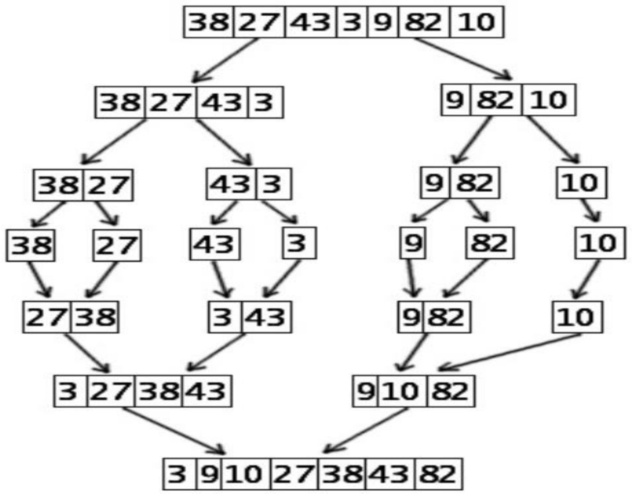
תרגול 1
הפרד ומשול

אלגוריתמים 1 סמסטר א תשפ"ג

הפרד ומשול

במדעי המחשב, **הפרד ומשול** היא תבנית תכנון אלגוריתמים חשובה.
היא מבוססת על שבירה רקורסיבית של הבעיה לשתיים או יותר תת-בעיות מאותה הצורה (או צורה דומה לה), עד שהבעיות הופכות לפשוטות דיין כדי שניתן יהיה לפתור אותן ישירות(מקרה בסיס). לאחר מכן הפתרונות לתת הבעיות משולבים יחד כדי לתת פתרון לבעיה המקורית.



מגדלי האנוי
Hanoi Towers

האגדה מספרת על מקדש בהודו בו הכוהנים מעבירים מגדל בן 64 קומות. ברגע בו הם יסיימו, העולם יגיע לקיצו.

שאלה:

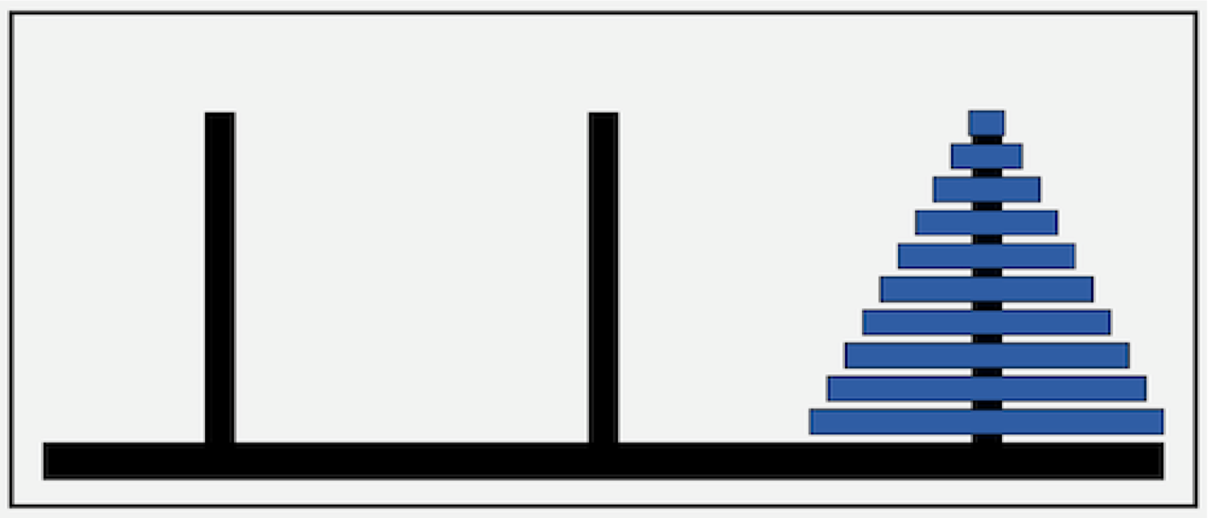
כמה צעדים הם צריכים על מנת להעביר את המגדל?

