פרק 3

מיון - חיפוש - מיזוג

מה הוא מיון?

מיון נתונים הוא סידורם בסדר עולה או יורד. מיון יכול להיות מספרי או אלפביתי. ברשימת נתונים ממוינת ניתן לייעל את זמן איתור הנתונים.

מה הוא חיפוש?

חיפוש הוא תהליך איתור של נתון מסוים בסדרת נתונים.

מה הוא מיזוג?

מיזוג הוא תהליך בו משלבים (ממזגים) שתי סדרות של נתונים ממוינים לסדרת נתונים ממוינת אחת הכוללת את נתוני שתי הסדרות.

מיון וחיפוש - לשם מה?

בחיי היום יום אנו נתקלים לעתים קרובות בצורך במיון וחיפוש:

- כדי למצוא בקלות תלמיד ברשימת התלמידים בכיתה או כדי למצוא בקלות נמען בספר טלפונים מממיינים את הנתונים על-פי השדה 'שם משפחה', ועבור שמות משפחה זהים ממיינים מיון נוסף לפי שם פרטי.
- כדי לחפש מילה במילון ממיינים את המילים בסדר לקסיקוגרפי.
- במשרד הפנים לצורך איתור (חיפוש) אדם במאגר משתמשים במספר תעודת הזהות שלו, המאפשר חיפוש מהיר במאגר הממוין לפי ת.ז.
- כדי לבחון נתונים סטטיסטיים של רשימת נתונים (למשל ציונים) ממיינים את הנתונים ואז ניתן בקלות למצוא למשל את החציון, את השכיח, את הגבוהים ביותר או את הנמוכים ביותר.

בפרק זה נלמד אלגוריתמים שונים של שיטות מיון, חיפוש ומיזוג:

- ✓ חלק א: שיטות מיון - עמוד 69;
- ✓ חלק ב: שיטות חיפוש - עמוד 78;
- ✓ חלק ג: מיזוג - עמוד 82.

.....

.....

.....

חלק א: שיטות מיון

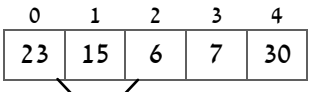
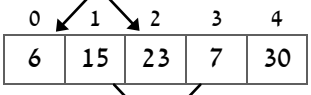
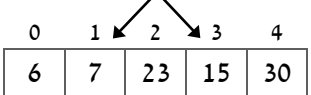
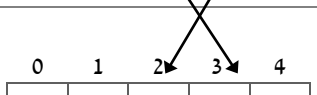
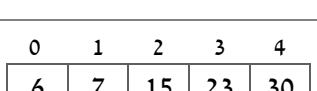
בחלק זה יתוארו מספר שיטות למיון. בעבור מערך חד-ממדי של מספרים נציג אלגוריתמים לשינוי סדר האיברים במערך כך שיתקבל מערך ממוין. האלגוריתמים שיוצגו יציינו את המערך **בסדר עולה** כלומר מן הערך הקטן ביותר שיושם בתחילת המערך לערך הגדול ביותר שיושם בסוף המערך. כל ערך במערך גדול או שווה מן הערך הקודם לו במערך.

שיטות המיון שנציג בפרק זה הן: מיון בחירה – Selection sort; מיון בועות – Bubble sort (או בשמו הנוסף: החלפת שכנים) ומיון הכנסה – Insertion sort.

מיון בחירה - *Selection sort*

מיון בחירה בסדר עולה של מערך חד-ממדי מתבסס על בחירה (מציאה) בכל שלב של הערך המינימלי מבין הערכים שעוד לא מוינו והעברתו למקומו. בשלב הראשון יימצא המינימלי מבין הערכים בכל המערך ויושם בתא הראשון. בשלב השני יימצא המינימלי מבין הערכים החל מן המקום השני (כי הראשון כבר מסודר) ויושם בתא השני וכך הלאה. בכל שלב ימצא מינימלי ויועבר למקום שלו כך שבסופו של דבר יועבר כל איבר אל מקומו על פי סדר המיון ויתקבל מערך ממוין. ההעברה היא למעשה החלפה בין זוג איברים, כך שהאיבר שנמצא בכל שלב במקום אליו מושם המינימלי יתחלף איתו במקומות כדי שלא ימחקו איברים. מיון בחירה נקרא גם בשם מיון ישיר.

דוגמה עבור מערך a בגודל 5:

הסבר	המערך שמתקבל בכל סריקה
המערך המקורי	
בסריקה הראשונה נמצא את מיקומו של האיבר המינימלי מהמקום ה-0 ועד למקום ה-N-1: מיקומו של האיבר המינימלי הוא 2, מכיוון ש-6 הוא האיבר הקטן ביותר. נחליף בין האיבר במקום ה-0, לאיבר במקום ה-2.	
בסריקה השנייה נמצא את מיקומו של האיבר המינימלי מהמקום שמספרו הסידורי 1 ועד למקום ה-N-1: מיקומו של האיבר המינימלי הוא 3, מכיוון ש-7 הוא האיבר הקטן ביותר. נחליף בין האיבר במקום ה-1, לאיבר במקום ה-3.	
בסריקה השלישית נמצא את מיקומו של האיבר המינימלי מהמקום שמספרו הסידורי 2 ועד למקום ה- N-1: מיקומו של האיבר המינימלי הוא 3, מכיוון ש-15 הוא האיבר הקטן ביותר. נחליף בין האיבר במקום ה-2 לאיבר במקום ה-3.	
בסריקה הרביעית והאחרונה, נמצא את מיקומו של האיבר המינימלי מהמקום שמספרו הסידורי 3 ועד למקום ה-N-1: מיקומו של האיבר המינימלי הוא 3, מכיוון ש-23 הוא האיבר הקטן ביותר. במקרה זה ההחלפה לא משנה את סדר האיברים.	

כדי למצוא מינימלי במערך אנו רגילים להשתמש במשתנה מינימלי תורן. במיון בחירה יש צורך **להחליף** בין הערך המינימלי, לערך הקיים במיקום אליו יש להעביר את המינימלי ולכן יש לדעת גם את מיקומו. למעשה, ניתן להסתפק רק במיקום של הערך מינימלי כפי שיוצג באלגוריתם הבא. המשתנה השומר את המקום של הערך המינימלי בכל שלב הוא pmin.

