

Recursion And Runtime Stack

Recursion

רקורסיה

- רקורסיה היא תהליך בו הפונקציה (שיטה) קוראת לעצמה
- ישנן בעיות שהפיתרון הטבעי שלהן הוא רקורסיבי
- רקורסיה היא שיטה שבה הפתרון לבעיה מסתמך על פתרון בעיות קטנות ודומות לבעיה הגדולה יותר.
- בהינתן בעיה שרוצים לפתור:
- ✓ נמצא כודם תת-בעיה דומה (קטנה יותר) אשר אנו יודעים מהו הפתרון שלה
- הפתרון של תת-הבעיה יהיה תנאי העצירה של הפונקציה
- ✓ נניח שהפונקציה יודעת לפתור את הבעיה המוקטנת
- ✓ נשתמש בפתרון של הבעיה המוקטנת כדי לפתור את הבעיה הגדולה יותר
- וזו יהיה הצעד של הרקורסיה.

מימוש לולאה בעזרת שימוש ברקורסיה

- ניתן להביע כל לולאה באמצעות רקורסיה (ולהפך)
- ✓ את הבלוק של הלולאה מעברים לתוך הפונקציה
- ✓ מבצעים קריאות רקורסיביות לפונקציה
- ✓ מוסיפים תנאי עצירה

```
public void whileMethod(){  
    if (!cond) {  
        return;  
    }  
    statement;  
    whileMethod();  
}
```



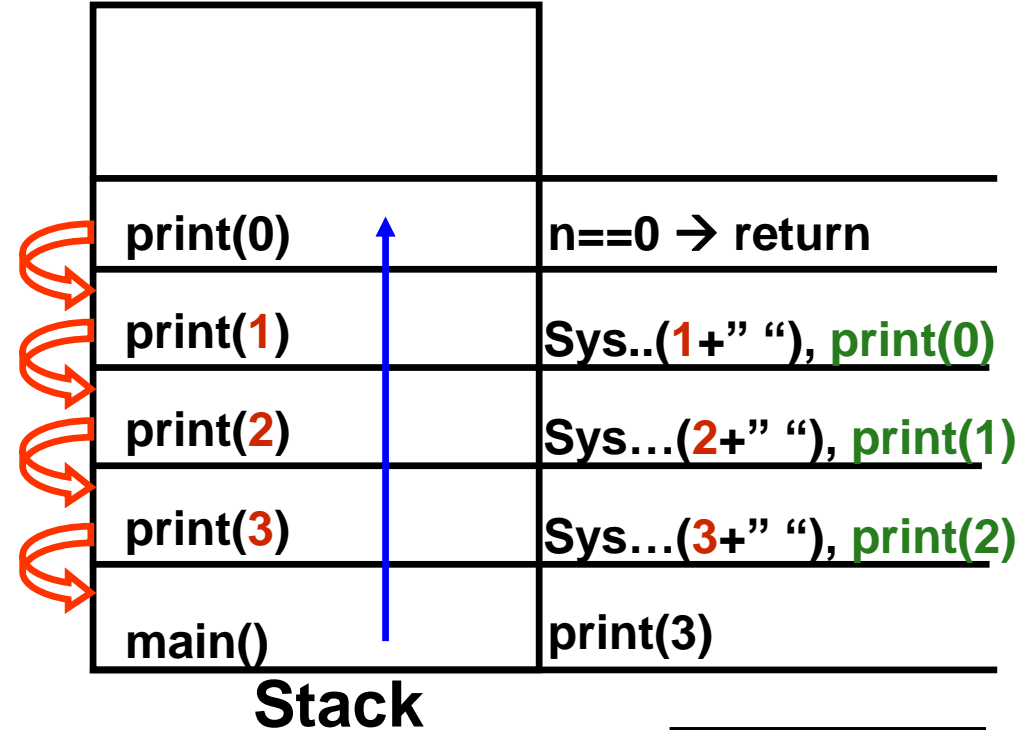
```
while (cond) {  
    statement;  
}
```

הערה: לכל קריאה רקורסיבית לפונקציה יש
עותק משלה למשתנים המקומיים - דבר שאין בלולאה

כתוב שיטה רקורסיבית שמקבלת מספר חיובי n ומדפיסה את כל המספרים מ 1 עד n בסדר יורד

```
public void print(int n){  
    if(n<=0)  
        return;  
    System.out.print(n+" ");  
    print(n-1);  
}
```

```
int n=3;  
while(n>0){  
    System.out.print(n+" ");  
    n--;  
}
```