תיאור המשחק:

המשחק כולל שלושה מוטות אנכיים ("המגדלים") ומספר דסקיות בגדלים שונים שניתן להשחיל על המוטות. בתחילת המשחק, הדסקיות מסודרות על פי הגודל על אחד המוטות, כשהגדולה ביותר למטה והקטנה ביותר למעלה. מטרת המשחק להעביר את כל הדסקיות ממוט אחד לאחר.

חוקי המשחק:

1. מותר להזיז רק דסקית אחת בכל פעם.
2. אסור לשים דסקית על דסקית שקטנה ממנה.



- <u>נסמן:</u>

n - מספר הקומות במגדל

h(n) - את מספר הצעדים שעלינו לעשות כדי להעביר את המגדל כולו ממוט A אל מוט C.

- עבור n=1: **h(n)=1**
- עבור n=2:

א) מעבירים את הדסקית הקטנה מ-**A** ל-**B** (נסמן את השלב ב- **AB**)

ב) מעבירים את הדסקית הגדולה מ-**A** ל-**C** (נסמן את השלב ב- **AC**)

ג) מעבירים את הדסקית הקטנה מ-**B** ל-**C** (נסמן את השלב ב- **BC**)

סה"כ שלושה שלבים: **h(2)=2h(1)+1=3**

- עבור n=3:

א) מעבירים שתי דסקיות עליונית מ-**A** ל-**B** בעזרת **C** - 3 צעדים.

ב) מעבירים את הדסקית התחתונה (הגדולה) מ-**A** ל- **C** - צעד אחד.

ג) מעבירים שתי דסקיות מ-**B** ל-**C** בעזרת **A** - 3 צעדים.

סה"כ שבעה שלבים: **h(3)=2h(2)+1=7**

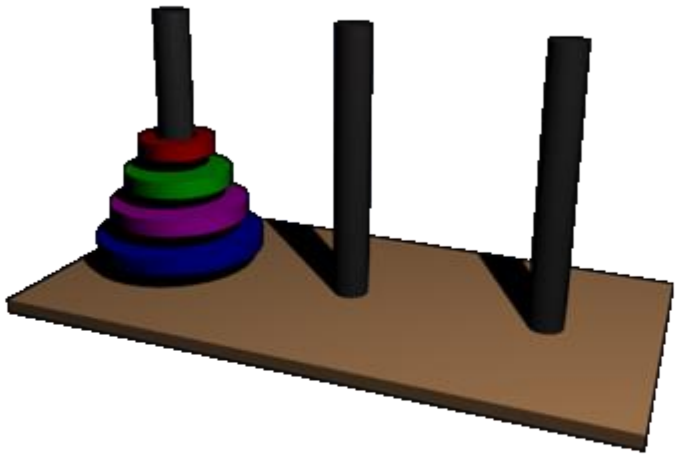
- עבור n כלשהו:

א) מעבירים **n-1** דסקיות עליונית מ-**A** ל-**B** בעזרת **C** ← **h(n-1)** צעדים.

ב) מעבירים את הדסקית התחתונה (הגדולה יותר) מ-**A** ל-**C** ← צעד **אחד**.

ג) מעבירים **n-1** דסקיות מ-**B** ל-**C** בעזרת **A** ← **h(n-1)** צעדים.

סה"כ: **h(n)=2h(n-1)+1**



Pseudo Code:

```
Hanoi (n, A, B, C):
    If n==1
        Print (“moving from” + A + “ to ” + C)
        return 1
    part1 = Hanoi (n-1, A, C, B)
    Print (“moving from” + A + “ to ” + C)
    part2 = Hanoi (n-1, B, A, C)
    return (1 + part1 + part2)
```

תזכורת - **משפט האב/ מאסטר:**

בהינתן נוסחת נסיגה לזמן ריצתו של אלגוריתם, ניתן להשתמש במקרים מסוימים בשיטה כדי למצוא חסם אסימפטוטי הדוק לזמן הריצה של האלגוריתם כולו. יתרון השיטה בכך שהיא ניתנת ליישום במגוון רחב של מקרים ומספקת פתרון מיידי, כמעט ללא חישוב.

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + n^k$$

1. $a < b^k$ — $T(n) \sim n^k$
2. $a = b^k$ — $T(n) \sim n^k \log_b n$
3. $a > b^k$ — $T(n) \sim n^{\log_b a}$

Longest Common Prefix

קלט: מערך מחרוזות באורך n

פלט: קידומת משותפת ארוכה ביותר.

