Recursion:

* when a function calls itself is known as Recursive function.

```
fun (int x)
  {
        ----
        ----
        ----
        fun (x-1);  → calling the same func.
  }
```

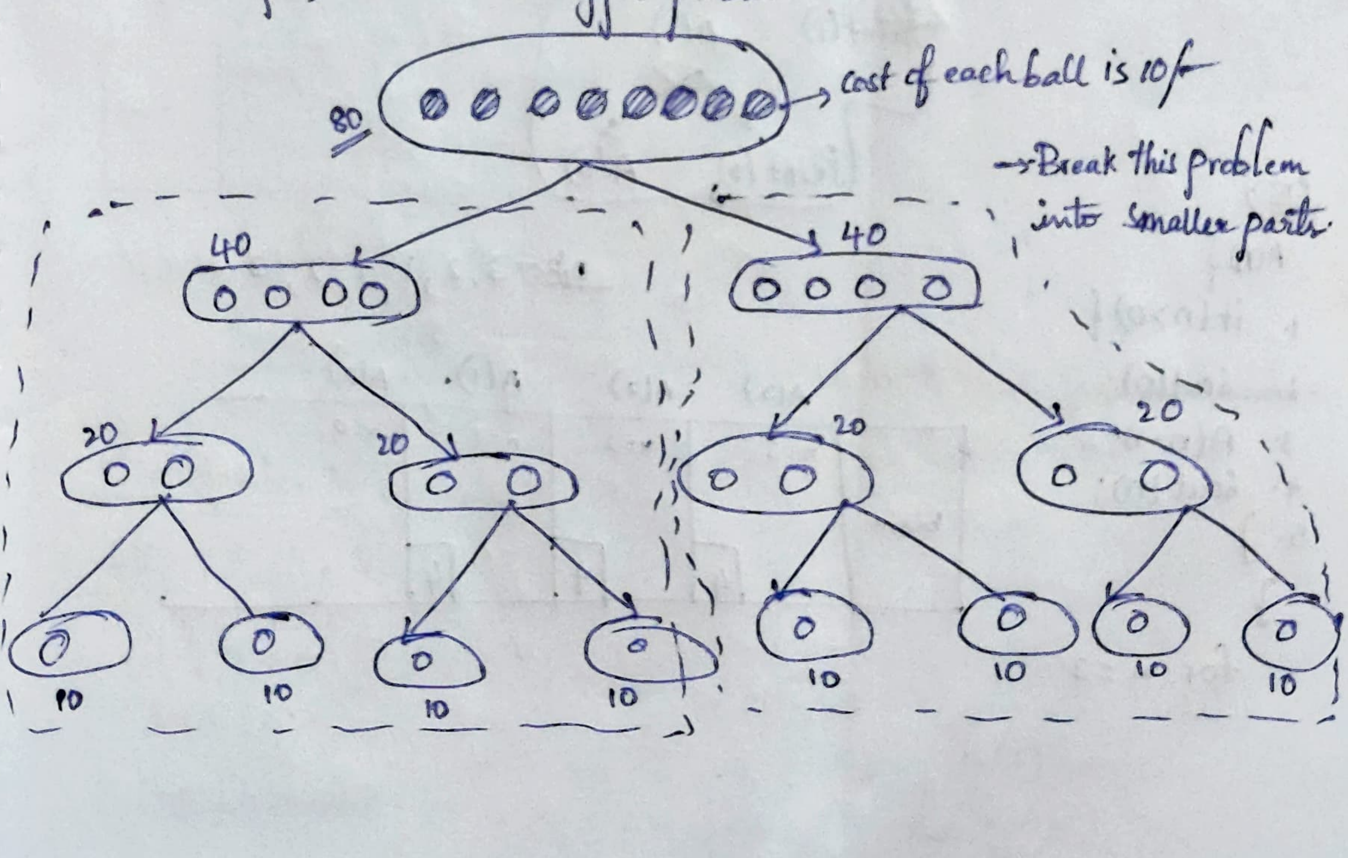
→ If the same function is called within a function it is called Direct Function. as shown in above example.

```
→     A (int x) {

           B(x-1);      → This is indirectfunction.

      }
```

→ If $f(x) = x^2$, can you find $f(f(x))$ → This is Recursion

$\Rightarrow f(x^2) = (x^2)^2 = x^4$

→ forexample, we have a bigger problem.



