סיכום – אלגוריתמי מיון

1. **מיון בועות (Bubble Sort)**

מיון מערך ע"י השוואה בין כל 2 איברים סמוכים במערך:
נעבור על המערך n פעמים ובכל פעם נחליף בין 2 איברים סמוכים אם השמאלי יותר גדול (אחרת לא נעשה דבר) עד שאחרי המעבר הראשון האיבר הגדול ביותר יגיע לסוף, במעבר השני, האיבר השני בגודלו יגיע למקומו וכן הלאה.
דוגמא עבור המערך: $[5,3,7,1,6,4]$ - נעבור על המערך ונתחיל להשוות בין כל 2 איברים סמוכים. $3 > 5$ ולכן נחליף ביניהם ונקבל: $[3,5,7,1,6,4]$, כעת נמשיך עם 5, נשווה עם 7 ולא נחליף $(5 \ngtr 7)$, נעבור ל 7 ונמשיך להחליף עד שנקבל: $[3,5,1,6,4,7]$. כעת נחזור לתחילת המערך (מעבר שני) ונבצע את אותה פעולה כך ש 6 יגיע למקומו: $[3,1,5,4,6,7]$. לאחר המעבר השלישי: $[1,3,4,5,6,7]$ - המערך ממוין.
סיבוכיות האלגוריתם: עוברים על המערך n פעמים ובכל פעם עוברים על המערך, בכל פעם על פחות איברים: $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=O(n^2)$
מימוש:

```java
public static void bubbleSort(int[] a) {
	boolean flag = true;
	for (int i = 0; i < a.length && flag; i++) {
		flag = false;
		for (int j = 0; j < a.length-i-1; j++) {
			if(a[j]>a[j+1]){
				flag = true;
				swap(a,j,j+1);
			}
		}
	}
}
```

הסבר: נבצע את התהליך (הלולאה החיצונה) כמספר איברי המערך – כי צריך במקרה הגרוע להזיז את כל האיברים למקום שלהם, או שנעצור את הלולאה אם באחד המעברים לא ביצענו אפילו החלפה אחת – כלומר, המערך כבר ממוין – לשם כך יש את המשתנה flag שאומר האם ביצענו במעבר לפחות החלפה אחת (ואז צריך להמשיך) או שלא ביצענו החלפות (והמערך ממוין). בלולאה הפנימית אנו עוברים על כל האיברים מההתחלה עד הסוף אך בכל פעם עד אחד פחות כי בסוף כל מעבר – האיבר הגדול ביותר מהנותרים כבר הגיע למקומו בסוף ולכן אין צריך לבדוק אותו שוב. בתוך הלולאה הפנימית אנו בודקים מי הגדול מבין 2 איברים סמוכים, אם השמאלי יותר גדול אנו מבצעים החלפה בין ה 2, אחרת אנו לא עושים דבר ומתקדמים הלאה. בכל מקרה נבצע תמיד השוואה עם הגדול ביותר שמצאנו עד עכשיו ולכן הוא יידחף לסוף.

2. **מיון הכנסה (Insertion Sort)**

מיון מערך ע"י מעבר על המערך ועבור כל איבר נבצע החלפה עם הסמוך לו מלפניו עד שהאיבר שלפניו יהיה קטן ממנו.
דוגמא עבור המערך: $[5,3,7,1,6,4]$ - נעבור על המערך ונתחיל להשוות את 3 עם 5 ונבצע החלפה: $[3,5,7,1,6,4]$. נעבור ל 7 ולא נחליף: $(5<7)$. נמשיך ל 1 ונחליף עם 7 שיותר גדול ממנו: $[3,5,1,7,6,4]$, כעת נמשיך להזיז את 1 עד המקום המתאים לו: $[3,1,5,7,6,4]$ ואז $[1,3,5,7,6,4]$. נחזור ל 6 ונמשיך באותה דרך עד הסוף: $[1,3,4,5,6,7]$.
סיבוכיות האלגוריתם: המקרה הגרוע: כאשר המערך ממוין בסדר יורד: עוברים על המערך ועבור כל איבר, נבצע החלפה עד שנזיז אותו למקום הראשון, בכל פעם יותר הזזות:
$1+2+\cdots+(n-1)=\frac{n(n-1)}{2}=O(n^2)$.
מימוש:

```
public static void insertionSort(int[] a) {
	for (int i = 1; i < a.length; i++) {
		int j = i;
		while(j>0 && a[j-1]>a[j]){
			swap(a,j-1,j);
			j--;
		}
	}
}
```