דוגמא:

LCP([“algebra”,”algorithms”,”algory”,”algeirs”]) = “alg”

אינטואיציה:

$$LCP([s_1,\ldots,s_n]) = LCP(LCP([s_1,\ldots,s_{n/2}], LCP([s_{n/2+1},\ldots,s_n]))$$

נשתמש ב-Divide and Conquer.

בכל שלב:

- נחלק את המערך הנתון לשני חלקים
- נפעיל את האלגוריתם על כל אחד מהם
- נחזיר את הקידומת המשותפת הארוכה ביותר בין כל שתי תוצאות.

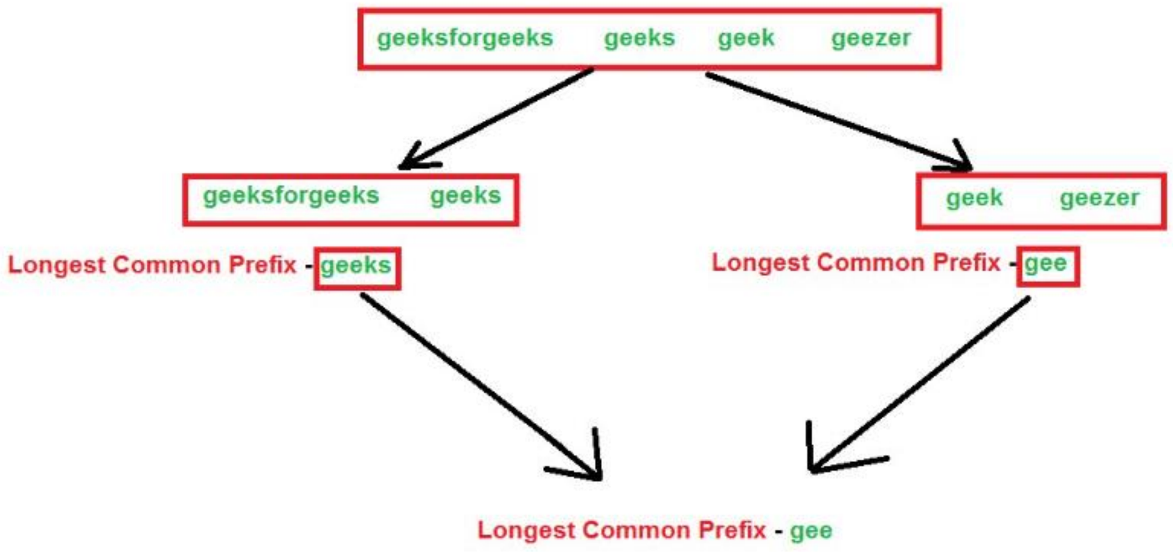
Pseudo Code:

LCP (strs[], s, e):

```
if s == e
    return strs[s]
mid = (s + e) /2
lcpl = LCP(strs, s, mid)
lcpr = LCP(strs, mid + 1, e)
return common_prefix (lcpl, lcpr)
```

Common_prefix (s1, s2):

```
length = min(s1.size, s2.size)
for i = 0 to (length-1)
    if s1.char_at(i) != s2.char_at(i)
        return s1.substring(0, i-1)
return s1.substring(0, length-1)
```



Maximum Sub Array

קלט: מערך של מספרים שלמים $A = [a_1,\dots,a_n]$

פלט: תת מערך רציף בעל הסכום הגדול ביותר במערך.

דוגמא:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

רעיונות?

השיטה הנאיבית תהיה לעבור על כל תתי המערכים האפשריים בשתי לולאות.
זמן ריצה של שיטה זו תהיה $O(n^2)$

נשתמש ב-**הפרד ומשול אשר ישפר את זמן הריצה ל O(nlogn)**:

- נחלק את המערך לשני חלקים שווים באורכם ונמצא פתרון עבור כל אחד מהם.
  נסמן את הפתרונות: maxR, maxL
- מצא את הסכום המקסימלי הכולל את האלמנט <u>האמצעי</u> במערך.
  נסמן את הפתרון: maxM
- נחזר את המקסימום מבין השלושה.