→ cost of each ball is 10/-

→ Break this problem into smaller parts.

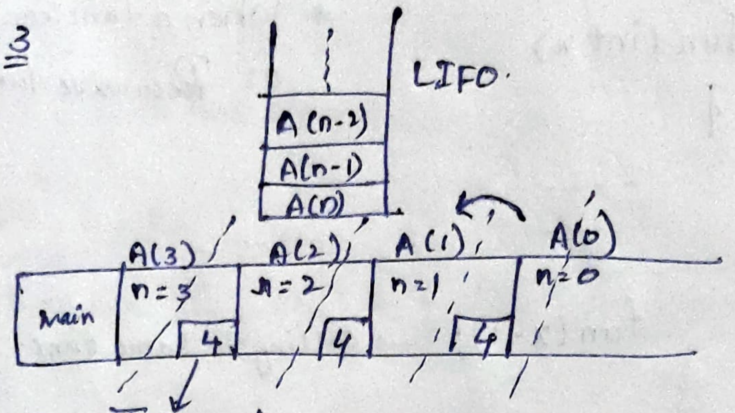
→ Tracing the Recursion:

```
A(n){
1. if(n>0)
2. Sout(n-1);
3. A(n-1);
4. }
5.}
```

n = 3

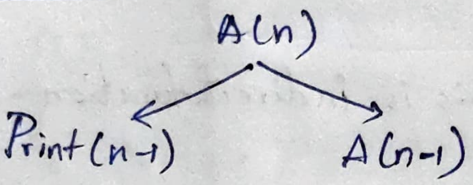
O/P: 2, 1, 0

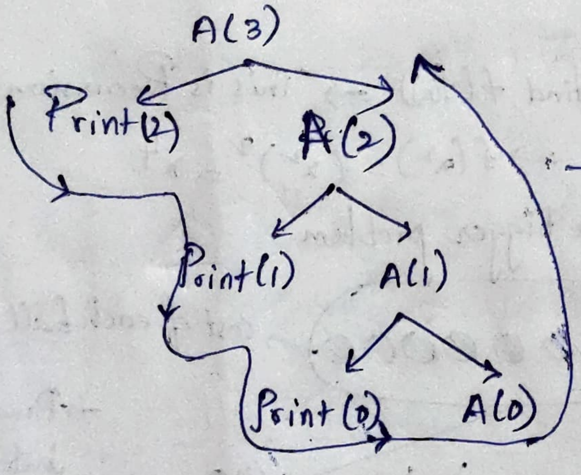
① Recursive Stack.



Instruction pointer
(The Next line to get executed once you are done with recursive call]

② Tree Method:





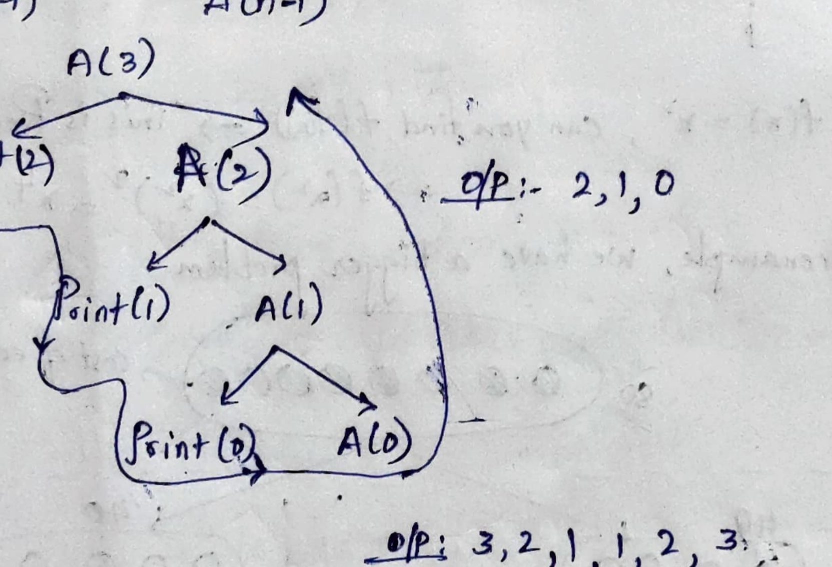
O/P:- 2,1,0

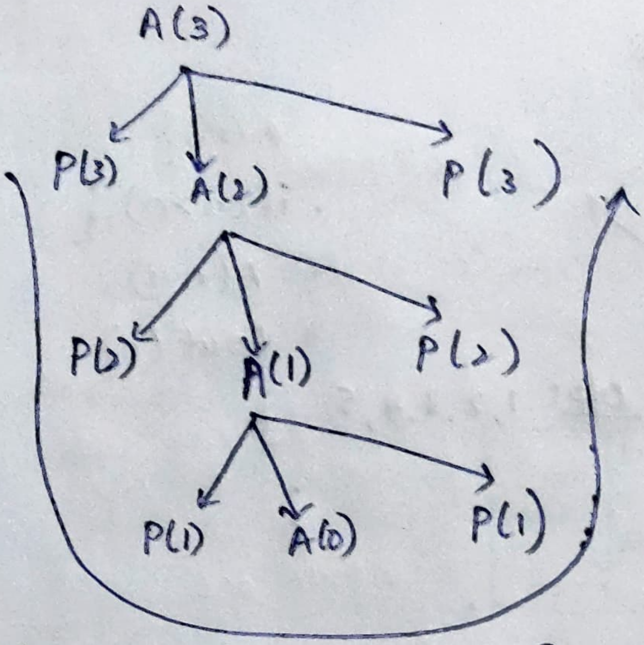
ex: 2

```
A(n){
1. if(n>0){
2. Sout(n);
3. A(n-1);
4. Sout(n);
5. }
}
```

for n=3

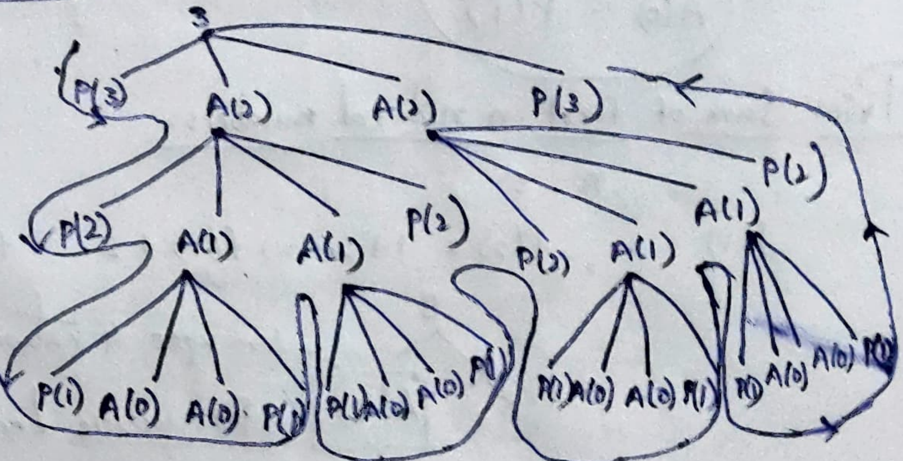
O/P: 3,2,1,1,2,3.



