפתרון הישיר עושה, ומה הסיבוכיות שלו?** בדרך כלל לולאה בתוך לולאה: $O(n^2)$.
2. **איזה נתון לא נוצל?** רשימה ממוינת: שני מצביעים כמו במיזוג. רשימה דו-סטרית: שני קצוות. "רק האחרון חשוב": משתני עזר במעבר אחד.
3. **שלד של מעבר אחד:** בכל סיבוב לפחות מצביע אחד מתקדם, ואף אחד לא חוזר אחורה.
4. **מעקב על הדוגמה מהשאלה:** יוצאת אותה תוצאה כמו בשיטה המקורית?
5. **מקרי קצה:** רשימה ריקה, רשימה אחת נגמרת לפני השנייה, ערכים שווים.
6. **סיבוכיות ולמה:** "כל מצביע מתקדם לכל היותר n פעמים, ולכן עד $2n$ סיבובים: $O(n)$ זמן, $O(1)$ מקום."

דוגמה – (2021א, מועד 70, שאלה 5)

שתי רשימות ממוינות. `setSp` מקשרת (במצביע `_sp`) כל איבר ב-`this` לאיבר **הראשון הפנוי** עם אותו ערך ב-`list`, ומחפשת כל פעם מראש `list`: $O(n^2)$. הגרסה היעילה, כמו מיזוג:

```
IntNodeSp temp1 = _head, temp2 = list._head;
while ((temp1 != null) && (temp2 != null)) {
    if (temp1.getNum() == temp2.getNum()) {
        temp2.setSp(temp1);       temp1.setSp(temp2);
        temp1 = temp1.getNext();  temp2 = temp2.getNext(); // שניהם
    }
    else if (temp1.getNum() < temp2.getNum()) temp1 = temp1.getNext();
    else temp2 = temp2.getNext();
}
```

על `{26, 26, 47, 52, 69, 95, 95}` ו-`{15, 26, 26, 69, 69, 95}` יוצאים אותם ארבעה זוגות כמו במקורית, ול-47, ל-52 ול-95 השני נשאר `null`. $O(n)$ זמן, $O(1)$ מקום.

שימו לב: בשוויון מקדמים **את שניהם**. אם מקדמים רק את `temp1`, ה-26 השני נקשר שוב ל-26 **הראשון** של `list`.

שאלות נוספות מאותו סוג: (2020א, שאלון קורונה, שאלה 1), (2021א, מועד 85, שאלה 5), (2022א, שאלון 476, מועד 74, שאלה 5), (2016א, שאלון 481, מועד 87, שאלה 5).

6.7 מלכודות

ניסוח	קוד
בודקים `null` קודם. ב-`&&` הצד השמאלי נבדק ראשון. `p != null && p.getValue() != num` נופל ב-`NullPointerException` כשהערך לא ברשימה.	`while (p != null && p.getValue() != num)` `  p = p.getNext();`
לא מזיזים את `_head` כדי לעבור על הרשימה. לולאה עם `_head = _head.getNext()` מחשבת נכון, אבל משאירה רשימה ריקה. עוברים תמיד עם מצביע עזר.	`for (IntNode p = _head; p != null; p = p.getNext())` `  s += p.getValue();`
סדר בהוסמה. כאן `p.getNext()` הוא כבר `t`, ולכן `t` מצביע על עצמו וההמשך של הרשימה אבד.	`p.setNext(t);` `t.setNext(p.getNext());  // לא נכון`

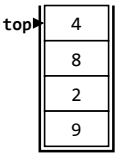
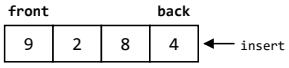
שימו לב: `p.getNext().getNext()` רק אחרי שבדקנו ש-`p.getNext() != null`. על הצומת האחרון זה `NullPointerException`.

שימו לב: אחרי היפוך, משתנה שהצביע על הראש הישן מצביע עכשיו על **האחרון**: אחרי ההיפוך של {1, 2, 3, 4} הוא רואה רק {1}. קוראים את התוצאה מהראש החדש.