הסבר: בלולאה החיצונה אנו עוברים על כל האיברים (החל מהשני) ועבור כל איבר מתחילים להשוות עם הקודם לו, אם יש צורך בהחלפה, נבצע החלפה ונמשיך לאיבר שלפני וכך נמשיך להזיז במידת הצורך את האיבר אחורה עד המקום הסידורי המתאים לו (כלומר, עד שהגענו לאיבר שיותר קטן או עד שהגענו להתחלה במידה והאיבר שלנו קטן מכולם) המשתנה j יחזיק בתוכו את המיקום של האיבר שעובדים עליו (ואם ביצענו החלפה אחורה אז j יורד ב 1), לאחר שסיימנו להזיז את האיבר שלנו, נעבור (ע"י הלולאה החיצונה) לאיבר הבא.

3. מיון בחירה (Selection Sort)

מיון מערך ע"י מעבר על המערך n פעמים ובכל פעם נמצא את האיבר הקטן ונשים אותו במקומו כך שבמעבר הראשון על המערך נמצא את האיבר המינימאלי ונחליף אותו עם האיבר הראשון, במעבר השני נמצא את האיבר הקטן ביותר השני (כלומר, האיבר הכי קטן חוץ מהראשון) ונחליף אותו עם האיבר במקום השני.

דוגמא עבור המערך: $[5,7,3,1,6,4]$ - נעבור על המערך ונחפש את האיבר הכי קטן: 1. נחליף אותו עם האיבר הראשון: $[1,7,3,5,6,4]$. נעבור פעם שנייה על המערך החל מהאיבר השני ונמצא את הקטן ביותר: 3, ונחליף אותו עם האיבר השני: $[1,3,7,5,6,4]$. נמשיך כך עד האיבר האחרון ונקבל מערך ממוין: $[1,3,4,5,6,7]$.

סיבוכיות האלגוריתם: עוברים על המערך n פעמים ובכל פעם מחפשים איבר מינימאלי במערך, בכל פעם עוברים על פחות איברים: $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=O(n^2)$

מימוש:

```
public static void selectionSort(int[] a) {
	int minIndex;
	for (int i = 0; i < a.length; i++) {
		minIndex = getMinIndex(a,i);
		swap(a,i,minIndex);
	}
}
```

הסבר: נבצע את התהליך (הלולאה החיצונה) כמספר איברי המערך. בכל פעם נמצא את האיבר הקטן ביותר ע"י פונקציה שעוברת על המערך החל מ i ומוצאת את האיבר המינימאלי ומחזירה את המיקום שלו, ונשים אותו בהתחלה, לאחר מכן נמשיך לחפש את המינימאלי החל מהאיבר השני ונשים אותו במקום השני וכן הלאה עד שנגיע לסוף המערך – ושם נשים את הנותר – האיבר המקסימאלי.

4. מיון מניה (Counting Sort)

מיון מערך שבו טווח האיברים חסום (כמו המספרים השלמים). נבדוק מיהם האיברים המקסימאלי והמינימאלי במערך ונמצא ע"י כך את טווח האיברים (max-min). לאחר מכן, נבנה מערך חדש שהאינדקסים שבו ייצגו את האיבר עצמו והערך בכל אינדקס ייצג כמה פעמים הופיע האיבר המיוצג ע"י האינדקס במערך המקורי.

דוגמא: עבור המערך [6,0,3,4,2,3] אנו רואים כי האיבר המקסימאלי הוא 6 והאיבר המינימאלי הוא 0 ולכן נגדיר מערך בגודל 7 כדי שיהיו בו האינדקסים 0 עד 6. נעבור על המערך המקורי ועבור כל איבר ניגש למקום במערך החדש המייצג את האיבר ונוסיף לו 1 כדי לציין שהאיבר