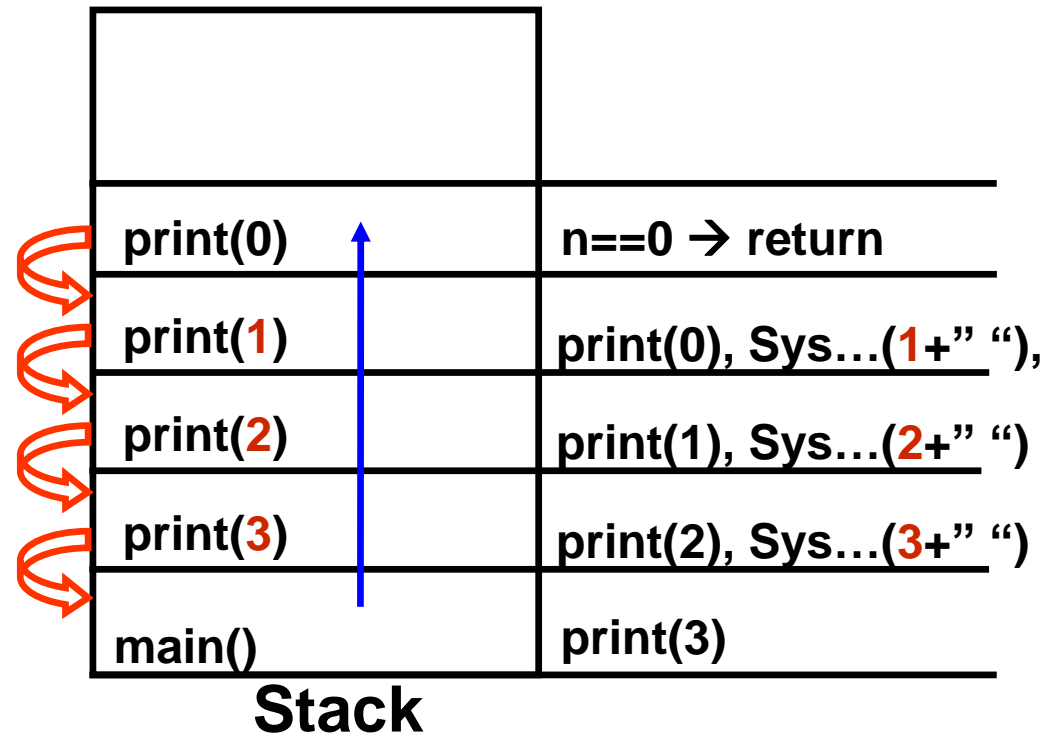


```
print(3);  
Output: 3 2 1
```

כתוב שיטה רקורסיבית שמקבלת מספר חיובי n ומדפיסה
את כל המספרים מ 1 עד n בסדר עולה

```
public void print(int n){  
    if(n<=0)  
        return;  
    print(n-1);  
    System.out.print(n+" ");  
}
```

```
print(3);  
Output: 1 2 3
```



כתוב שיטה רקורסיבית אשר מקבלת מספר חיובי n ומחזירה את סכום כל המספרים מ 0 עד n

```
public int sum (int n) {  
    if(n==0) // base case  
        return 0;  
    else // recursive case  
        return sum(n-1) + n;  
}
```

ניתן לוותר על else →

הרעיון נשתמש ב n כאינדקס לרקורסיה

איך ניתן לשפר את התוכנית הקודמת כך שתעבוד
גם על מספרים שליליים?

```
public int sum (int n) {  
    if(n==0)  
        return 0;  
    if(n<0)  
        return -sum(-n);  
  
    return sum(n-1) + n;  
}
```

כתוב שיטה רקורסיבית אשר מקבלת מערך arr ומספר size שמייצג את גודל המערך- השיטה תדפיס את איברי המערך

```
public void print(int[] arr, int size){  
    if(arr==null || size==0){  
        return;  
    }  
    System.out.println(arr[size-1]);  
    print(arr,size-1);  
}
```

tail recursive

← מהאיבר האחרון לראשון

→ מהאיבר הראשון לאחרון

```
public void print ( int[ ] arr , int size ){  
    if(arr==null || size==0){  
        return;  
    }  
    print(arr, size-1);  
    System.out.println(arr[size-1]);  
}
```

non tail recursive

כתוב שיטה רקורסיבית אשר מקבלת
מערך arr כפרמטר ומדפיסה את איברי המערך

```
public void print(int[] arr){  
    print(arr, 0);  
}  
private void print(int[] arr, int i){  
    if(arr==null || i==arr.length)  
        return;  
    System.out.print(arr[i] + " ");  
    print(arr,i+1);  
}
```

כתוב שיטה רקורסיבית אשר מקבלת מערך arr
השיטה תחזיר את סכום איברי המערך

```
public static int sum(int[ ] arr){  
    return sum(arr , 0);  
}  
  
private static int sum(int[ ] arr , int i) {  
    if (i == arr.length)  
        return 0;  
    return arr[ i ] + sum(arr , i+1);  
}
```

כתוב שיטה רקורסיבית אשר מקבלת מערך arr, מספר size שמייצג את גודל המערך ומספר num כלשהו - השיטה תחזיר את מספר הפעמים מופיע num במערך

```
public int count (int[] arr , int size , int num) {  
    if(size==0)  
        return 0;  
    if( arr[size - 1] != num)  
        return count (arr , size-1, num);  
    return 1 + count (arr, size-1, num);  
}
```

כתוב שיטה רקורסיבית אשר מקבלת שני מספרים a , b
חיוביים ושלמים ומחזירה את הסכום שלהם $a+b$

```
public int sum (int a , int b) {  
    if(b==0) // base case  
        return a;  
    return sum(a , b-1)+1; //recursive case  
}
```

הרעיון : נשתמש באחד המספרים כאינדקס למשל b
שאלה : איך התוצאה תשתנה אם b מספר שלילי?
איך ניתן לתקן זאת כך שהשיטה תעבוד גם על מספרים שליליים?
תשובה בעמוד הבא

כתוב שיטה רקורסיבית אשר מקבלת שני מספרים a , b
חיוביים/שליליים ושלמים ומחזירה את הסכום שלהם $a+b$

```
public int sum (int a, int b) {  
    if(b==0)  
        return a;  
    if(b<0)  
        return -sum(-a,-b);  
    return sum(a,b-1)+1;  
}
```

$$a + b = - (-a - b)$$

כתוב שיטה רקורסיבית אשר מקבלת מספר חיובי n ומחזירה

את האיבר ה- n -י בסדרה :

$$\begin{cases} a_1 = 3 \\ a_n = a_{n-1} + 2 \end{cases}$$

```
public int an(int n) {  
    if (n == 1)  
        return 3;  
    return an(n-1) + 2;  
}
```

כתוב שיטה רקורסיבית אשר מקבלת שני מספרים a , b
שלמים וחיוביים ומחזירה את השארית $a \% b$

```
public int mod (int a , int b) {  
    if(b==0)  
        return 0;  
    if(a<b)  
        return a;  
    return mod (a-b , b);  
}
```

איך ניתן לשפר את התוכנית הקודמת כדי שזו
תעבוד גם אם המספרים a , b הם שלמים חיוביים/שלילים

הערה: התוצאה תהיה
שלילית אך ורק אם a
הוא שלילי

```
public static int mod(int a, int b) {  
    if(b==0)  
        return 0;  
    if(Math.abs(a)<Math.abs(b))  
        return a;  
    if(a<0)  
        return -mod(Math.abs(a) , b);  
    return mod(a - Math.abs(b) , Math.abs(b));  
}
```

$5 \% 3 = 2$
 $5 \% -3 = 2$
 $-5 \% 3 = -2$
 $-5 \% -3 = -2$

בהינתן שני מספרים שלמים וחיוביים a , b
שכנע את עצמך שהשיטה `mult` מחזירה $a*b$


```
public int mult(int a, int b) {  
    if (b == 0)  
        return 0;  
    if (b % 2 == 0)  
        return mult(a + a, b / 2);  
    return mult(a + a, b / 2) + a;  
}
```