אלגוריתם למיון בחירה	**Java**

מיון_בחירה(a)
{טענת כניסה: הפעולה מקבלת מערך a }
{טענת יציאה: הפעולה ממיינת את אברי המערך בסדר עולה }
1) **עבור** i מ-0 עד (1 - אורך_מערך(a)) **בצע**
pmin ← i
עבור j מ-(i+1) עד אורך_מערך(a) **בצע**
אם a[j]<a[pmin] **אז**
pmin ←j
temp ←a[i]
a[i] ←a[pmin]
a[pmin] ←temp
	static void selectionSort(**int** a[])
	{
	int temp, pmin;
	for (**int** i=0 ; i<a.length-1; i++)
	{
	pmin=i;
	for (**int** j=i+1; j<a.length; j++)
	if (a[j]<a[pmin])
	pmin = j;
	temp = a[i];
	a[i] = a[pmin];
	a[pmin] = temp;
	}
	}

טבלת מעקב עבור מערך a בגודל 5:

מערך a	**i**	**pmin**	**j**	**החלף**
0 1 2 3 4 23 15 6 7 30	0	0	1	
		1		
		2		
0 1 2 3 4 6 15 23 7 30				(a[0], a[2])
	1	1	2	
		3		
0 1 2 3 4 6 7 23 15 30				(a[1], a[3])
	2	2	3	
		3		
0 1 2 3 4 6 7 15 23 30				(a[2], a[3])
	3	3	4	
				(a[3], a[3])

רגע חושבים: מה יש לשנות במיון כדי שהמערכים יהיו ממוינים בסדר יורד?

כמה פעולות מתבצעות במיון זה? לולאה חיצונית בגודל n-1 ולולאה פנימית בגודל מתקצר:

n-1, n-2, n-3,… 1

כלומר מספר ההשוואות המתבצעות הוא סכום של סדרה חשבונית שהאיבר הראשון שלה הוא 1 והאיבר האחרון שלה הוא n-1 וההפרש שלה הוא 1, והוא: $(n-1)n/2$.

מיון בועות - bubble sort

מיון בועות של מערך חד-ממדי מתבסס על סריקת המערך, והשוואה של כל זוג איברים צמודים. אם זוג איברים אינו מסודר בסדר הרצוי יש להחליף ביניהם.
מיון זה נקרא גם בשם מיון החלפת שכנים.

דוגמה עבור מערך a בגודל 5 :

15	23	7	6	30

בסריקה הראשונה נבצע את הבדיקות הבאות:

מציין הסריקה	המערך לפני ההשוואה של a[i] עם a[i+1]	תנאי ההחלפה	המערך לאחר ביצוע החלפה של a[i] עם a[i+1]
0	0 1 2 3 4 15 23 7 6 30	a[0] > a[1] ? 15 > 23 ? שקר אין החלפה	0 1 2 3 4 15 23 7 6 30
1	0 1 2 3 4 15 23 7 6 30	a[1] > a[2] ? 23 > 7 ? אמת החלפה בין a[1] ↔ a[2]	0 1 2 3 4 15 7 23 6 30
2	0 1 2 3 4 15 7 23 6 30	a[2] > a[3] ? 23 > 6 ? אמת החלפה בין a[2] ↔ a[3]	0 1 2 3 4 15 7 6 23 30
3	0 1 2 3 4 15 7 6 23 30	a[3] > a[4] ? 23 > 30 ? שקר אין החלפה	0 1 2 3 4 15 7 6 23 30

במעבר הראשון התבצעו מספר החלפות. התוצאה המוחלטת של מעבר זה היא **שהערך המקסימלי במערך הגיע למקומו – המקום האחרון במערך.** תוצאה זו מובטחת מאחר וכאשר מציין הסריקה נמצא על תא המערך בו נמצא הערך המקסימלי, מכאן ואילך הוא תמיד יהיה גדול מן הערך הבא אחריו (השכן שלו) ולכן יתחלף איתו וכך (יבעבע) עד למקום האחרון במערך.

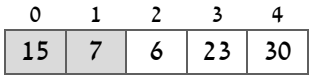
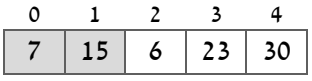
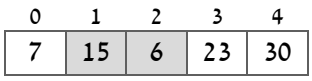
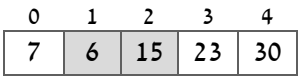
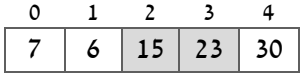
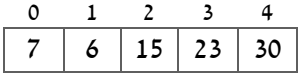
התהליך המתבצע מסביר גם את השמות שניתנו לשיטה זו :

החלפת שכנים – על שום ההשוואה בין שכנים והחלפה לפי הצורך.

בועות – על שום הערך המקסימלי בכל שלב שמבעבע ומגיע למקומו בתום הסריקה.

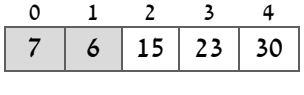
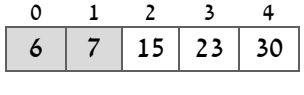
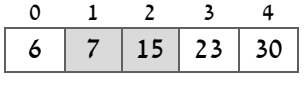
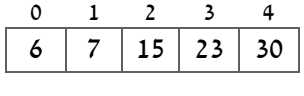
מאחר ואחרי הסריקה הראשונה האיבר המקסימלי הגיע למקומו אין צורך לשוב ולהשוות אליו, לכן הסריקה השנייה תהיה עד המקום הלפני האחרון. בסריקה השנייה השני בגודלו יגיע למקום השני מסוף המערך, וכך הלאה....

בסריקה השנייה נבצע את הבדיקות הבאות:

מציין הסריקה	המערך לפני ההשוואה של a[i] עם a[i+1]	תנאי ההחלפה	המערך לאחר ביצוע החלפה של a[i] עם a[i+1]
0		a[0] > a[1] ? 15> 7 ? אמת החלפה בין a[0] ↔ [a1]	
1		a[1] > a[2] ? 15 > 6 ? אמת החלפה בין a[2] ↔ a[1]	
2		a[2] > a[3] ? 15 > 23 ? שקר אין החלפה	

שים ♥ : אין צורך לבדוק אם 23>30 מאחר והערך 30 - המקסימלי נמצא כבר במקומו. בתום הסריקה השנייה, השני בגודלו נמצא בוודאות במקום הלפני אחרון במערך.