הופיע. עבור 6 ניגש למערך במקום 6 ונוסיף שם 1, וכן הלאה עד סוף המערך, נקבל שהמערך החדש יכיל: $[1,0,1,2,1,0,1]$ - כל איבר, כמה פעמים הוא הופיע. לאחר מכן, נסרוק את המערך של המופעים ונכניס כמספר המופעים את האיבר למערך המקורי (נדרוס את האיברים), בדוגמא: 0 הופיע פעם אחת ולכן נכניס למערך המקורי פעם אחת 0, נמשיך ל 1 – לא הופיע ולכן נמשיך הלאה ונכניס את 2 פעם אחת, 3 הופיע פעמיים ולכן נכניס אותו פעמיים וכן הלאה, בסופו של דבר נקבל מערך ממוין: $[0,2,3,3,4,6]$.
מה קורה כאשר הטווח לא מתחיל מ 0? הרי אינדקס של מערך מתחיל מ 0? הפתרון הוא לבצע הזזה, כלומר להתייחס לאיבר המינימאלי כ 0 וכל השאר יזוזו בהתאם אליו. לדוגמא: אם המערך הוא: $[4,2,8,4,3,5]$ - האיבר המינימאלי הוא 2 ולכן נתייחס ל 2 כ 0 ואם 2 מופיע אז נוסיף למופע של 0, אם 3 יופיע – נוסיף למופע של 1 וכן הלאה (פשוט כל איבר יזוז ב 2, הזזה = חיסור הערך של האיבר המינימאלי מכל איבר) וגם כשנחזיר את המערך, אם נראה ש 0 הופיע פעם אחת – נשים 2 במערך פעם אחת וכן הלאה. אם האיבר המינימאלי הוא מספר שלילי נבצע הזזה באיבר הזה: אם המערך: $[3,-2,-1,5,3,2]$ אז נבצע הזזה ב -2 (הזזה זה האיבר פחות המינימום ולכן כאן זה יהיה ועוד 2 כי
$a[i-(-2)] = a[i+2]$ ולכן -2 ייוצג ע"י האינדקס 0 , -1 ייוצג ע"י 1 וכן הלאה.
<u>סיבוכיות האלגוריתם</u>: עוברים על המערך המקורי פעם אחת, על המערך שיוצרים פעם אחת (גודלו הוא גודל הטווח) ואז עוברים על מספר המופעים (השווה לגודל המערך המקורי) ולכן בסה"כ הסיבוכיות היא: $O(n+k)$ כאשר: n - גודל המערך המקורי ו k - גודל הטווח. אם הטווח קטן מ n נקבל שהסיבוכיות היא: $O(n)$.
<u>מימוש</u>:

```java
public static void countingSort(int[] a) {
	int min = a[0];
	int max = a[0];
	int k = 0;
	for (int i = 0; i < a.length; i++) {
		if(a[i]>max) max = a[i];
		else if(a[i]<min) min = a[i];
	}
	int[] freq = new int[max-min+1];
	for (int i = 0; i < a.length; i++) {
		freq[a[i]-min]++;
	}
	for (int i = 0; i < freq.length; i++) {
		for (int j = 0; j < freq[i]; j++) {
			a[k++] = i+min;
		}
	}
}
```

<u>הסבר</u>: נמצא את הטווח ע"י מציאת המקסימום והמינימום במערך (הלולאה הראשונה). לאחר מכן, נגדיר את מערך התדירויות (הסופר כמה פעמים מופיע כל איבר) בגודל $max - min + 1$ כדי לכלול את כל הטווח (כולל הקצוות). נעבור על המערך המקורי וכל מופע של איבר נוסיף באינדקס המתאים לו ($freq[a[i] - min]$ - בשביל ההזזה באיבר המינימאלי כדי שהאיבר המינימאלי ייוצג ע"י 0 וכן הלאה, הזזה = להחסיר מכל איבר את הערך של האיבר המינימאלי). בלולאה האחרונה נעבור על מערך המופעים ועבור כל איבר נכניס למערך המקורי את אותו איבר כמספר הפעמים שהוא הופיע – הלולאה האחרונה הפנימית (המשתנה k מצביע על המיקום במערך המקורי שאליו נכניס כל איבר ומתקדם בכל הכנסה).
<u>ההשמה</u>: $i + min$ היא כדי להחזיר את ההזזה שעשינו, לדוגמא: אם 2 היה האיבר המינימאלי אז האינדקס 0 במערך המופעים יצג את 2 והערך במקום ה 0 היה מספר המופעים של 2 ולכן כשאנו מכניסים למערך המקורי, צריך להכניס 2 ולא 0 ולכן מוסיפים את 2 בחזרה.