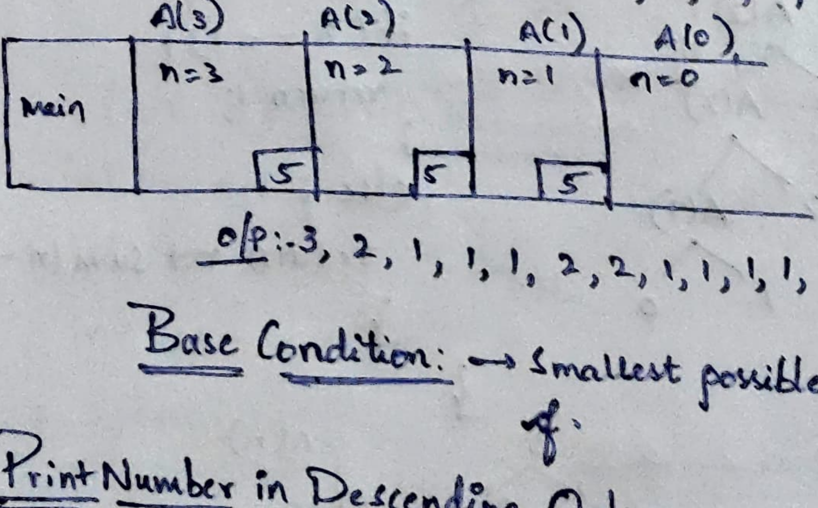


o/p :- 3, 2, 1, 1, 2, 3.

ex:3
```
  A(n) {
1  if (n>0){
2   sout(n);
3   A(n-1);
4   A(n-1);
5   Sout(n);
   }
}
```



o/p: 3, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 3



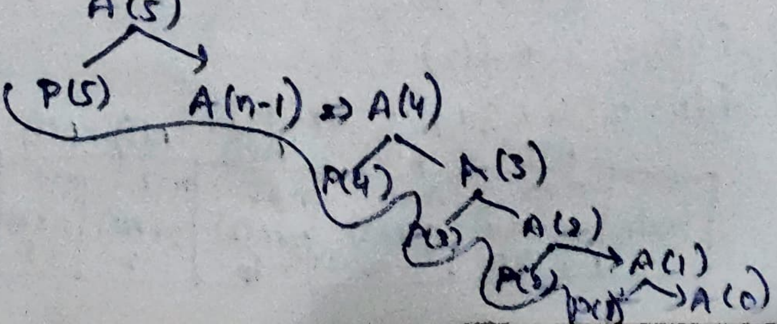
o/p :- 3, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 3

Base Condition: → Smallest possible work that you are aware of.

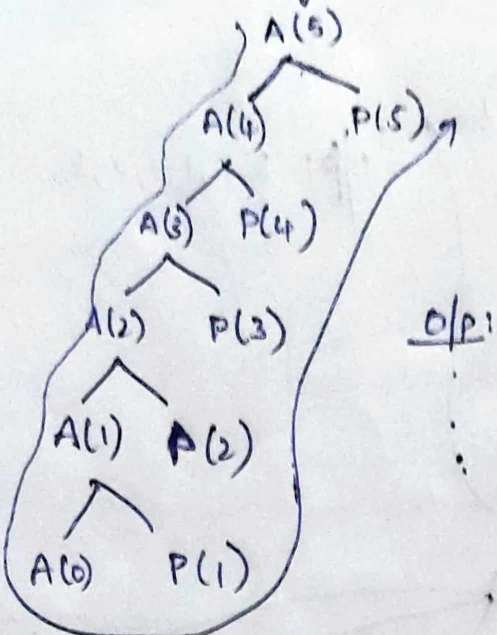
Print Number in Descending Order:

o/p: 5, 4, 3, 2, 1

```
if (n>0){
Sout (n);
PrintDescending (n-1);
}
```



Print Numbers In Ascending Order:



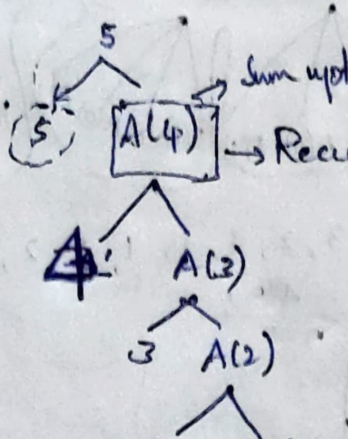
```
A(n){
 . if(n>0),{
 3. A(n-1)
 4. Sout (n)
    }
}
```

O/P: 1,2,3,4,5

Print Sum of first n natural numbers:

n = 5

f(1) = 1 , f(2) = 1+2 ⇒ f(1) + 2   f(3) = f(2) + 3



→ Sum upto 4 natural numbers.