`mult(2,5)→10`

`mult(3,6)→18`

נתונים שני מספרים חיוביים a , b
כתוב שיטה רקורסיבית אשר מחשבת את מכפלתם

הפונקציה תעבוד
גם אם a הוא שלילי



```
public int mult(int a , int b) {  
    if(b==0)  
        return 0;  
    return mult(a , b-1) + a;  
}
```

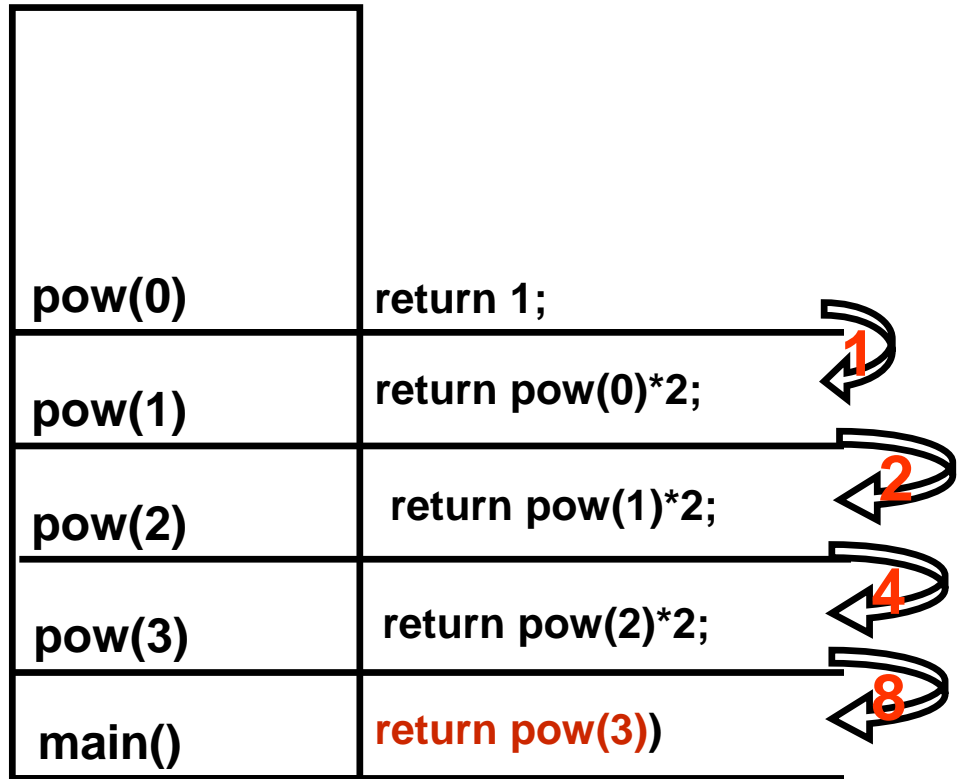
מה צריך לשנות בתוכנית הקודמת
כדי שזו תעבוד עבור כל קלט – גם מספרים שליליים

```
public int mult(int a , int b){  
    if(b==0)  
        return 0;  
    if(b<0)  
        return - mult(a , -b);  
    return a + mult(a , b-1);  
}
```

הפתרון הקודם כלל בתוכו את האפשרות ש a שלילי.
המשתנה b משמש אותנו כאינדקס ולכן ערכו חייב להיות
חיובי. מצד שני בפועל b יכול להיות שלילי. לכן נהפוך את
הסימן של b לחיובי (במידה והוא שלילי) ונבצע קריאה
רקורסיבית לפונקציה (כאשר b חיובי) בסוף החישוב נהפוך
את הסימן של הערך המוחזר

כתוב שיטה רקורסיבית שמקבלת מספר חיובי n ומחשבת את 2^n

```
public long pow (int n) {  
    if (n == 0)  
        return 1;  
    return pow(n-1)*2;  
}
```



`main(): System.out.println(pow(3))`

כתוב שיטה רקורסיבית שמקבלת
שני מספרים a , b חיוביים ושלימים ומחשבת a^b

```
public long pow(int a , int b){  
    if (b == 0)  
        return 1;  
    return pow(a , b-1) * a;  
}
```

$$a^b = a \cdot a^{b-1}$$

כתוב שיטה רקורסיבית שמקבלת שני מספרים
 a^b a, b חיוביים/שליליים ושלימים ומחשבת
שימו לב התוצאה יכולה להיות שבר

```
public double pow(int a , int b){  
    if (b == 0)  
        return 1;  
    if(b<0)  
        return 1.0/pow(a , -b);  
    return pow(a , b-1) * a;  
}
```

$$a^b = a \cdot a^{b-1}, b > 0$$

$$a^b = \frac{1}{a^{-b}}, b < 0$$

כתוב שיטה רקורסיבית שמקבלת מספר שלם וחיובי n - ומחשבת $n!$

```
public int factorial(int n, int fact)
{
    if (n == 0)
        return fact;

    return factorial(n-1, n*fact);
}

int factorial(int n)
{
    return factorial(n, 1);
}
```

tail recursive

factorial(5, 1)
factorial(4, 5)
factorial(3, 20)
factorial(2, 60)
factorial(1, 120)
factorial(0, 120)
120

$n! = n * (n-1)!$

```
public int factorial(int n)
{
    if (n == 0)
        return 1;

    return n * factorial(n-1);
}
```

non tail recursive

factorial(5)
5 * factorial(4)
5 * (4 * factorial(3))
5 * (4 * (3 * factorial(2)))
5 * (4 * (3 * (2 * factorial(1))))
5 * (4 * (3 * (2 * (1 * factorial(0)))))
5 * (4 * (3 * (2 * (1 * 1))))
120

כתוב שיטה רקורסיבית אשר מקבלת מערך arr
ומספר n שמייצג את גודל המערך - השיטה
תחזיר את האיבר המקסימאלי שבמערך

```
public int max(int[ ] arr , int size) {  
    int tmp;  
    if(size==1) {  
        return arr[0];  
    }  
    tmp= max(arr,size-1);  
    if(arr[size-1]>tmp)  
        tmp=arr[size-1];  
    return tmp;  
}
```