5. מיון בסיס (Radix Sort)

מיון מערך שבו טווח הספרות או האותיות במילה חסום. אנו ממיינים בכל שלב לפי ספרה אחת, מתחילים מהספרה הימנית ביותר וממיינים לפי הספרה הזו (למי שיש פחות ספרות, הספרה הימנית נחשבת כ 0), לאחר מכן ממיינים לפי הספרה הבאה אחריה וכן הלאה עד הספרה השמאלית ביותר.

דוגמא עבור המערך: $[567,144,908,76,4,23,566]$ נמצא את האיבר הגדול ביותר כדי לדעת מה מספר הספרות שלו וכך נדע מה מספר הספרות המקסימאלי. לאחר שמצאנו (908) נעבור על הספרות: נתחיל מספרת היחידות ונמיין: $[23,144,4,76,566,567,908]$ נעבור לספרת העשרות: $[4,908,23,144,566,567,76]$, נעבור למאות: $[4,23,76,144,566,567,908]$ וקיבלנו מערך ממוין. (בכל מיון, אם לא הייתה קיימת ספרה – זה נחשב כ 0).

ניתן להכליל את המיון גם למספרים המיוצגים בבסיס אחר (לא עשרוני)

סיבוכיות האלגוריתם: עוברים על מספר הספרות של האיבר המקסימאלי ועבור כל ספרה, נעבור על כל הספרות האפשריות ועבור כל ספרה אפשרית נעבור על כל המערך: $O(b \cdot n \cdot \log k)$ כאשר n - גודל המערך, k - האיבר המקסימאלי, $\log k$ - מספר הספרות של האיבר המקסימאלי ו b - הבסיס. אם הבסיס של המספרים ידוע: $O(n \cdot \log k)$ ואם מספר הספרות של המספר המקסימאלי קטן מ n נקבל שהסיבוכיות היא: $O(n)$.

מימוש:

```
public static void radixSort(int[] a) {
	int base = 10;
	int max = a[0],digit = 1;
	for (int i = 0; i < a.length; i++) {
		if(a[i]>max) max = a[i];
	}
	int maxDigits = getNumberOfDigits(max);
	for(int d = 0; d < maxDigits; d++) {
		int[] temp = new int[a.length];
		int pos = 0;
		for (int i = 0; i < base; i++) {
			for (int j = 0; j < a.length; j++) {
				int theDig = (a[j]/digit)% base;
				if (theDig == i) {
					temp[pos++] = a[j];
				}
			}
		}
		for (int i = 0; i < a.length; i++) {
			a[i] = temp[i];
		}
		digit*=10;
	}
}
```

הסבר: נמצא את האיבר המקסימאלי (הלולאה הראשונה) ונשמור את מספר הספרות שלו ב maxDigits ע"י חישוב של מספר הספרות באמצעות הפונקציה *getNumberOfDigits* – פונקציה שמקבלת מספר שלם ומחזירה את מספר הספרות שלו (חלוקה ב 10 עד לקבלת 0). לאחר מכן, נעבור על כל ספרה (החל מספרת האחדות – המשתנה digit מייצג על איזו ספרה אנו עומדים, בהתחלה הוא שווה ל 1) – המעבר מתבצע ע"י הלולאה על d, נגדיר מערך זמני שישמור את התוצאה של המיון לפי כל ספרה ומשתנה pos שיצביע על המקום במערך שאליו נכניס את האיברים. נעבור על כל סוגי הספרות מהקטנה עד הגדולה (המשתנה base שומר את כמות הספרות האפשריות – הבסיס שבמקרה שלנו הוא עשרוני) ועבור כל ספרה נעבור על כל המערך ולכל איבר, ניקח אותו ונחתוך את הספרות הראשונות ע"י $a[i]/digit$ שמשמיט את כל הספרות הימניות עד $digit$ ואז $\%base$ שייקח את הימנית ביותר ממה שנשאר. ואם יש שוויון

בספרות אז נכניס את כל האיבר למערך הזמני ומכיוון שאנו עוברים על הספרות בסדר עולה (מ 0 עד 9), המערך הזמני יהיה ממוין לפי הספרה שאנו עומדים עליה. בסוף התהליך, נעתיק בחזרה למערך המקורי – הלולאה האחרונה ונתקדם למיין לפי הספרה הבאה.