→ Recursive leap of faith.

```
sum(n) {
  if(n==1){
    return 1;
  }
  else {
    return n+sum(n-1);
  }
}
```

Factorial:

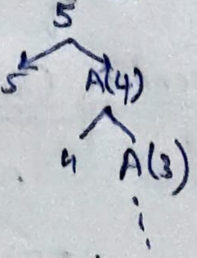
n = 5 , O/P : 120

f(0) = 1 , f(1) = 1

f(2) = 2 * f(1)

f(3) = 3 * f(2)

f(4) = 4×3×2 = 4×f(3)

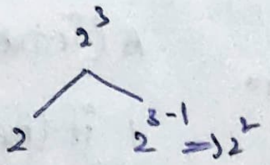


```
A(n)
{
  if(n==0){
    return 1;
  }
  else
  {
    return n* A(n-1);
  }
}
```

	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)
main	n=5 24*A(4) =120	n=4 4*A(3) 4×6=24	n=3 3×A(2) = 6	n=2 2×A(1) = 2	n=1 1×A(0) =1	n=0 1

→ Power of 2 numbers

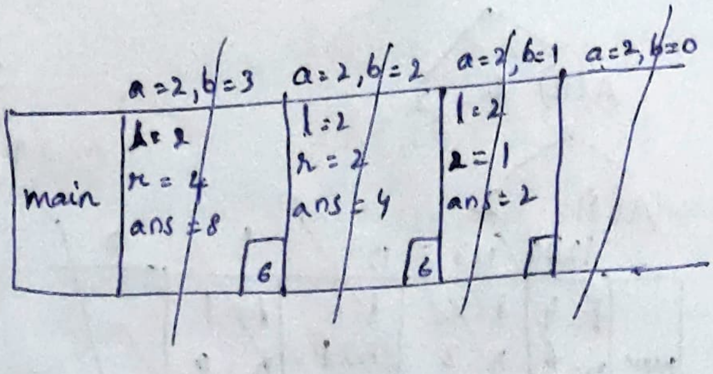
i/p :- 2^3   o/p : 8



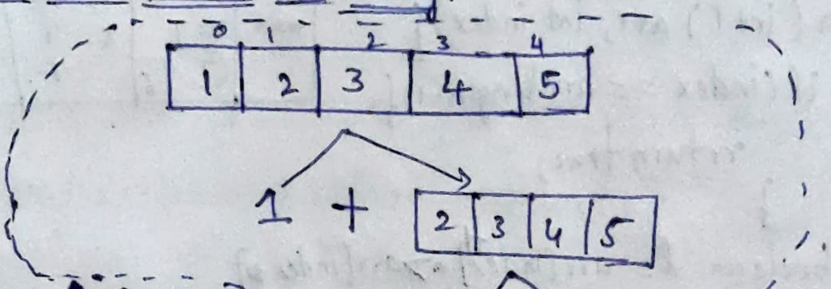
if (exponent == 1)
return base;

$2^0 = 1$, $2^1 = 2$

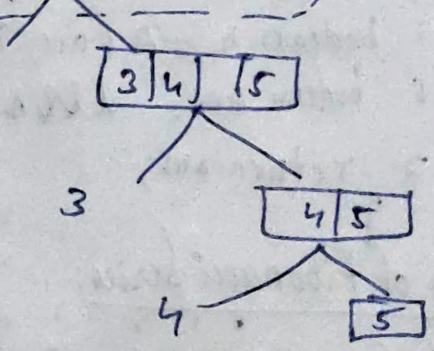
```
A (int a, int b) {
1.   if (b == 0) {
2.     return 1;
3.   }
4  int l = a;
5  int r = A(a, b-1)
6  int ans = l * r;
7. return ans;
}
```