בסריקה השלישית נבצע את הבדיקות הבאות:

מציין הסריקה	המערך לפני ההשוואה של a[i] עם a[i+1]	תנאי ההחלפה	המערך לאחר ההשוואה של a[i] עם a[i+1]
0		a[0] > a[1] ? 7 > 6 ? אמת החלפה בין a[1] ↔ a[0]	
1		a[1] > a[2] ? 7 > 15 ? שקר אין החלפה	

שים ♥ בשיטה זו <u>בכל סריקה</u> איבר אחד תופס את מקומו ולכן במערך בגודל n נדרשות במקרה הגרוע ביותר n-1 סריקות.

אלגוריתם למיון בועות	Java

מיון בועות (a)
{טענת כניסה: הפעולה מקבלת מערך a }
{טענת יציאה: הפעולה ממיינת את המערך בסדר עולה }
1) **עבור** i מ- 0 עד (אורך_מערך(a) - 1) **בצע**
3.1) **עבור j** מ-0 עד (אורך_מערך(a) - i -1) **בצע**
3.1.1) **אם** a[j] > a[j+1] **אז**
temp ← a[j]
a[j] ← a[j+1]
a[j+1] ← temp
	static void bubbleSort (**int** a[])
	{
	int temp ;
	for (**int** i=0 ; i<a.length-1; i++)
	{
	for (**int** j=0 ; j<a.length-i-1 ; j++)
	if (a[j]>a[j+1])
	{
	temp = a[j];
	a[j] = a[j+1];
	a[j+1] = temp;
	}
	}
	}

כמה פעולות מתבצעות במיון זה?

באלגוריתם זה עבור מערך בגודל n הלולאה החיצונית מתבצעת n-1 פעמים והלולאה הפנימית מתבצעת בפעם הראשונה n-1 פעמים, בפעם השנייה n-2 וכן הלאה... ולכן גם כאן מספר פעולות ההשוואה המתבצעות הוא: $n/2(n-1)$.

האם אפשר לייעל את האלגוריתם הנ"ל?

בשיטת מיון זו ניתן לייעל את האלגוריתם כך שביצועו יסתיים לאחר סריקה מלאה שלא היו בה החלפות – כלומר, כאשר המערך ממוין. לצורך כך נשתמש בדגל שלפי ערכו בסיום כל סריקה תהיה אפשרות לדעת אם הייתה החלפה, כלומר המערך לא ממוין ולכן יש להמשיך לסריקה נוספת, או שלא הייתה החלפה ולכן המערך כבר ממוין וניתן להפסיק את תהליך המיון. במקרה זה נוסיף תנאי על הדגל ללולאה החיצונית.

משתנה הדגל בפתרון שלפניך הוא sorted.

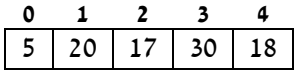
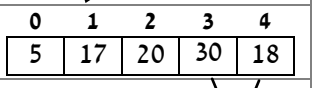
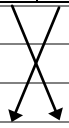
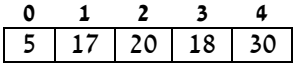
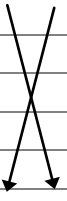
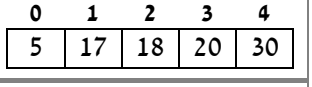
```
static void bubbleSort2 (int a[])
{
   int temp ;
   boolean sorted=false;
   for (int i=0 ; i<a.length-1 && !sorted ; i++)
   {
      sorted = true;
      for (int j=0 ; j<a.length-i-1 ; j++)
            if (a[j]>a[j+1])
            {
               temp = a[j];
               a[j] = a[j+1];
               a[j+1] = temp;
               sorted = false;
            }
   }
}
```

המשתנה sorted מקבל את הערך true לפני כל סריקה פנימית של המערך. אם לא הייתה החלפה במשך כל הסריקה יישאר ערכו של sorted – true, לכן הלולאה החיצונית לא תתבצע פעם נוספת. אם בוצעה לפחות החלפה אחת, כלומר יתכן שהמערך אינו ממוין, יהפוך ערכו של sorted להיות false כדי לוודא ביצוע של סריקה נוספת (הלולאה החיצונית תתבצע לפחות עוד פעם אחת).

לסיכום:

האלגוריתם למיון בועות עם דגל יביא לכך שבתהליך המיון שלחלק מן המערכים יחסכו סריקות חוזרות כאשר המערך כבר ממוין. יחד עם זאת יש לשים לב שנוספו פעולות לניהול של המשתנה sorted.

טבלת מעקב עבור מערך בגודל 5:

i	(i<a.length-1) && (! sorted)	sorted	j	a[j]>a[j+1]	מערך a
0	true	false			
		true			
			0		
				5 >20 false	
			1		
		false		20 > 17 true	
			2		
				20 >30 false	
			3		
		0		30 > 18 true	
1	true	true			
			0		
				5 > 17 false	
			1		
				17 > 20 false	
			2		
		false		20 > 18 true	
2	true	true			
			0		
				5 > 17 false	
			1		
				17 > 18 false	

סריקה ראשונה

סריקה שנייה

סריקה שלישית

שים לב

מכיוון שבביצוע השלישי של הלולאה החיצונית עבור i=2 לא התבצעו החלפות, נשאר ערכו של true sort דבר המעיד על כך שהמערך כבר ממוין ולכן אין צורך בביצוע נוסף של הלולאה החיצונית.

האם ניתן לעשות תהליך שיפור דומה על מיון בחירה?