שימוש במשתנה עזר

```
public int max( int[ ] a , int n ) {  
    if(n==1)  
        return a[0];  
    return max( a[n-1] , max(a,n-1) );  
}  
  
private int max( int a , int b ) {  
    return a>b ? a:b;  
}
```

שימוש בשיטת עזר

כתוב שיטה רקורסיבית אשר מקבלת מערך arr
ומחזירה את האיבר המקסימאלי שבמערך

```
public int max ( int[ ] a ) {  
    return max( a , 0 );  
}  
  
private int max( int[ ] a , int i ) {  
    if( i==a.length-1 )  
        return a[ i ];  
    return max( a[ i ] , max( a , i+1 ) );  
}  
  
private int max( int a , int b ) {  
    return a>b ? a:b;  
}
```

כתוב שיטה רקורסיבית אשר מקבלת מערך arr ומשתנים start
 end- ומחזירה את האיבר המקסימאלי בין start ו- end שבמערך
 $\text{max}(\{1,2,5,1,3,4,1\}, 1, 4) \rightarrow 5$

↑ start=1 ↑ end=4

```
public int max(int[] arr ,int start, int end){
    int max;
    if ( start==end )
        return arr[start];
    max=max( arr , start+1 , end );
    if ( max < arr[start] )
        max=arr[start];
    return max;
}
```

max(arr,4,4)	if(4==4) return arr[4]; 3
max(arr,3,4)	max=max(arr,4,4) ; if(max<arr[3]); x return max; 3
max(arr,2,4)	max=max(arr,3,4) ; if(max<arr[2]) max=arr[2]; return max; 5
max(arr,1,4)	max=max(arr,2,4); if(max<arr[1]) ; x return max; 5
main()	max({1,2,5,1,3,4,1},1,4) 5

System.out.println(max({1,2,5,1,3,4,1},1,4)) → 5

כתוב שיטה סטטית רקורסיבית אשר מקבלת מערך arr
ומחזירה את המקסימום והמינימום.

```
public static int[ ] minMax(int[ ] a){  
    int[ ] result={a[0],a[0]};  
    return minMax(a , 0 , result);  
}
```

```
private static int[ ] minMax(int[ ] a , int i, int[ ] result) {  
    if (i==a.length)  
        return result;  
    if ( a[i]<result[0] )  
        result[0]=a[i];  
    if ( a[i]>result[1] )  
        result[1]=a[i];  
    return minMax(a , i+1 , result);  
}
```

* כתוב שיטה רקורסיבית אשר מקבלת מערך דו מימדי arr ומחזירה את מס השורה בעלת הסכום המקסימלי.

```
public static int max (int[ ][ ] arr ) {  
    if(arr==null)  
        return -1;  
    return max ( arr , 1 , 0 , sum (arr[0] , 0) );  
}  
private static int max (int[ ][ ] arr , int i , int row , int maxSum ) {  
    if ( i == arr.length )  
        return row;  
    int s = sum ( arr[i] , 0 );  
    if ( s> maxSum ){  
        maxSum=s;  
        row=i;  
    }  
    return max (arr, i+1, row , maxSum);  
}  
private static int sum (int[ ] arr, int i){  
    if ( i == arr.length )  
        return 0;  
    return arr[i] + sum (arr , i+1);  
}
```

כתוב שיטה רקורסיבית שמקבלת שני מספרים a,b שלמים וחיוביים ומחזירה $\text{GCD}(a,b)$

$$\text{gcd}(a, b) = \begin{cases} a & \text{if } b = 0 \\ \text{gcd}(b, a \% b) & \text{otherwise} \end{cases}$$

$\text{gcd}(18,12)=6$

$\text{gcd}(616,165)=11$

$\text{Gcd}(1071,1029)=21$

```
public int gcd(int a, int b) {  
    if (b == 0)  
        return a;  
    return gcd(b, a%b);  
}
```

Euclid's Algorithm

$a \% b$	a/b	b	a
121	3	165	616
44	1	121	165
33	2	44	121
11	1	33	44
0	3	11	33
-	-	0	11

כתוב שיטה רקורסיבית שמקבלת מספר שלם וחיובי ומחזירה את הצורה הבינארית שלו

```
public String binary (int a) {  
    if (a<2)  
        return "" + a;  
    return binary(a/2) ⊕ (a%2);  
}
```

חיבור מחרוזות

שארית	תוצאת חילוק	
0	2	4
0	1	2
1	0	1

כתוב שיטה רקורסיבית שמקבלת מספר n שלם חיובי/שלילי ומחזירה מספר חדש המורכב מספרות המספר בסדר הפוך

גרסה 1

```
public static String revDig(int a) {  
    if(a>=0&&a<10)  
        return ""+a;  
    if(a<0)  
        return "-" + revDig(-a);  
    return a%10 + "" + revDig(a/10);  
}
```

```
public static int revDeg (int num) {          גרסה 2
    return revDig(num,0);
}
public static int revDig (int num , int revNum){
    if(num == 0) {
        return revNum;
    }
    if(num<0)
        return -revDig (-num , 0);

    revNum = revNum * 10 + (num % 10);
    num = num/10;

    return revDig(num, revNum);
}
```


כתוב שיטה רקורסיבית שמקבלת מחרוזת s ומספר n ומחזירה מחרוזת
חדשה המורכבת מ n פעמים המחרוזת המקורית

"ab", 3 → "ababab"

"ab", 0 → ""

```
public String repeat (String s, int n){  
    if(n==0)  
        return "";  
    return repeat(s,n-1)+s;  
}
```

כתוב שיטה רקורסיבית שמקבלת
מחרוזת ומחזירה true אם s היא פלינדרום
“aba”→true, “abc”→false, “”→false, “abba”→true

```
public static boolean palindrome(String s){  
    if(s== null || s.length()==0)  
        return false;  
    if(s.length()==1)  
        return true;  
    if(s.length()==2 &&(s.charAt(0) == s.charAt(1)) )  
        return true;  
    if(s.charAt(0) != s.charAt(s.length()-1) )  
        return false;  
    return palindrome(s.substring(1,s.length()-1));  
}
```