שימו לב: "בלי לשנות את הרשימה": `setValue`, `setNext` ו-`_head = null` משנים את הרשימה המקורית, וגם `new IntList(_head)` הוא אותם צמתים. אם אסור לשנות, מחזירים את הרשימה למצבה בסוף, כמו בשאלה שהופכת חצי, משווה, והופכת בחזרה. (2022ב, שאלון 477, מועד 91, שאלה 5)

7 מחסניות ותורים

7.1 הממשק (API)

ניסוח	שרטוט	קוד
מחסנית – Stack, LIFO: האחרון שנכנס יוצא ראשון. `push(x)` דוחף לראש המחסנית. `pop()` שולף: מוציא **ומחזיר**. `top()` מחזיר את הראש ולא מוציא. בקוד: שתיהן מחזירות 4, אבל רק pop מוציא אותו. `isEmpty()`. (מצגות 7–12, עמ' 132)		`s.push(9); s.push(2);` `s.push(8); s.push(4);` `s.top();  // 4` `s.pop();  // 4`
תור – Queue, FIFO: הראשון שנכנס יוצא ראשון. `enqueue(x)` מכניס לזנב התור. `dequeue()` מוציא מראש התור ומחזיר. `first()` מחזיר את הראש ולא מוציא. `isEmpty()`. (עמ' 138)		`q.first();  // 9` `q.dequeue();  // 9` `q.enqueue(5);  // 2 8 4 5`
הצצה בעזרת הפעולות הבסיסיות. כשהממשק בשאלה חלקי, בונים מה שחסר מ-pop ו-push. (עמ' 134)		`int temp = s.pop();` `s.push(temp);` `return temp;  // כמו top`

- **מבנה ריק:** pop, top, dequeue ו-first על מבנה ריק הן שגיאה. במימוש של הקורס: NullPointerException. לכן בודקים isEmpty() לפני. (עמ' 133, 140)
- **המימוש ברשימה מקושרת:** מחסנית היא רשימה עם top_, ו-push מוסיף בראש. תור הוא רשימה עם head_ ו-tail_: נכנסים בזנב ויוצאים מהראש. כל פעולה $O(1)$. (עמ' 133, 139)
- **מבנים מהשאלה:** StrStack (מחרוזות), "תור ישראלי" (insertAtHead), תור-כפול, תלת-מחסנית. משתמשים **רק** בשיטות שבממשק שהשאלה נותנת.

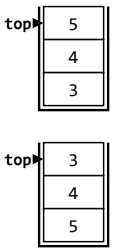
7.2 קריאה ושחזור – מבנה עזר

שאלות "כתבו שיטה" על מחסנית או תור כמעט תמיד דורשות שבסוף המבנה יחזור למצבו.

תבנית – לעבור על מחסנית בלי לקלקל אותה

1. **מוציאים הכול** (pop) ובדרך עושים את העבודה: סכום, ספירה, חיפוש.
2. **כל איבר שיצא נדחף למחסנית עזר.** היא מחזיקה את האיברים בסדר הפוך.
3. **מחזירים מהעזר למקורית:** הפוך של הפוך הוא **אותו סדר**.
4. **בתור:** אותו דבר עם **תור עזר**, כי תור שומר על הסדר.

```
public static int sum(Stack s) {
    Stack temp = new Stack();
    int sum = 0;
    while (!s.isEmpty()) {
        int x = s.pop();
        sum += x;
        temp.push(x);
    }
    while (!temp.isEmpty())
        s.push(temp.pop());   // מחזירים
    return sum;
}
```



מעבירים	הסדר
מחסנית ← מחסנית	מתהפך
מחסנית ← מחסנית ← מחסנית	נשמר
תור ← מחסנית ← תור	מתהפך
מחסנית ← תור ← מחסנית	מתהפך
תור ← תור	נשמר

בציור: s לפני (למעלה), ו-temp אחרי הלולאה הראשונה, כש-s ריקה. בסוף s חוזרת למצבה.