מיון הכנסה - Insertion sort

מיון הכנסה הוא מיון בו בכל שלב מוכנס איבר אחד למערך ממוין כך שנשמר סדר המיון במערך.
נתאר את המיון בשני שלבים:
שלב ראשון: הכנסה של איבר חדש למערך ממוין תוך שמירה על סדר המיון.
שלב שני: קליטת ערכים ומיונם לתוך מערך בשיטת הכנסה כך שבכל שלב האיברים במערך ממוינים.
האלגוריתם שיוצג כאן ימיין רשימת נתונים מן הקלט לתוך מערך כך שבכל שלב ייקלט איבר אחד ויוכנס למערך תוך שמירה על סדר המיון שלו. באופן זה בתום הקלט יתקבל מערך ממוין.

שלב ראשון: הכנסה של איבר חדש למערך ממוין

לדוגמה נתון המערך הבא, שחמשת האיברים הראשונים שלו ממוינים:

5	7	9	11	13	

num הוא הערך שיש להכניס לתוך המערך הממוין

num	המערך לאחר ההכנסה	הסבר
3	3 5 7 9 11 13	3<5 ולכן עליו להכנס ולהיות ראשון במערך הממוין
10	5 7 9 10 11 13	9<10 וגם 10<11 ולכן עליו להכנס ביניהם במערך הממוין

תהליך הכנסה של איבר חדש למערך ממוין:

1. איתור מיקום ההכנסה של הערך החדש.
2. הזזת כל האיברים במערך הגדולים מן הערך החדש כדי לפנות עבורו מקום.
3. הכנסת הערך החדש במקום שהתפנה עבורו.

הגדרת הפעולה:

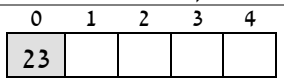
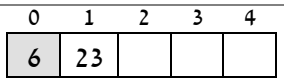
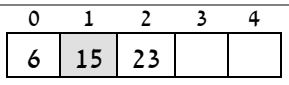
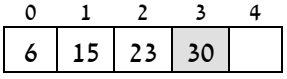
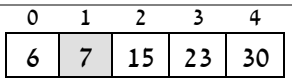
אלגוריתם להכנסת ערך למערך ממוין	Java

הכנס-למערך-ממוין(a, n, num)
{טענת כניסה: הפעולה מקבלת מערך a, את מספר האיברים הממוינים הנמצאים במערך – n, ואת הערך להכנסה num }
{טענת יציאה: הפעולה מכניסה את הערך החדש num למערך a כך שנשמר סדר המיון }
1) i ← 0
2) **כל עוד** (i<n) וגם (a[i]<num) **בצע**
i ← i + 1
3) **עבור** j מ- n עד (i+1) **בצע**
a[j] ← a[j-1]
4) a[i] ←num
	static void addToSortArray(**int**[] a,
	int n, **int** num)
	{
	int i = 0;
	for (i=0 ; i<n && a[i]<num ; i++);
	for (**int** j=n ; j>i ; j--)
	a[j] = a[j-1];
	a[i] = num;
	}

שלב שני: מיון ערכי קלט למערך בשיטת הכנסה

כדי למיין רשימת מספרים המתקבלת מן הקלט לתוך מערך, נפעיל על כל ערך שנקלט את הפעולה הכנס-למערך-ממוין וכך בכל שלב המערך יהיה ממוין, ובסוף תהליך הקליטה כל ערכי הקלט יהיו ממוינים בתוך המערך.

נעקוב אחר התהליך עבור רשימת הקלט הבאה משמאל לימין: 7, 30, 23, 6, 15 ובעבור מערך ריק:

הערך	המערך אחרי ההכנסה	הסבר
23		הערך 23 יוכנס למקום הראשון במערך.
6		הערך 6 קטן יותר מכל האיברים במערך לכן יוכנס להיות ראשון.
15		הערך 15 קטן מ- 23 וגדול מ- 6 ולכן יוכנס לפני 23.
30		הערך 30 גדול יותר מכל אברי המערך ולכן יוכנס בסוף.
7		הערך 7 קטן מ- 15 וגדול מ- 6 ולכן יוכנס לפני 15.

לפניך אלגוריתם לפעולה למיון הכנסה המשתמש בפעולה הכנס-למערך-ממוין. הפעולה מקבלת את גודל רשימת הנתונים, קולטת את הנתונים ומחזירה מערך ממוין המכיל אותם. size הוא מספר הערכים הנקלטים וממוינים.

אלגוריתם למיון הכנסה	Java

מיון-הכנסה(size)
{טענת כניסה: הפעולה מקבלת מספר שלם size המציין את מספר הערכים שיש לקלוט }
{טענת יציאה: הפעולה קולטת size ערכים, ממיינת אותם למערך ומחזירה אותו }
1) **עבור** i מ- 0 עד size-1 **בצע**
1.1) קבל כקלט ערך ל- num
1.2) הכנס-למערך-ממוין(a, i, num)
2) **החזר** a
	static int[] insertionSort(**int** size)
	{
	int[] a = **new** int[size];
	int num;
	for (**int** i=0 ; i<size ; i++)
	{
	System.out.print("Enter value:");
	num = reader.nextInt();
	addToSortArray(a, i, num);
	}
	return a;
	}

כמה פעולות מתבצעות במיון זה?

גם באלגוריתם זה עבור מיון של n ערכים מן הקלט הלולאה החיצונית מתבצעת n פעמים. בתוך הפעולה הכנס-למערך-ממוין יש שתי לולאות, אך הראשונה (חיפוש המיקום) מגיעה עד למקום שממנו והלאה ממשיכה השנייה (דחיפת האיברים). סה"כ הפעולות שיתבצעו הם בהתאם למספר האיברים הקיימים כבר באותו זמן במערך כלומר בהתחלה 1 עד לסוף n-1. מכאן, שמספר הפעולות שמבצע האלגוריתם הוא בקרוב כמו במיונים הקודמים.

תרגילים: מיונים - פתח ויישם אלגוריתם

שאלה 1: מיון יורד בשיטת הבחירה ✯

כתוב פעולה הממיינת מערך בסדר יורד בשיטת מיון בחירה.

שאלה 2: מיון יורד בשיטת בועות ✯

כתוב פעולה הממיינת מערך בסדר יורד בשיטת מיון בועות.

שאלה 3: מיון יורד בשיטת הכנסה ✯

כתוב פעולה הממיינת מערך בסדר יורד בשיטת מיון הכנסה.