כתוב שיטה רקורסיבית שמקבלת
מחרוזת str ותו c ומחזירה את המיקום של c ב-str

```
public static int find(String str, char c) {  
    if(str.length() == 0)  
        return -1;  
    if(str.charAt(0) == c)  
        return 0;  
    return find(str.substring(1), c) + 1;  
}
```

כתוב שיטה רקורסיבית המקבלת מחרוזת
כפרמטר ומחליפה את כל הרווחים במחרוזת ב - #

```
public String replace(String line) {  
    if(line.length()==0) // line.equals("")  
        return "";  
    if(line.charAt(0)==' '  
        return "#" + replace(line.substring(1));  
    return line.charAt(0) + replace(line.substring(1));  
}
```

כתוב שיטה רקורסיבית המקבלת מחרוזת כפרמטר ומחזירה את אותה מחרוזת בסדר הפוך

```
public String reverse (String str) {  
    if(str == null || str.equals(""))  
        return str;  
    return reverse(str.substring(1)) + str.substring(0,1);  
}
```

גרסה 1 : שימוש ב `substring(..)`

```
public String reverse(String str) {  
    if(str == null || str.equals(""))  
        return str;  
    return reverse(str.substring(1)) + str.charAt(0);  
}
```

גרסה 2 : שימוש ב `charAt(..)`

כתוב שיטה רקורסיבית המקבלת מחרוזת כפרמטר
ומחזירה את אותה מחרוזת ללא כפילויות רציפות
"abbcccddefca" → "abcdefca"

```
public String removeDup(String str) {  
    if(str == null || str.length() < 2)  
        return str;  
    if( str.charAt(0) == str.charAt(1) )  
        return removeDup(str.substring(1));  
    else  
        return str.charAt(0) + removeDup(str.substring(1));  
}
```

כתוב שיטה רקורסיבית המקבלת
שתי מחרוזת s1,s2 ממוינות בסדר אלפביתי
ומחזירה את המיזוג שלהן שהוא גם ממוין בסדר אלפביתי

```
public static String merge(String s1 , String s2) {  
    if(s1==null || s1.length()==0 )  
        return s2;  
    if(s2==null || s2.length()==0)  
        return s1;  
    if(s1.charAt(0) < s2.charAt(0))  
        return s1.charAt(0) + merge(s1.substring(1),s2);  
    return s2.charAt(0) + merge(s1,s2.substring(1));  
}
```

"abcdegh"+"acfgggi"→"aabccdefgggghi"

כתוב שיטה רקורסיבית (לפי הפרד ומשול) אשר מקבלת מערך arr ומחזירה את האיבר המקסימאלי שבמערך $\{1,2,1,1,3,4,1\} \rightarrow 4$

```
public static int max(int[ ] arr) {  
    return max(arr,0,arr.length-1);  
}  
private static int max(int[ ] arr, int left , int right) {  
    if (left == right)  
        return arr[left];  
    int middle = (left + right) / 2;  
    int max1 = max (arr, left, middle);  
    int max2 = max (arr, middle+1, right);  
    return max1>max2 ? max1:max2;  
}
```

הערה: תרגיל זה נועד
רק כדי להסביר את
שיטת הפרד ומשול

ע"י שימוש
במשתנה עזר

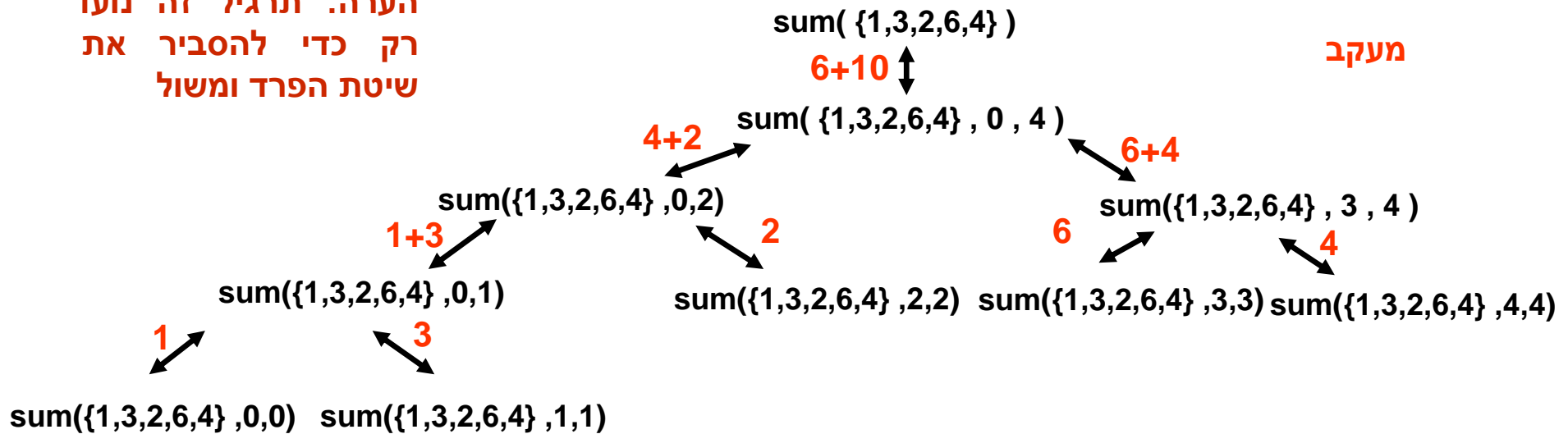
ע"י שימוש ב
overloading
וללא משתנה עזר

```
public static int max(int[ ] arr) {  
    return max(arr , 0 , arr.length-1);  
}  
private static int max(int[ ] arr, int left ,int right) {  
    if (left == right)  
        return arr[left];  
    return max ( max (arr, left, ((left + right)/2)) , max (arr, ((left + right)/2)+1, right) );  
}  
private static int max(int a, int b){  
    return a>b ? a:b;  
}
```


כתוב שיטה רקורסיבית **(לפי הפרד ומשול)** אשר מקבלת
מערך arr ומחזירה את סכום איברי המערך

```
public static int sum(int[ ] a){
    return sum(a,0,a.length-1);
}
private static int sum(int[ ] a,int left, int right){
    if(left == right)
        return a[right];
    return sum( a , left , ((left+right)/2) ) + sum( a , ((left+right)/2)+1, right);
}
```