-2	-5	6	-2	-3	1	5	-6

מצא את הסכום המקסימלי **הכולל** את האלמנט האמצעי במערך:

נעשה זאת כך:

```
MaxCrossingSum(arr,mid){
    sum=0
    left_sum=INT_MIN
    for (i=mid; i>=0;i--){
        sum+=arr[i]
        if (sum>left_sum) -> left_sum=sum
    }
    sum=0
    right_sum=INT_MIN
    for(i=mid;i<arr.length;i++){
        sum+=arr[i]
        if(sum>right_sum)->right_sum=sum
    }
Return max(right_sum+left_sum-arr[mid],right_sum,left_sum)
}
```

-2	-5	6	-2	-3	1	5	-6

Max(4,3,4+3)=7

```
Int MaxSubArr(int arr []){
      if arr.length==1 -> return arr[0]
      int mid=arr.length /2
      maxL=MaxSubArr (arr[0] to arr [mid])              //T(n/2)
      maxR=MaxSubArr (arr[mid] to arr[length-1])        //T(n/2)
      maxM=MaxCrossingSum(arr,mid)                       //O(n)
      Return max(maxR,maxL,maxM)
}
```

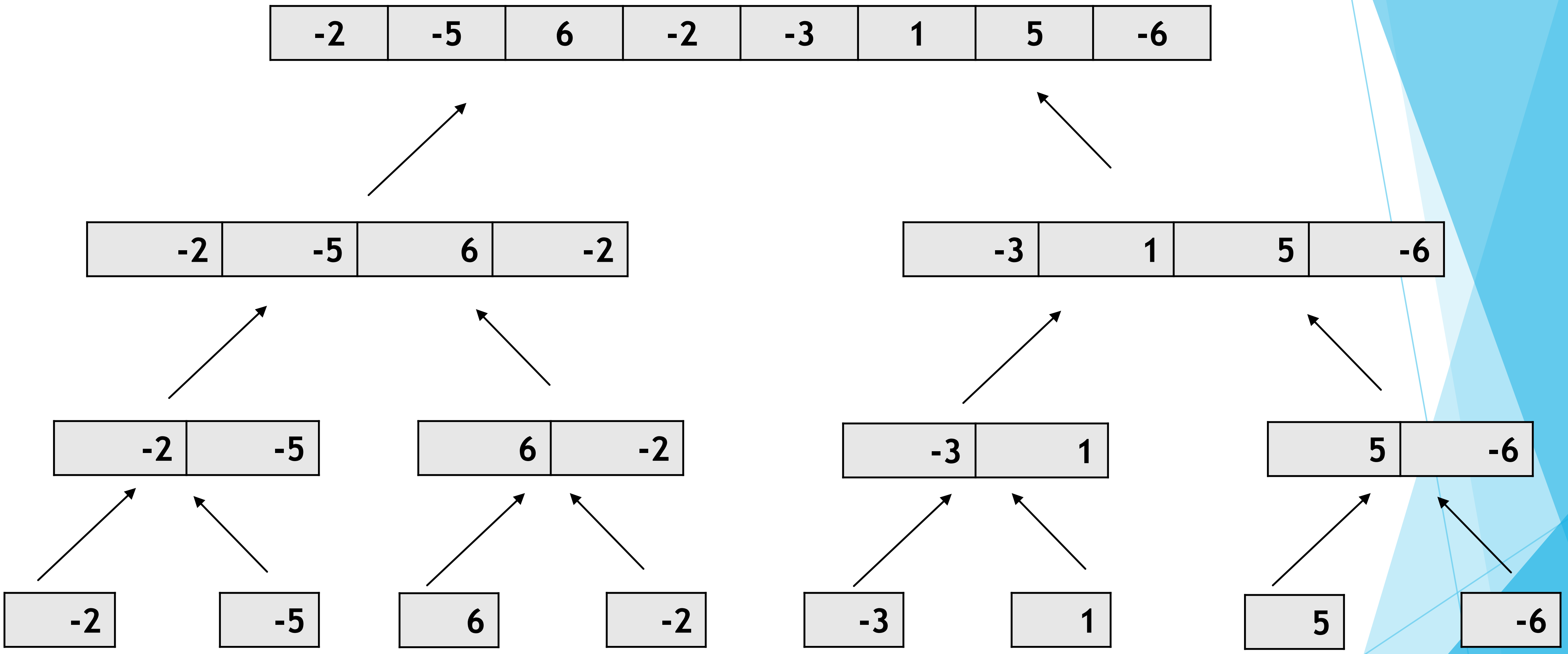
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + n^k$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^1$$

a=2
b=2   a=b^k
k=1   2=2^1

1. $a < b^k$ $T(n) \sim n^k$
2. $a = b^k$ $T(n) \sim n^k \log_b n$
3. $a > b^k$ $T(n) \sim n^{\log_b a}$