שאלה 4: מיון חלקי ✯

כתוב פעולה הממיינת חלק ממערך, החל מהמקום start ועד למקום last.

שאלה 5: מערך בשני חצאים ✯

כתוב פעולה הממיינת בנפרד את אברי החצי הראשון של המערך, ואת אברי החצי השני של המערך. השתמש בפעולה שהגדרת בשאלה 4.

שאלה 6: מיון בחלקים ✯✯

כתוב פעולה המקבלת מערך וערך נוסף N וממיינת את המערך בחלוקה ל- N חלקים. כלומר, מיון N האיברים הראשונים ביניהם, מיון N האיברים הבאים ביניהם, וכן הלאה. השתמש בפעולה שהגדרת בשאלה 4. הנחה: אורך המערך מתחלק ב- N.

שאלה 7: מצטערים מפטרים... ✯✯

במפעל ובו 500 עובדים נדרשים לבצע צמצומים. לצורך כך ביקש בעל המפעל לקבל שתי רשימות: האחת של 5 העובדים החדשים ביותר, השנייה של 10 העובדים הוותיקים ביותר כדי לאפשר להם לצאת לפנסיה מוקדמת.

פתח ויישם בשלבים אלגוריתם הקולט למערך אחד את מספר העובד, ולמערך שני את הוותק של כל עובד, בהתאמה. על האלגוריתם להציג כפלט את שתי הרשימות. לכל עובד ברשימה יש להדפיס את מספרו הסידורי ואת הוותק שלו.

שאלה 8: מיון שורות במערך ✯✯

פתח ויישם אלגוריתם המקבל מערך דו-ממדי של מספרים, וממיין כל שורה במערך בסדר יורד. לצורך מיון כל שורה השתמש באחת משיטות המיון שהוצגו.

שים ❤: שורה במערך דו-ממדי היא מערך חד-ממדי ולכן ניתן להשתמש בפעולות שהוגדרו עבור מערך חד-ממדי.

שאלה 9: מיון מוחלט ✯✯✯

מערך דו-ממדי נקרא "ממוין באופן מוחלט" אם מתקיימים בו התנאים: כל אחת מהשורות שלו ממוינת, כל אחת מהעמודות שלו ממוינת, וגם כל אחד מן האלכסונים ממוין. כל המיונים הם לפי אותו מפתח, עולה או יורד. כתוב פעולה המקבלת מערך דו-ממדי, ומחזירה true אם המערך "ממוין באופן מוחלט", או false אחרת. יש להשתמש בפעולות עזר.

חלק ב: שיטות חיפוש

חיפוש הוא תהליך בו מאתרים ערך מסוים בסדרת נתונים. תהליך החיפוש מחזיר את המיקום של ערך החיפוש בתוך סדרת הנתונים.
בחלק זה נכיר מספר שיטות לחיפוש של ערך במערך.
בעבור מערך חד-ממדי וערך נתון, נחפש האם הערך נמצא במערך. לפעמים די לומר האם הערך נמצא במערך או לא, ולפעמים רוצים לקבל את המיקום (מציין) של הערך במערך אם הוא נמצא.
האלגוריתמים שנציג בפרק זה הם: **חיפוש סדרתי** (לינארי) **במערך לא ממוין**, **חיפוש סדרתי** (לינארי) **במערך ממוין**, ו**חיפוש בינארי במערך ממוין**.

חיפוש סדרתי במערך לא ממוין

בהינתן מערך חד-ממדי לא ממוין וערך כלשהו, יש לבד אם הערך נמצא במערך ואם כן - לקבל את מיקומו במערך. נגדיר פעולה המקבלת את המערך ואת הערך לחיפוש. אם הערך נמצא במערך הפעולה תחזיר את מיקומו, אחרת אם אינו נמצא תחזיר את הערך 1- המסמן שהערך לא נמצא במערך.
המשמעות של המונח סדרתי (ליניארי) היא שהחיפוש מתבצע לפי סדר איבר אחר איבר.

אלגוריתם לחיפוש סדרתי	Java

חיפוש-סדרתי (a, value)
{**טענת כניסה**: הפעולה מקבלת מערך a וערך לחיפוש value}
{**טענת יציאה**: הפעולה מחזירה את מיקום הערך במערך - אם הוא קיים במערך, או 1- אם אינו קיים במערך}
1) i ←0
2) **כל עוד** i<(a)אורך_מערך **בצע**
אם a[i] = value **אז**
החזר i
אחרת
i ←i+1
3) **החזר** 1-
	להלן הפונקציה linearSearch:
	static int linearSearch(**int** a[], **int** value)
	{
	for (**int** i=0 ; i<a.length ; i++)
	if (a[i]==value)
	return i;
	return -1;
	}

שים ♥ עבור ערך שנמצא במערך, יפסיק החיפוש כאשר מצאנו אותו. עבור ערך שאינו נמצא במערך, יש להמשיך ולחפש עד לסוף המערך.

חיפוש סדרתי במערך ממוין

רגע חושבים:

אם ידוע לנו כי המערך ממוין האם אנו יכולים לשפר את החיפוש הסדרתי אחר ערך שאינו נמצא במערך?

התשובה היא: כן.

הכיצד? אם ידוע שהמערך ממוין, כאשר מגיעים לערך במערך הגדול מן הערך אותו אנו מחפשים ניתן להפסיק את החיפוש כי מאיבר זה ואילך כל האיברים גדולים ממנו.

שים ♥: אם הערך שאנו מחפשים אינו נמצא במערך אך גדול מכל אברי המערך לא יהיה חיסכון בחיפוש כי החיפוש יפסק רק לאחר שהושלמה סריקת המערך.