הערה: תרגיל זה נועד
רק כדי להסביר את
שיטת הפרד ומשול

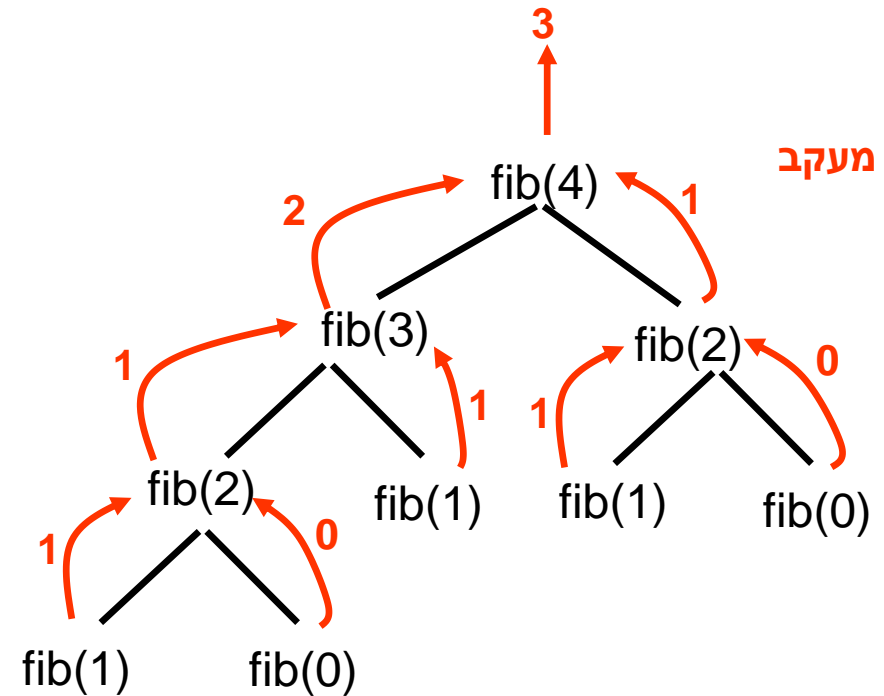


* כתוב שיטה רקורסיבית אשר מקבל מספר חיובי n ומחזירה את ערך Fibonacci(n)

```
public int fib(int n) {  
    if (n == 0)  
        return 0;  
    if (n == 1)  
        return 1;  
    return fib(n - 1) + fib(n - 2);  
}
```

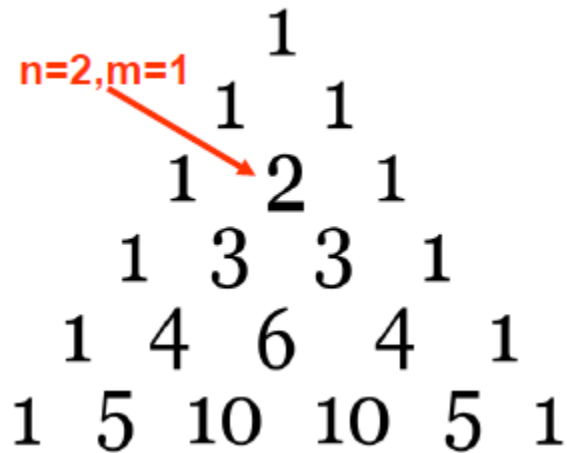
fib(n) = 0 if n is 0
= 1 if n is 1
= fib(n-1) + fib(n-2) otherwise

Fib(0) = 0
Fib(1) = 1
Fib(2) = Fib(1) + Fib(0) = 1
Fib(3) = Fib(2) + Fib(1) = 2
Fib(4) = Fib(3) + Fib(2) = 3
Fib(5) = Fib(4) + Fib(3) = 5
Fib(6) = Fib(5) + Fib(4) = 8



tree / non-linearly recursive

****משולש פסקל** הוא סידור של מספרים בצורת משולש, הנבנה באופן הבא: הקודקוד העליון של משולש זה מכיל את המספר 1, וכל מספר במשולש מהווה את סכום שני המספרים שנמצאים מעליו (המספרים שנמצאים על שוקי המשולש הם כולם 1). למשולש פסקל יש חשיבות רבה **בקומבינטוריקה** מכיוון שהמספר ה-m בשורה ה-n, נותן את התשובה לשאלה "בכמה דרכים שונות אפשר לבחור m עצמים מתוך n עצמים, ללא חזרות וללא חשיבות לסדר?".



בהינתן n ו-m
כתוב שיטה רקורסיבית אשר
מחשבת את המספר לפי
שיטת פסקל

$$\binom{n}{0} = 1$$

$$\binom{n}{n} = 1 \quad 0 < m < n$$

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$$

http://en.wikipedia.org/wiki/Pascal's_triangle

```
public int pascal (int n , int m) {  
    if (m == 0)  
        return 1;  
    if (m == n)  
        return 1;  
    return ( pascal(n-1, m-1) + pascal(n-1, m) );  
}
```

***כתוב שיטה רקורסיבית המקבלת מערך
של תווים a [] char ומדפיסה את כל תתי הקבוצות
האפשריות (ללא חזרות) המורכבות משילוב של התווים במערך a

{a}
{b}
{c}
{bc}
{ab}
{ac}
↓
{abc}

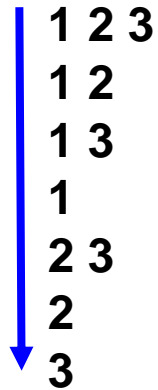
```
public static void combinations ( char[ ] array ) {
    combinations (array , 0, "{" );
}
private static void combinations(char[ ] array, int i,String prefix) {
    if ( i == array.length )
        return;
    System.out.println ( prefix + array [ i ] + "}" );
    combinations (array , i + 1 , prefix );
    combinations (array , i + 1 , prefix + array [ i ]);
}
```

a={'a','b','c'} → {a} {b} {c} {bc} {ab} {ac} {abc}

שאלה: איך סדר
ההדפסה היה
משתנה אם נחליף
את סדר הקריאות
הרקורסיבית ←

{a}
{ab}
{abc}
{ac}
{b}
{bc}
↓
{c}

***כתוב שיטה רקורסיבית המקבלת רשימה
של מספרים a [] int ומדפיסה את כל תתי הקבוצות
האפשריות (ללא חזרות) המורכבות משילוב של המספרים ברשימה