T(n) ~ n^k * log(b)a
=n*logn


-2 -5 6 -2 -3 1 5 -6
-2 -5 6 -2
-3 1 5 -6
-2 -5
6 -2
-3 1
5 -6
-2
-5
6
-2
-3
1
5
-6

ספירת הפיכות סדר במערך
Counting inversions

יהיה A מערך באורך n
הפיכת סדר בA היא זוג אינדקסים (i,j) כך ש $i<j$ אבל $A[i]>A[j]$

2	**4**	**1**	**3**	**5**

במערך שלנו יש שלושה הפיכות סדר:
<2,1>,<4,1>,<4,3>

נרצה לבנות אלגוריתם יעיל שמקבל מערך, ויחזיר את כמות הפיכות הסדר במערך.

דרך א: Brute Force approach

נעבור על כל איבר ונשווה אותו לאיברים הבאים בתור אחריו במערך.

בכל פעם שנמצא הפיכת סדר נספור אותה.

הבעיה זמן ריצה מאוד גבוהה של $O(n^2)$

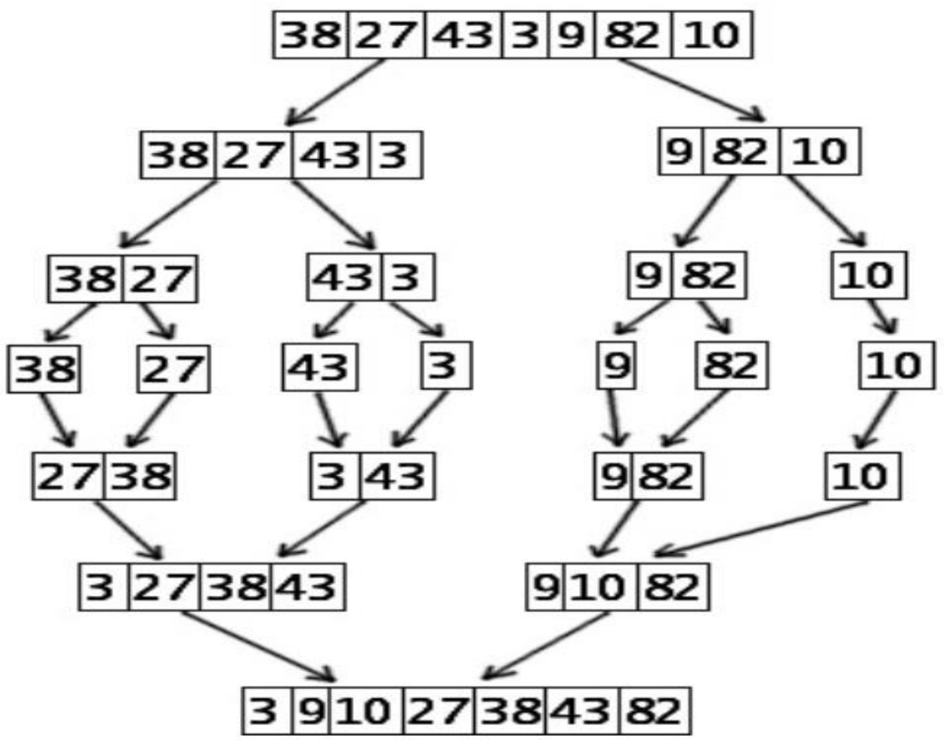
נרצה למצוא דרך לשפר את זמן הריצה.