אלגוריתם לחיפוש סדרתי במערך ממוין	Java

חיפוש-סדרתי-במערך-ממוין (a)
{**טענת כניסה**: הפעולה מקבלת מערך ממוין a וערך לחיפוש value}
{**טענת יציאה**: הפעולה מחזירה את מיקום הערך במערך - אם נמצא במערך, או 1- אם הערך אינו נמצא במערך}
1) i ←0
2) **כל עוד** i<(a)אורך_מערך **בצע**
2.1) **אם** a[i]=value **אז**
החזר i
2.2) **אחרת**
אם a[i]>value **אז**
החזר -1
3) **החזר** -1
	static int linearSortSearch(**int** a[]**, int** value)
	{
	for (**int** i=0 ; i<a.length ; i++)
	{
	if (a[i]==value)
	return i;
	else
	if (a[i]>value)
	return -1;
	}
	return -1;
	}

חיפוש בינארי במערך ממוין (האריה במדבר)

חיפוש בינארי מתבצע רק **במערך ממוין**. למשל: חיפוש מספר טלפון על פי שם מנוי, מתבצע ברשימה ממוינת. הרעיון המרכזי בחיפוש זה המאפשר לנצל את מיון המערך הוא: בכל פעם פונים לאמצע טווח החיפוש. אם הערך שמחפשים לא נמצא באמצע, בודקים: אם החיפוש גדול מן הערך האמצעי. ניתן להמשיך ולחפש אותו רק בחצי העליון של טווח החיפוש, אחרת אם ערך החיפוש קטן מן הערך האמצעי ניתן להמשיך ולחפש אותו רק בחצי התחתון של טווח החיפוש.
בדרך זו אנו מצמצמים את טווח החיפוש בכל שלב פי 2.

דוגמה: נתון הערך num =15 והמערך הממוין a הבא:

0	1	2	3	4	5	6	7
1	3	6	12	15	23	30	60

המטרה היא לבדוק האם num (ערכו 15) הוא איבר במערך a.
המשתנים המשתתפים בתהליך הם:
low מציין של האיבר הראשון בטווח החיפוש
high מציין של האיבר האחרון בטווח החיפוש
middle מציין של האיבר האמצעי בטווח החיפוש
נגדיר תחילה את גבולות החיפוש. אם מעוניינים לחפש במערך כולו, טווח החיפוש יהיה (0–a.length-1). המשתנה low יקבל בהתחלה את הערך 0 תחילת טווח החיפוש, ואילו המשתנה high יקבל את הערך 7 סוף טווח החיפוש (num -1). בכל שלב נחשב את middle על-ידי 2 /(low+high).

low	high	middle	טווח החיפוש במערך	בדיקת היחס בין num לבין a[middle]	מסקנה
0	7				
0	7	(0+7)/2 3	טווח החיפוש 0-7 (המערך כולו) 0 1 2 3 4 5 6 7 1 3 6 12 15 23 30 60	האם num = a[middle] ? 15=a[3]? 15=12? לא האם num<a[middle] ? 15<a[3]? 15<12? לא	המסקנה היא ש-num>a[middle], ומאחר והמערך ממוין, num יכול להימצא רק מהמקום ה-middle+1 ועד למקום ה-high. נהפוך את ערכו של low להיות middle+1.
4	7	(4+7)/2 5	טווח החיפוש קטן פי 2, והפך להיות בטווח 4-7 0 1 2 3 4 5 6 7 1 3 6 12 15 23 30 60	האם num = a[middle] ? 15=a[5]? 15=23? לא האם num<a[middle] ? 15<a[5]? 15<23? כן	מאחר והמערך ממוין, המסקנה היא num יכול להימצא רק מהמקום ה-low למקום ה-middle-1. נהפוך את ערכו של high להיות middle-1.
4	4	(4+4)/2 4	טווח החיפוש קטן פי 2, והפך להיות בטווח 4-4 0 1 2 3 4 5 6 7 1 3 6 12 15 23 30 60	האם num = a[middle] ? 15=a[4]? 15=15? כן	מצאנו את num במקום ה-middle.

מה יקרה אם num לא נמצא במערך? מתי יפסק החיפוש?
החיפוש יפסק כאשר נגמר טווח החיפוש, כלומר כאשר low יהיה גדול מ-high.

אלגוריתם לחיפוש בינארי	Java

חיפוש-בינארי (a, num)
{**טענת כניסה**: הפעולה מקבלת מערך a וערך לחיפוש num }
{**טענת יציאה**: הפעולה מחזירה את מיקום הערך במערך - אם הערך קיים במערך, או 1- אם הערך אינו קיים במערך
1) low ← 0
2) high ← אורך_מערך(a)-1
3) middle ← -1
4) **כל עוד** low<=high **בצע**
4.1) middle ← (low+high)/2
4.2) **אם** num=a[middle] **אז**
middle **החזר**
4.3) **אחרת**
אם num<a[middle] **אז**
high ← middle-1
אחרת
low ← middle+1
5) **החזר** 1-
	static int binarySearch(**int** a[], **int** num)
	{
	int low=0, high=a.length-1, middle=-1;
	while (low<=high)
	{
	middle = (low+high)/2;
	if (num==a[middle])
	return middle;
	else
	if (num<a[middle])
	high = middle-1;
	else // num>a[middle]
	low = middle+1;
	}
	return -1;
	}

טבלת מעקב עבור הערך num=15 והמערך הממוין a הבא:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	3	6	12	15	23	30	60	70	75	80	90	92	95	98

BinarySearch	low	high	middle	a[middle]=value	a[middle]>value	a[middle]<value
	0	14	(0+14)/2=7	false	true	false
	0	7	(0+7)/2=3	false	false	true
	4	7	(4+7)/2=5	false	true	false
	4	4	(4+4)/2=4	true		
4						

רגע חושבים:

- מי מהחיפושים יעיל יותר לדעתך? נמק תשובתך.
- בהנחה שבמערך ממוין 16 איברים, כמה פעמים לכל היותר תתבצע הלולאה בחיפוש סדרתי, וכמה פעמים תתבצע הלולאה לכל היותר בחיפוש בינארי?
- בדוק גם עבור מערך בן 32 איברים. מה הכלל?

שאלה 10: חיפוש בינארי ☆☆

נתון מערך שממוין בסדר יורד. כתוב פעולה שמחפשת איבר במערך בשיטת החיפוש הבינארי.

שאלה 11: איבר בין שני מציינים ☆☆

כתוב פעולה המקבלת מערך, ערך לחיפוש ושני מציינים במערך. הפעולה תחזיר את מיקום הערך בתוך קטע המערך בין שני המציינים. אם הערך אינו נמצא בקטע מערך זה תחזיר הפעולה 1-.