6. **<u>מיון מיזוג (Merge Sort)</u>**

מיון מערך בצורה רקורסיבית ע"י פירוק המערך ל 2 בכל פעם עד לקבלת תתי מערכים בגודל 1 ואז מיזוג של כל 2 בצורה ממוינת: נעבור על 2 תתי המערכים הממוינים וניקח בכל פעם את הקטן יותר מ 2 האיברים הראשונים עד שנגמר אחד מהמערכים ולאחר מכן, נעתיק את המשך המערך שעדיין לא הסתיים. מכיוון שתת מערך בגודל 1 הוא תמיד ממוין נקבל שלאורך כל הדרך אנו ממזגים מערכים ממוינים למערך ממוין.

<u>דוגמא</u> עבור המערך: $[5,7,3,1,6,4]$ - נפעיל את הפונקציה באופן רקורסיבי על תתי המערכים: $[5,7,3]$ ו $[1,6,4]$, $[5,7,3]$ יפעיל את הפונקציה על תתי המערכים: $[5]$ ו $[7,3]$, $[7,3]$ ייתפרק ל $[7]$ ו $[3]$. $[5]$ יחזור כמערך ממוין, $[7]$ ו $[3]$ יחזרו כמערכים ממוינים, נבצע מיזוג ביניהם: $[3,7]$ ונחזיר את המערך הממוין, נבצע מיזוג עם $[5]$: $[3,5,7]$ ונחזיר את המערך הממוין. אותו תהליך יתבצע עבור הצד השני: $[1,6,4]$ יתפרק ויתמזג למערך: $[1,4,6]$. בסוף התהליך, נמזג את 2 תתי המערכים שחזרו: $[3,5,7]$, $[1,4,6]$ למערך אחד: $[1,3,4,5,6,7]$ - וקיבלנו מערך ממוין.

<u>סיבוכיות האלגוריתם</u>: בשלב הראשון אנו רק "מפרקים" את המערך לתתי מערכים ע"י חלוקה ל 2 חלקים שווים עד לקבלת מערך בגודל 1 ולכן כמות הפעמים שנצטרך לחלק ב 2 היא $\log_2 n$.
לאחר מכן, עבור כל 2 תתי מערכים אנו מבצעים מיזוג. גודלם של 2 תתי המערכים ביחד הוא לכל היותר כגודל המערך המקורי (בשלב האחרון בו אנו ממזגים את 2 החצאים למערך אחד ממוין) שהוא n ולכן בסה"כ הסיבוכיות היא: $O(n \cdot \log n)$.

<u>מימוש</u>:

```
public static void mergeSort(int[] a) {
	mergeSort(a,0,a.length);
}

private static void mergeSort(int[] a, int low, int high) {
	int n = high - low;
	if(n <= 1) return;
	int mid = (low + high)/2;
	mergeSort(a,low,mid);
	mergeSort(a,mid,high);
	int i = low, j = mid, k = 0;
	int[] temp = new int[n];
	while(i<mid && j<high) {
		if(a[i] < a[j]) temp[k++] = a[i++];
		else temp[k++] = a[j++];
	}
	while(i<mid) temp[k++] = a[i++];
	while(j<high) temp[k++] = a[j++];
	for (int l = 0; l < n; l++) {
		a[low + l] = temp[l];
	}
}
```

<u>הסבר</u>: מכיוון שהפונקציה היא רקורסיבית ותנאי העצירה הוא כאשר גודל המערך שווה ל 1 נזדקק לפונקצית מעטפת – זו הפונקציה המקורית שמקבלת את המערך וקוראת לפונקציה שעושה את כל החישוב. זאת מכיוון שפונקציה רקורסיבית צריכה לקבל ערך משתנה בכל פעם כפרמטר ואילו את המערך אנו לא משנים (אנו לא באמת מקטינים אותו לגודל 1 אלא רק זזים על האינדקסים) אלא מועברים בכל פעם ערכי תחילת תת המערך וסוף תת המערך.