1 2 3
1 2
1 3
1
2 3
2
3

```
public void allSubSets( int[ ] arr ){  
    boolean[ ] history=new boolean[arr.length];  
    allSubSets(arr,0,history);  
}  
private void allSubSets( int[ ] arr , int i , boolean[ ] history) {  
    if ( i==arr.length ) {  
        print(arr,history,0);  
    } else {  
        history[i]=true;  
        allSubSets(arr,i+1,history);  
        history[i]=false;  
        allSubSets(arr,i+1,history);  
    }  
}  
private void print( int[ ] arr , boolean[ ] history , int i) {  
    if(i==arr.length){  
        System.out.println();  
    } else {  
        if(history[i]==true)  
            System.out.print(arr[i]+" ");  
        print(arr,history,i+1);  
    }  
}
```

Recursion And Runtime Stack
Jazmawi Shadi

*** כתוב שיטה ריקורסיבית isTrans אשר מקבלת שתי מחרוזות תווים s1 ו-s2. השיטה תחזיר true אם במחרוזת s2 מופיעים כל התווים של s1, לפי הסדר, אבל לפעמים תו מסוים מ s2 יכול להופיע ברצף מספר לא ידוע של פעמים יותר מאשר במחרוזת s1.

```
public boolean isTrans (String s1, String s2) {  
    if(s1.length()==0 && s2.length()==0)  
        return true;  
  
    if(s1.length()==0 || s2.length()==0)  
        return false;  
  
    if(s1.charAt(0) != s2.charAt(0))  
        return false;  
  
    return isTrans (s1.substring(1), s2.substring(1)) ||  
           isTrans(s1, s2.substring(1));  
}
```

s1= "abacd" , s2=("abacd" , "aaaabacd" , "abacd", "aabbccdd" , "abbbccd") → true
s1= "abacd" , s2=("a" , "abcd" , "aacbbdd" , "abacd") → false

***** תהי נתונה קבוצה S של מספרים טבעיים (שלמים גדולים מ-0), ויהי n מספר טבעי כלשהו. נאמר ש- n הוא סכום מתוך S אם ניתן לבטא את S כסכום של מספרים (עם או בלי חזרות) מתוך S .
דוגמאות:**

תהי $S = \{4, 5\}$, אזי למשל 13 הוא סכום מתוך S שכן $13 = 5 + 4 + 4$, אבל 6 אינו סכום מתוך S , שכן לא ניתן לבטא את 6 כסכום של מספרים מתוך S .
תהי $S = \{4, 9, 3\}$, אזי למשל 15 הוא סכום מתוך S שכן $15 = 3 + 3 + 9$, אבל 5 אינו סכום מתוך S , שכן לא ניתן לבטא את 5 כסכום של מספרים מתוך S .

שימו לב: 0 הוא סכום מתוך כל קבוצה S , שכן ניתן לבטא את 0 כסכום של קבוצה ריקה של מספרים. כמו כן, לכל מספר טבעי n השייך לקבוצה S , n הוא סכום מתוך S , כן ניתן לבטא את n כסכום של עצמו בלבד.

כתבו שיטה סטטית רקורסיבית בוליאנית המקבלת כפרמטרים מערך s מלא במספרים טבעיים שונים זה מזה, המהווים את איברי הקבוצה, ומספר טבעי n , ומחזירה true אם n הוא סכום מתוך s . אחרת, השיטה תחזיר false.

המערך לא ממזין.

חתימת השיטה היא:

```
public static boolean isSumOf(int [ ] s, int n)
```

אתם יכולים להניח כי הפרמטרים תקינים. המערך s אינו null ואינו ריק, וכל האיברים שבו הם טבעיים ושונים זה מזה, והמספר n הוא טבעי.

השיטה צריכה להיות רקורסיבית ללא שימוש בלולאות כלל. ניתן להשתמש בשיטות עזר, אך גם הן לא יכולות להכיל לולאות.

אפשר להשתמש בהעמסת-יתר (overloading).


```
public static boolean isSumOf(int [ ] s , int n) {  
    return isSumOf(s, n, 0,0);  
}  
  
private static boolean isSumOf( int [ ] s , int n , int i , int sum ) {  
    if(sum==n)  
        return true;  
  
    if ( i >= s.length || sum>n )  
        return false;  
  
    return isSumOf( s , n , i , sum + s[i] ) || isSumOf( s , n , i + 1 , sum );  
}
```

```
public static boolean isSumOf(int [ ] s , int n) {  
    return isSumOf(s, n, 0);  
}  
  
private static boolean isSumOf( int [ ] s , int n , int i ) {  
    if(n==0)  
        return true;  
  
    if ( i >= s.length || n<0 )  
        return false;  
  
    return  isSumOf( s , n - s[i] , i ) || isSumOf( s , n , i + 1 );  
}
```

```

public static void isSumOf(int [ ] s , int n) {
    int [ ] history = new int[s.length];
    isSumOf( s, n, 0 , 0 , history );
}
private static void isSumOf( int [ ] s , int n , int i , int sum, int[ ] history) {
    if(sum==n) {
        print( s , history , 0 , 1 );
        return;
    }

    if ( i >= s.length || sum > n )
        return;

    history[i]++;
    isSumOf( s , n , i , sum + s[i] , history ) ;
    history[i]--;
    isSumOf( s , n , i + 1 , sum , history);
}
public static void print( int [ ] arr , int [ ] history , int l , int t) {
    if( i==arr.length ) {
        System.out.println();
    } else {
        if( t <= history[i] ) {
            System.out.print(arr[i]+" ");
            print( arr , history , i , t+1);
        }
        else
            print( arr , history , i+1, 1);
    }
}
}

```

בהנחה והינו רוצים להדפיס את רשימת המספרים נשתמש במערך עזר , שמשמש לשמירת רשימת המספרים שמהווה סכום מ S .