א. עבור מערך ממוין

ב. עבור מערך לא ממוין

שאלה 12: בדיקת קיום הימצאות בחצאים ☆☆

כתוב פעולה המקבלת מערך ומחזירה true אם כל ערך המופיע בחצי הראשון שלו מופיע גם בחצי השני שלו.

א. עבור מערך ממוין

ב. עבור מערך לא ממוין

שאלה 13: חיפוש טרינארי ☆☆☆☆

חיפוש טרינארי עובד בדומה לחיפוש בינארי על מערך ממוין. בחיפוש זה במקום לחצות כל קטע ל-2, יש לחלק כל קטע ל-3. כתוב פעולה המקבלת מערך ממוין וערך לחיפוש, ומחזירה את מיקומו במערך לפי חיפוש טרינארי. אם האיבר לא נמצא, יוחזר 1-.

חלק ג: מיזוג

תהליך מיזוג מתבצע בין מערכים ממוינים. בהינתן שני מערכים ממוינים (או יותר), המטרה לבנות מערך חדש המכיל את כל האיברים מן המערכים הממוינים מבלי להפעיל על המערך החדש אלגוריתם מיון. כלומר, לנצל את היות המערכים ממוינים. בחלק זה נציג אלגוריתם למיזוג שני מערכים ממוינים.

דוגמה: עבור שני המערכים הבאים

0	1	2
e	g	i

0	1	2
a	b	h

יהיה מערך המיזוג:

0	1	2	3	4	5
a	b	e	g	h	i

תהליך המיזוג

הרעיון המרכזי באלגוריתם המיזוג הוא שבכל שלב משווים איבר מן המערך הראשון לאיבר מן המערך השני. הקטן ביניהם מועתק למערך החדש. האלגוריתם מתייחס בכל פעם לאיבר תורן אחד מכל מערך. כאשר משווים זוג איברים, והערך של הקטן מועתק למערך החדש יהיה האיבר הבא באותו מערך לאיבר התורן הבא.

תהליך המיזוג יתבצע באופן הבא:

1. קובעים מציין בתחילת כל מערך.
2. בודקים: אם האיבר במציין של המערך הראשון קטן מן האיבר במציין של המערך השני, מעתיקים אותו למערך החדש ומקדמים את המציין במערך הראשון.
   אחרת מעתיקים את האיבר מן המערך השני למערך החדש, ומקדמים את המציין במערך השני.
3. חוזרים על שלב 2, עד שהמציין של אחד המערכים חורג מגבול המערך שלו.
4. מעתיקים למערך החדש את שארית האיברים הממוינים מן המערך ממנו לא הועתקו כל האיברים.

א. האם המערכים שיש למזג צריכים להיות זהים באורכם?
תשובה: לא. ראה דוגמה להלן.

ב. מה קורה כאשר אחד מן המערכים מסתיים?

ג. בהנחה שמספר האיברים במערך הראשון n ומספר האיברים במערך השני הוא m.
כמה איברים יהיו במערך המיזוג?

ד. מה יש לשנות באלגוריתם המיזוג אם המערך הממוזג צריך להיות ללא כפילויות? כמה איברים יהיו במערך המיזוג במקרה זה?

דוגמה: עבור שני המערכים הבאים

0	1	2
10	17	90

0	1	2	3	4
1	3	12	100	110

יהיה מערך המיזוג:

0	1	2	3	4	5	6	7
1	3	10	12	17	90	100	110

הדגמת תהליך המיזוג

שלב	מערך המיזוג נבנה בשלבים	בדיקה	j	i	מערך שני – מציין j	מערך ראשון – מציין i
ראשון	0 1 2 3 4 5 6 7 1	1<10 true	0	0	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
שני	0 1 2 3 4 5 6 7 1 3	3<10 true	1	0	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
שלישי	0 1 2 3 4 5 6 7 1 3 10	12<10 false	2	0	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
רביעי	0 1 2 3 4 5 6 7 1 3 10 12	17<12 false	2	1	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
חמישי	0 1 2 3 4 5 6 7 1 3 10 12 17	100<17 false	3	1	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
שישי	0 1 2 3 4 5 6 7 1 3 10 12 17 90	90<100 true	3	2	0 1 2 3 4 1 3 12 100 110	0 1 2 10 17 90
שביעי	0 1 2 3 4 5 6 7 1 3 10 12 17 90 100 110		3		בשלב זה מועתקים כל האיברים שנותרו במערך זה	

Java	אלגוריתם למיזוג

static int[] merge(**int** a1[], **int** a2[])
{
  int p1=0, p2=0, p3=0;
  int[] a3= **new int**[a1.length+a2.length];
  while (p1<a1.length && p2<a2.length)
  {
  if (a1[p1]<a2[p2])
  {
  a3[p3] = a1[p1];
  p1++;
  }
  else
  if (a2[p2]<a1[p1])
  {
  a3[p3] = a2[p2];
  p2++;
  }
  else
  {
  a3[p3] = a1[p1];
  p3++;
  a3[p3] = a2[p2];
  p1++;
  p2++;
  }
  p3++;
  }
  while (p1<a1.length)
  {
  a3[p3] = a1[p1];
  p1++;
  p3++;
  }
  while (p2<a2.length)
  {
  a3[p3] = a2[p2];
  p2++;
  p3++;
  }
  return a3;
}
	מיזוג (a1, a2)
	{טענת כניסה: האלגוריתם מקבל שני מערכים ממוינים a1,a2}
	{טענת יציאה: האלגוריתם מחזיר מערך ממוין חדש על-פי תהליך המיזוג }
	1) p1 ← 0
	2) p2 ← 0
	3) p3 ← 0
	4) **כל עוד** (p1<(a1)אורך_מערך) **וגם**
	p2< (a2)אורך_מערך **בצע**
	4.1) **אם** a1[p1]<a2[p2] **אז**
	a3[p3] ← a1[p1]
	p1 ← p1+1
	אחרת
	אם a2[p2]<a1[p1] **אז**
	a3[p3] ← a2[p2]
	p2 ← p2+1
	אחרת {שוויון}
	a3[p3] ← a1[p1]
	p3 ← p3+1
	a3[p3] ← a2[p2]
	p1 ← p1+1
	p2 ← p2+1
	4.2) p3 ← p3+1
	5) **כל עוד** (p1< (a1)אורך_מערך) **בצע**
	5.1) a3[p3] ← a1[p1]
	5.2) p1 ← p1+1
	5.3) p3 ← p3+1
	6) **כל עוד** (p2< (a2)אורך_מערך) **בצע**
	6.1) a3[p3] ← a2[p2]
	6.2) p2 ← p2+1
	6.3) p3 ← p3+1

שאלה 13: מערך מיזוג ללא כפילויות ✯

באלגוריתם המיזוג שמופיע בעמוד 84 המערך הממוזג יכול להכיל איברים חוזרים בהתאם להופעתם במערכים a1 ו-a2.