התהליך מתבצע כך: תחילה נחשב את גודל תת המערך שאנו עובדים עליו:`n = high - low`
אם גודל תת המערך הוא 1 אז תת המערך ממוין (רק איבר אחד) ולכן סיימנו (`return;`). אחרת, צריך להמשיך לפרק את תת המערך ל 2 תתי מערכים: מההתחלה עד האמצע ומהאמצע עד הסוף. ולכן קוראים לפונקציה *mergeSort* שוב על 2 תתי המערכים כדי לבצע מיון עליהם. לאחר ש 2 תתי המערכים כבר

ממוינים –(וזו הרקורסיה – הפעולה זהה על כל תת מערך בכל גודל והבסיס הוא תת מערך בגודל 1 שהוא מערך ממוין ולכן תמיד נמזג 2 מערכים ממוינים), נותר למזג אותם למערך אחד ממוין וזאת ע"י אלגוריתם מיזוג. נגדיר תחילה את המערך הזמני שישמור את תוצאת המיזוג. ונגדיר לכל מערך מצביע לאינדקס של האיבר הראשון שלו - i יצביע לאינדקס של תת המערך השמאלי המתחיל ב low ו j יצביע לתת המערך הימני המתחיל מהאמצע (mid) ו k יצביע על תחילת המערך הזמני ויתקדם בכל הכנסה של איבר למערך. נעבור על 2 המערכים עד שאחד מהם יגיע לסוף שלו (סוף תת המערך השמאלי הוא ב mid וסוף הימני הוא ב $high$) ונבדוק האם האיבר שאנו עומדים עליו במערך ה 1 קטן מהאיבר שאנו עומדים עליו במערך השני, אם כן, הוא ייכנס ראשון למערך התוצאה ונתקדם לאיבר הבא במערך ה 1, אחרת, נכניס את האיבר מהמערך ה 2 ונתקדם לאיבר הבא במערך ה 2. תהליך זה ייגמר ברגע שאחד המערכים מסתיים ואז מה שנותר זה להכניס את האיברים שנותרו במערך שלא נגמר למערך התוצאה. ולכן נבדוק מי מהמערכים עדיין לא נגמר ונוסיף את האיברים שלו למערך התוצאה. הלולאה האחרונה מעתיקה את מערך התוצאה למקומו במערך המקורי (יש לזכור כי אנו במערך המקורי נמצאים על תת המערך המתחיל מ low ומסתיים ב $high$ ולכן אנו מעתיקים את האיברים לטווח הזה ולכן יש הזזה: $a[i + low]$ כדי להתחיל מ low.

7. <u>מיון מהיר (Quick Sort)</u>

מיון מערך בצורה רקורסיבית ע"י בחירה של איבר כלשהו (ניקח בדרך כלל את האיבר הראשון) במערך והעברת כל האיברים הגדולים ממנו לצד ימין והקטנים ממנו לצד שמאל. לאחר מכן, מתמקדים באופן רקורסיבי בכל תת מערך בנפרד (הימני והשמאלי) ושוב בוחרים את האיבר הראשון ומעבירים את הגדולים ממנו לצד ימין (בתת המערך) והקטנים ממנו לצד שמאל (בתת המערך) עד שנגיע לתתי מערכים בגודל 1. בסוף התהליך נקבל מערך ממוין.

לדוגמא: עבור המערך:

6	3	9	7	1	4	8	5

נבחר את האיבר הראשון – 6 ונסדר את האיברים הגדולים מ 6 בצד ימין ואת האיברים הקטנים מ 6 בצד שמאל. לשם כך נגדיר 2 מצביעים – אחד מצביע על האינדקס שבהתחלה (1 - האיבר 3, כי 0 זה האינדקס של 6) ואחד מצביע על האיבר האחרון (7 – האיבר 5). בכל שלב נבדוק: אם האיבר שמשמאל נמצא במקומו (כלומר הוא קטן מ 6) אז נקדם את המצביע לאיבר הבא עד שנגיע לאיבר שלא נמצא במקום שלו, ברגע שמצאנו איבר שלא במקומו נעבור למצביע של האיברים מימין, אם האיבר נמצא במקומו (כלומר הוא גדול מ 6) אז נקדם את המצביע לאיבר הקודם (כלומר נתקדם שמאלה) עד שגם מימין נגיע לאיבר שאינו במקומו ואז נוכל לבצע החלפה בין האיברים. בדוגמא שלנו: 3 קטן מ 6 ולכן נקדם את המצביע לאיבר הבא.