2	4	1	3	5

בשביל לפתור את הבעיה ב O(nlogn) נרצה להשתמש בגרסא "משופצת" של MergeSort

תזכורת ל:Merge sort

```
MergeSort(arr,L,R){
If(L<R){
	//נמצא את נקודת האמצע לחלק את המערך לשתי חלקים
	M = (L+R)/2
	//נקרא באופן רקורסיבי לפונקציה עם החלק השמאלי
	 MergeSort(arr,L,M)
	//נקרא באופן רקורסיבי לפונקציה עם החלק השמאלי
	MergeSort(arr,M+1,R)
	//לאחר שמיינו את חלק שמאל וחלק ימין של המערך
	//נאחד אותם בO(n)  בעזרת פונקציית עזר Merge
	Merge(arr,L,M,R)
}
```



אז איך נפתור את הבעיה בעזרת Merge Sort?
ראשית נחלק את המערך ל2 חלקים.
אנחנו יודעים שמספר הפיכות הסדר במערך כולו שווה לסכום הבא:
מספר הפיכות הסדר בצד השמאלי של המערך+
מספר הפיכות הסדר בצד הימני של המערך +
מספר הפיכות הסדר כאשר איבר אחד בחלק השמאלי והאיבר השני בחלק הימני.

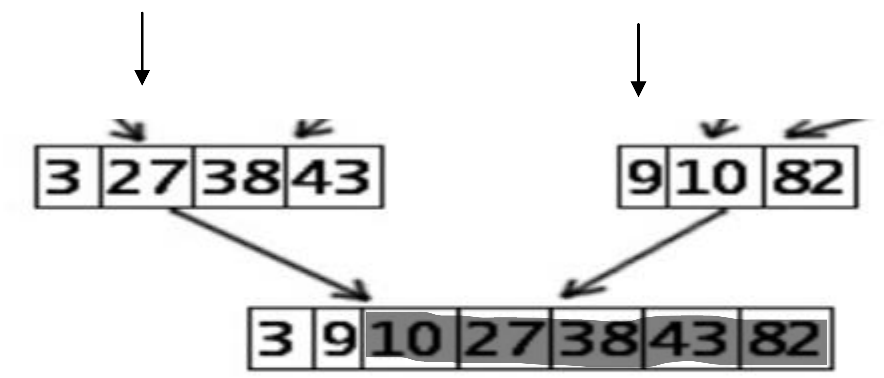
בשביל לחשב את מספר הפיכות הסדר בצד ימין נקרא לפונקציה עם החצי הימני של המערך (היא גם תמיין אותו)
בשביל לחשב את מספר הפיכות הסדר בצד שמאל נקרא לפונקציה עם החצי שמאלי של המערך (היא גם תמיין אותו)
בשביל לחשב את מספר הפיכות הסדר כאשר איבר אחד בצד שמאל והשני בימין נקרא לפונקציית העזר Merge
אשר גם תחבר בין שתי חצאי המערכים וגם תספור לנו את מספר הפיכות הסדר מהסוג הזה.

נסכום את כל שלושת המספרים שקיבלנו ונחזיר את התוצאה.

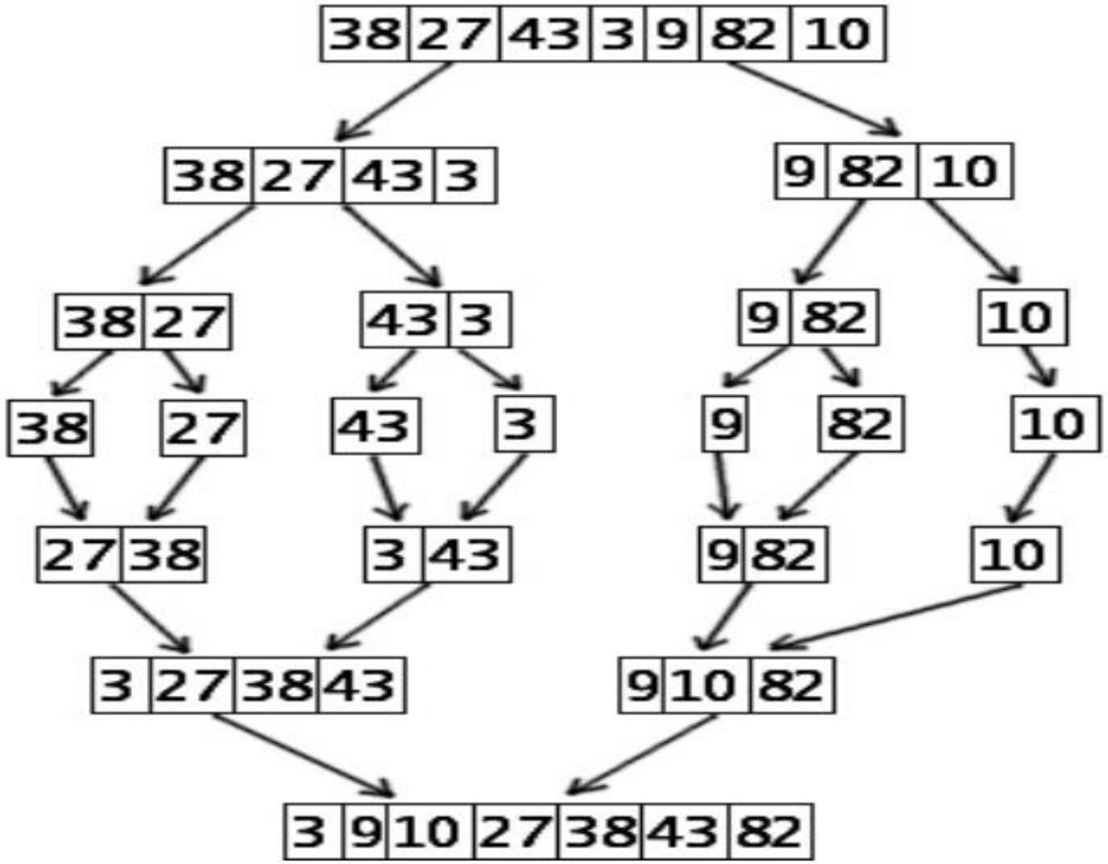
איך עובדת פונקציית ?Merge
ידוע שיש לנו שתי חצאי מערכים ממויינים.
כעת איך נמצג אותם לאחד בזמן ריצה של ?O(n)