שנה את האלגוריתם כך שאיברים זהים בין שני המערכים יועתקו למערך המיזוג רק פעם אחת.

שאלה 14: מערך מיזוג ✯

פתח ויישם אלגוריתם המקבל כקלט שני מערכים הראשון ממוין בסדר עולה והשני ממוין בסדר יורד. האיברים בתוך כל מערך שונים. האלגוריתם מחזיר מערך ממוזג הממוין בסדר עולה וללא חזרות.

שאלה 15: מערך מיזוג ✯✯

פתח ויישם אלגוריתם המקבל כקלט שני מערכים ממוינים בסדר עולה ובהם יש איברים חוזרים. האלגוריתם מחזיר מערך ממוזג כך שכל איבר במערך המיזוג יופיע פעם אחת בלבד.

שאלה 16: מיזוג משולש ✯✯

פתח ויישם אלגוריתם המקבל כקלט שלשה מערכים ממוינים בסדר עולה. האלגוריתם מחזיר מערך חדש ממוזג ובו הערכים משלושת המערכים. במערך המיזוג לא תהיינה כפילויות.

שאלה 17: איברים משותפים ✯✯✯✯

כתוב פעולה המקבלת שני מערכים ממוינים המכילים מספרים שלמים. על הפעולה להדפיס את האיברים המשותפים לשני המערכים. שים לב! התייחס בפתרונך לעובדה שהמערכים ממוינים. הצע שני אלגוריתמים לפתרון השאלה (ניתן לכתוב פעולה זו בעזרת לולאה אחת בלבד). ציין איזה אלגוריתם יעיל יותר ונמק תשובתך.

דגשים לסיכום הפרק

שיטות למיון n נתונים

השיטה		הסבר	כמה פעולות מתבצעות במיון זה?
מיון בחירה	**Selection sort**	בשיטה זו מוצאים מינימלי בכל שלב (בכל סריקה של המערך) ומעבירים אותו למקומו על פי סדר המיון.	לולאה חיצונית בגודל n-1 ולולאה פנימית בגודל מתקצר: n-1, n-2, n-3, 1 מספר ההשוואות המתבצעות הוא: 1+2+3,….+ n-1
מיון בועות (החלפת שכנים)	**Bubble sort**	בשיטה זו משווים כל זוג איברים צמודים. אם הזוג אינו מסודר בסדר הרצוי מחליפים ביניהם. בתום סריקה אחת הערך הגדול מגיע לסוף המערך.	<u>בכל סריקה</u> איבר אחד תופס את מקומו ולכן במערך בגודל n נדרשות במקרה הגרוע ביותר n-1 סריקות. לולאה חיצונית בגודל n-1 ולולאה פנימית בגודל מתקצר: n-1, n-2, n-3, 1
מיון הכנסה	**Insertion Sort**	במיון הכנסה בכל שלב מוכנס איבר אחד למערך ממוין כך שנשמר סדר המיון במערך.	עבור מיון של n ערכים הלולאה החיצונית מתבצעת n פעמים. סה"כ הפעולות שיתבצעו הם בהתאם למספר האיברים הקיימים באותו זמן במערך כלומר בהתחלה 1 עד ל- n-1.

שיטות חיפוש במערך בגודל n

השיטה	הסבר	כמה פעולות מתבצעות?
חיפוש סדרתי במערך <u>לא ממוין</u>	בשיטה זו משווים כל ערך במערך לערך החיפוש ומחזירים את המציין שלו במערך אם הוא קיים.	חיפוש סדרתי של ערך במערך דורש לכל היותר פנייה לכל אחד מאברי המערך – כאשר הוא לא נמצא. אם הוא נמצא יפסק החיפוש קודם. במקרה הגרוע ביותר: n השוואות.
חיפוש סדרתי במערך <u>ממוין</u>	בשיטה זו משווים כל ערך במערך לערך החיפוש. אם ערך החיפוש קיים במערך מחזירים את המציין שלו. הנתון שהמערך ממוין מאפשר להפסיק את החיפוש כאשר נמצא ערך הגדול מערך החיפוש כי משלב זה ואילך כל האיברים גדולים ממנו.	חיפוש סידרתי של ערך במערך ממוין דורש לכל היותר פניה לכל אחד מאברי המערך. אם ערך החיפוש נמצא <u>או</u> אם נמצא איבר גדול ממנו - יפסק החיפוש קודם. במקרה הגרוע ביותר: n השוואות.
חיפוש בינארי במערך ממוין	חיפוש בינארי מתבצע רק **במערך ממוין**. הרעיון המרכזי בחיפוש זה המאפשר לנצל את מיון המערך הוא: בכל פעם פונים לאמצע טווח החיפוש. אם הערך לא נמצא שם, אזי בודקים אם הוא גדול מן הערך האמצעי או קטן ממנו ובהתאם לכך מצמצמים את טווח החיפוש פי 2 בכל פעם.	בחיפוש בינארי מצמצמים בכל שלב את טווח החיפוש <u>פי 2</u>. במקרה הגרוע ביותר: $\log_2 n$ השוואות. הפיתוח המתמטי המסביר את מספר הפניות לאמצע קטע החיפוש הנותר הוא מעבר לנדרש כאן. אך חשוב לציין שערך זה קטן בהרבה מ- n ככל ש n גדול יותר.