המערך history שומר את מספר ההופעות של כל אחד מהמספרים שמהווים סכום מהמערך המקורי S.

```

public static boolean isSumOf(int [ ] s , int n) {
    int [ ] history = new int[s.length];
    return isSumOf( s, n, 0 , 0 , history );
}
private static boolean isSumOf( int [ ] s , int n , int i , int sum, int[ ] history) {
    if(sum==n) {
        print( s , history , 0 , 1 );
        return true;
    }

    if ( i >= s.length || sum > n )
        return false;

    history[i]++;
    boolean b1 = isSumOf( s , n , i , sum + s[i] , history ) ;
    history[i]--;
    boolean b2 = isSumOf( s , n , i + 1 , sum , history);
    return b1 || b2;
}
private static void print( int [ ] arr , int [ ] history , int i , int t) {
    if( i==arr.length ) {
        System.out.println();
    } else {
        if( t <= history[i] ) {
            System.out.print(arr[i]+" ");
            print( arr , history , i , t+1);
        }
        else
            print( arr , history , i+1, 1);
    }
}
}

```

***שאלה 4 - להגשה (25%)

נתון מערך דו-ממדי **ריבועי** המכיל בתוכו מספרים שלמים.

נגדיר **מסלול** (path) במערך המתחיל בתא $[x1][y1]$ ומסתיים בתא $[x2][y2]$ כך: סדרה של תאים במערך באופן שמתקיים כי:

א- התא הראשון בסדרה הוא $[x1][y1]$

ב- התא האחרון בסדרה הוא $[x2][y2]$

ג- המעבר מתא הוא רק לשכניו כאשר שכניו של תא הם התאים הצמודים אליו שמעליו, מתחתיו, מימינו ומשמאלו. במקרה

ורק חלק מהשכנים קיימים (כאשר התא $[i][j]$ נמצא בשולי המערך), יש להתחשב רק בשכנים הקיימים.

ד- כל תא במערך מופיע לכל היותר פעם יחידה במסלול.

לדוגמא, במערך בן חמש שורות וחמש עמודות קיימים המסלולים הבאים בין התא $[0][1]$ לתא $[2][4]$:

$[0][1] - [0][2] - [0][3] - [1][3] - [1][4] - [2][4]$

$[0][1] - [1][1] - [2][1] - [2][0] - [3][0] - [3][1] - [3][2] - [3][3] - [3][4] - [2][4]$

וקיימים, כמובן, עוד מסלולים רבים.

כתבו שיטה סטטית **רקורסיבית** המקבלת כפרמטר מערך דו-ממדי mat מלא במספרים שלמים חיוביים, וזוג תאים במערך המצויים מעל לאלכסון הראשי (כלומר, האלכסון $mat[i][i]$). השיטה צריכה להחזיר את מספר המסלולים שאינם חוצים את האלכסון הראשי (אך הם עשויים להכיל תאים המצויים באלכסון זה) הקיימים בין שני התאים.

בדוגמא לעיל, המסלול הראשון אינו חוצה את האלכסון הראשי, אך המסלול השני כן חוצה, ולכן הוא לא ייספר.

חתימת השיטה:

```
public static int numPaths (int[ ][ ] mat, int x1, int y1, int x2, int y2)
```

השיטה צריכה להיות **רקורסיבית** ללא שימוש בלולאות כלל. כך גם כל שיטות העזר שתכתבו (אם תכתבו) לא יכולות להכיל לולאות.

אפשר להשתמש בהעמסת-יתר (overloading).

אסור להשתמש במשתנים סטטיים (גלובליים)!

אסור להשתמש במערך עזר.

ערכי המטריצה לאחר הרצת השיטה `numPaths` צריכים להיות תואמים לערכי המטריצה לפני הרצת השיטה.

```

public static int numPaths (int[][] mat, int x1, int y1, int x2, int y2) {
    if (!borderOk(mat,x1,y1) || !borderOk(mat,x2,y2))
        return 0;

    if(x1==x2 && y1==y2)
        return 1;

    int x=mat[x1][y1];
    mat[x1][y1]=-1;
    int c=0;

    c=numPaths(mat,x1,y1+1,x2,y2) +
      numPaths(mat,x1,y1-1,x2,y2) +
      numPaths(mat,x1+1,y1,x2,y2) +
      numPaths(mat,x1-1,y1,x2,y2);

    mat[x1][y1]=x;
    return c;
}

private static boolean borderOk(int[][] mat , int x,int y) {
    if(mat.length!=mat[0].length || x<0|| x>=mat.length || y<0|| y>=mat.length || y<x|| mat[x][y]<=0)
        return false;

    return true;
}

```

```

public static void main()
{
    char a[][] =
    {
        {'h','e','l','l','o'},
        {'a','b','h','e','l'},
        {'f','o','e','o','l'},
        {'e','l','l','l','o'},
        {'h','l','o','s','j'}
    };

    System.out.println(count(a,1,2,"hello"));
}

```

***write a recursive function 'count' that receives a matrix Of 'char a[M][N]' ,int i,int j and a string 'char *s' and returns the number of times 's' appears in 'a' starting from place (i,j).

count(char a[M][N],int i, int j, String s)

Note: if s is null or empty string then count returns 0.

Output:

5

```
public static int count(char a[][],int i, int j, String s)
{
    if(a==null || s==null || i<0 || i>=a.length || j<0 || j>=a[0].length || a[i][j]=='#' || s.length()==0 || a[i][j] != s.charAt(0))
        return 0;

    if(s.length()==1 && a[i][j] == s.charAt(0))
        return 1;

    char x = a[i][j];
    a[i][j] = '#';

    int c =
        count(a,i-1, j,s.substring(1))+
        count(a,i+1, j,s.substring(1))+
        count(a,i, j+1,s.substring(1))+
        count(a,i, j-1,s.substring(1));

    a[i][j] = x;
    return c;
}
```


summary

While solving a backtracking problem a one should identify:

- ✓ What are the "choices" in the problem.
- ✓ What is the base case.
- ✓ How to make a choice.
- ✓ Should we create additional variables to remember a previous choice. if yes then is their a need to modify the values of existing variables. And How do we make the next/rest of the choices.
- ✓ Should we remove the made choice from the list of choices once we are done with exploring all the choices and if yes then how to remove a done choice.



END