דוגמה – (2017א, שאלון 455, מועד 87, שאלה 6, סעיף א): fun1 מעבירה את התור למחסנית ובחזרה, ולכן **הופכת את סדר האיברים בתור**: {1, 2, 3, 4} ← {4, 3, 2, 1}. תור ריק או תור עם איבר אחד לא משתנה.

שימו לב: בלי הלולאה השנייה התשובה נכונה, אבל המחסנית נשארת ריקה. וגם החזרה דרך **תור** עזר במקום מחסנית הופכת את המחסנית.

7.3 "מה עושה השיטה?" על מחסנית ותור

5 מתוך 6 השאלות: מעקב, ואז מה השיטה עושה באופן כללי.

תבנית פתרון – מציירים כל מבנה ועוקבים אחרי כל פעולה

1. **מציירים כל מבנה** עם שם המשתנה: תור לרוחב, הראש משמאל. מחסנית לגובה, הראש למעלה.
2. **לפני כל פעולה שואלים: מאיזה צד?** במחסנית תמיד מלמעלה. בתור יוצאים מהראש ונכנסים בזנב.
3. **`top` ו-`first` רק מציצים.** שום דבר לא יזז.
4. **שורה בטבלה לכל סיבוב:** מה יזז, ומה יש בכל מבנה.
5. **בסוף מסתכלים מה נשאר בכל מבנה:** התור התרוקן? התהפך? מחסנית מקומית נזרקת? זה חלק מהתשובה.
6. **מה** + **מקרי קצה:** מבנה ריק, איבר אחד, כבר ממוין, כולם שווים.

דוגמה – (2017א, שאלון 455, מועד 87, שאלה 6, סעיף ב)

```
while (!q.isEmpty()) {
    if (s.isEmpty() || s.top() <= q.first())
        s.push(q.dequeue());
    else {
        q.enqueue(s.pop());
        count++;
    }
}
return count;
```

מעקב על התור {3, 5, 2, 4}: בסוף המחסנית 2, 3, 4, 5 (5 למעלה), התור ריק, ומוחזר 5.

מה: `fun2` מעבירה את כל האיברים מהתור למחסנית, ממוינים מהקטן (למטה) לגדול (למעלה), ומחזירה כמה פעמים איבר הוחזר מהמחסנית לתור. **התור מתרוקן.** תור ריק או ממוין בסדר עולה: 0. הכי הרבה: $\frac{k(k-1)}{2}$, בתור בסדר יורד ({4, 3, 2, 1} ← 6).

שימו לב: המחסנית הממוינת היא משתנה מקומי ונעלמת בסוף השיטה. מה שנשאר לקורא: תור ריק וערך החזרה.

שאלות נוספות מאותו סוג: (2020ב, שאלון 475, מועד 84, שאלה 5), (2020ב, שאלון 475, מועד 96, שאלה 5), (2022א, שאלון 473, מועד 67, שאלה 5), (2022א, שאלון 473, מועד 89, שאלה 5).

7.4 מלכודות

שימו לב: המבנה המקורי חוזר למצבו, אלא אם השאלה אומרת אחרת. לולאת `while (!s.isEmpty())` שרק מוציאה משאירה מבנה ריק.

שימו לב: `top` ו-`first` לא מוציאים. לולאה שרק מציצה לא מתקדמת: כדי להתקדם צריך `pop` או `dequeue`.

שימו לב: לולאה עם מספר קבוע של פעולות על מבנה שהגודל שלו משתנה: בודקים שאין `dequeue` על תור ריק, למשל בחלוקה ל-`length/2` באורך אי-זוגי. (2022א, שאלון 473, מועד 89, שאלה 5)

שימו לב: ממשק מוגבל: כשהשאלה אוסרת מבנים נוספים או שיטות מסוימות (תלת-מחסנית בלי `top`, רק מעברים מותרים), כל שורה בפתרון משתמשת רק במה שמותר. (2020א, שאלון קורונה, שאלה 5)