6	3	9	7	1	4	8	5

9 גדול מ 6 ולכן הוא לא נמצא במקום שלו. מכאן נעבור למצביע

שבסוף, 5 לא גדול מ 6 ולכן 2 האיברים לא במקומם – נבצע החלפה:

6	3	5	7	1	4	8	9

נמשיך להתקדם באותו אופן, 7 לא במקומו כי הוא גדול מ 6, נעבור לצד ימין: 8 במקומו, נתקדם, 4 לא במקומו – נבצע החלפה:

ג

6	3	5	4	1	7	8	9

הגענו למצב שבו 2 המצביעים מצביעים לאותו מקום ולכן סיימנו את הסריקה – נותר להעביר את 6 למקומו : נחליף את 6 עם המקום אליו הגענו עם המצביעים.

1	3	5	4	6	7	8	9

כעת, נפרק את המערך ל 2 תתי מערכים לפי האיבר 6:

1	3	5	4

7	8	9

ונבצע מיון באותה הדרך על כל תת מערך בנפרד עד שנגיע לתתי מערכים בגודל אחד ונקבל בסוף את המערך הממוין.

<u>סיבוכיות האלגוריתם</u>: אנו עוברים בכל פעם לכל היותר כגודל המערך: n (בפעם הראשונה ולאחר מכן אנו עוברים על פחות) ומספר הפעמים שאנו מבצעים את הפעולה זה כמספר

החלוקות ב 2 של המערך עד לקבלת תתי מערכים בגודל 1 - $\log_2 n$. ולכן סה"כ הסיבוכיות היא: $O(n \cdot \log n)$. אך כל זאת במידה וחילקנו בכל פעם ל 2 חלקים שווים וזה תלוי בבחירת האיבר שבאמצע. ולכן, המקרה הגרוע - אם המערך ממוין – בכל פעם ניקח את האיבר הכי קטן והחלוקה תהיה 1 בצד שמאל ו $n-1$ בצד ימין ופעולה זו תתבצע n פעמים (בכל פעם נתקדם באיבר אחד). והסיבוכיות תהיה: $O(n^2)$.

<u>מימוש:</u>

```java
public static void quickSort(int[] a) {
	quickSort(a,0,a.length-1);
}

private static void quickSort(int[] a, int low, int high) {
	if(low<high) {
		int pivot = partition(a, low, high);
		quickSort(a, low, pivot-1);
		quickSort(a, pivot+1, high);
	}
}

private static int partition(int[] a, int low, int high) {
	int pivot = low++;
	while (low<=high){
		if (a[low] <= a[pivot]) low++;
		else if (a[high] > a[pivot]) high--;
		else swap(a, low, high);
	}
	swap(a, high, pivot);
	return high;
}
```

<u>הסבר</u>: פונקציה זו היא רקורסיבית ולכן נזדקק למעטפת. תנאי העצירה של הרקורסיה הוא כאשר גודל תת המערך הוא 1. (כלומר $low = high$, ההתחלה והסוף של המערך זה אותו איבר) בכל שלב נכנס לפונקציה `partition` שלוקחת את האיבר הראשון: low (האינדקס שלו) ומשתמשת ב 2 מצביעים: low המתחיל מ 1 יותר (כי הראשון הוא איבר האמצע) ו $high$ ועד שהמצביעים לא הגיעו לאותו מקום נבדוק: אם האיבר במקום low נמצא במקומו (כלומר הוא קטן מ `pivot` – איבר האמצע) אם כן, נקדם את low ימינה. ברגע שמצאנו איבר עם אינדקס low שלא במקומו. נעבור לבדוק את האיברים מימין: אם האיבר במקום ה $high$ נמצא במקומו נקדם את $high$ שמאלה עד שנגיע גם שם לאיבר שלא במקומו. ברגע ש 2 הצדדים לא במקומם – נבצע החלפה ביניהם. בסוף התהליך נחליף בין איבר האמצע למקום אליו הגענו עם המצביעים כדי לשים את איבר האמצע במקום שלו. הפונקציה `partition` מחזירה את המיקום של איבר האמצע כדי שנפעיל באופן רקורסיבי את `quickSort` על תת המערך שמשמאל לאיבר האמצע ועל תת המערך שמימין לאיבר האמצע.

8. ***<u>מיון ב Java</u>***

ב java קיימת מחלקה המכילה פונקציות המטפלות במערכים – Arrays.בין הפונקציות, קיימת הפונקציה Arrays.Sort(arr) המקבלת מערך arr וממיינת אותו בסיבוכיות $O(n \cdot \log n)$.