1.נשים מצביע לאיבר הראשון של כל מערך.
2.בכל איטרציה נכניס את האיבר הקטן יותר אל תוך המערך החדש ונקדם את המצביע של אותו תת מערך ב1.
2.1 אם האיבר שהכנסנו הוא מצד ימין במערך, כלומר שאותו איבר שהכנסו יותר קטן מכל האיברים במערך השני שיותר גדולים מהאיבר שעליו אנו מצביעים
כלומר:
שכאשר מכניסים איבר מהחלק הימני, מוסיפים לספירה את הכמותi - L.lengthכאשר L.lengthהוא אורך המערך השמאלי ו iוהוא המיקום של המצביע השמאלי (בהנחה שהמערך מתחיל מאפס)

בדוגמא שלנו אנחנו מצביעים על 9 ועל 27.
9 קטן מ27.
משמע 9 גם קטן מכל האיברים שגדולים מ27 (28,43)
לכן מצאנו 3 הפיכות סדר חדשות
<27,9>
<38,9>
<43,9>



```
Int CountInversion(arr,L,R){
Count = 0;
If(L<R){
	//נמצא את נקודת האמצע לחלק את המערך לשתי חלקים//
	M = (L+R)/2
	//נמצא את כמות הפיכות הסדר בצד שמאל//
	 A_L=CountInversion(arr,L,M)
	// נמצא את כמות הפיכות הסדר בצד ימין //
	A_R=CountInversion(arr,M+1,R)
	//נמצא את כמות הפיכות הסדר כאשר איבר אחד בצד ימין, והשני בשמאל//
	//ועל הדרך נאחד גם את המערכים//
	A_M=Merge(arr,L,M,R)
	Count = A_L+A_M+A_R
}
Retrurn Count
}
```



סדר הפיכות:

<38,27>

<43,3>

<27,3>

<38,3>

<82,10>

<27,9>

<38,9>

<43,9>

<27,10>

<38,10>

<43,10>

```
Int CountInversion(arr,L,R){
Count = 0;
If(L<R){
      M = (L+R)/2
       A_L=CountInversion(arr,L,M)          T(n/2)
      A_R=CountInversion(arr,M+1,R)       T(n/2)
      A_M=Merge(arr,L,M,R)                  O(n)
      Count = A_L+A_M+A_R
}
Retrurn Count
}
```

T(n) = 2T(n/2) + O(n)

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + n^k$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n^1$$

1. $a < b^k$	$T(n) \sim n^k$
2. $a = b^k$	$T(n) \sim n^k \log_b n$
3. $a > b^k$	$T(n) \sim n^{\log_b a}$

a=2
b=2 a=b^k
k=1 2=2^1

T(n) ~ n^k * log(b)a
=n*logn

תודה רבה ! ☺