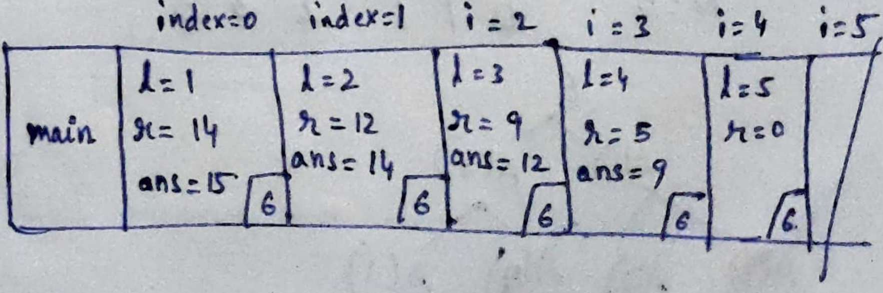


→ Sum of all values inside an Array:



```
A (arr, 0) { 1. if(index==arr.length){
                 2. return 0;
                 3. }
4. int l = arr [index]
5. int r = A [arr, index+1];
6. int ans = l + r;
7. return ans;
}
```

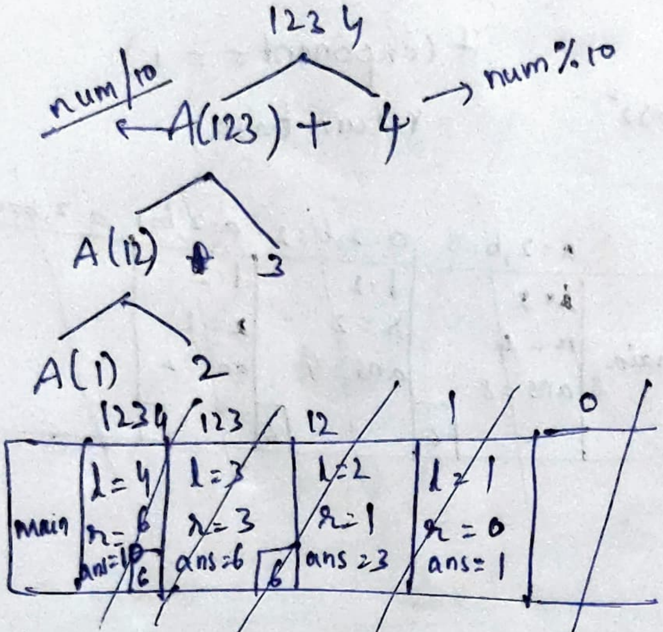




Sum of digits:

I/p:1234 : o/p: 1+2+3+4 = 10.

1)10(0
10
0



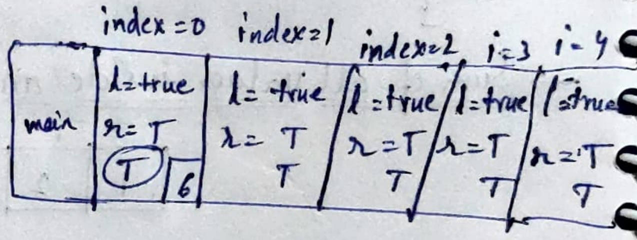
```
A (int num) {
    if (num==0) {
        return 0;
    }
    int l = num % 10;
    int r = A (n /10);
    int ans = l+r
    return ans;
}
```

check whether array is Sorted:

1	2	3	4

o/p: True / false.

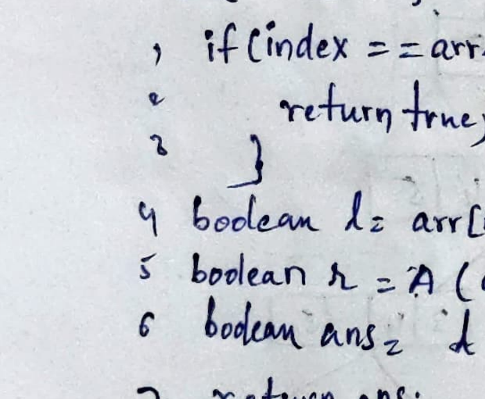
```
A ( int [] arr, int index) {
    if (index == arr.length-1) {
        return true;
    }
    boolean l = arr[index] >= arr[index+1]
    boolean r = A (arr, index+1)
    boolean ans = l && r;
    return ans;
}
```



Nth Term of Fibonacci Series:

N=5

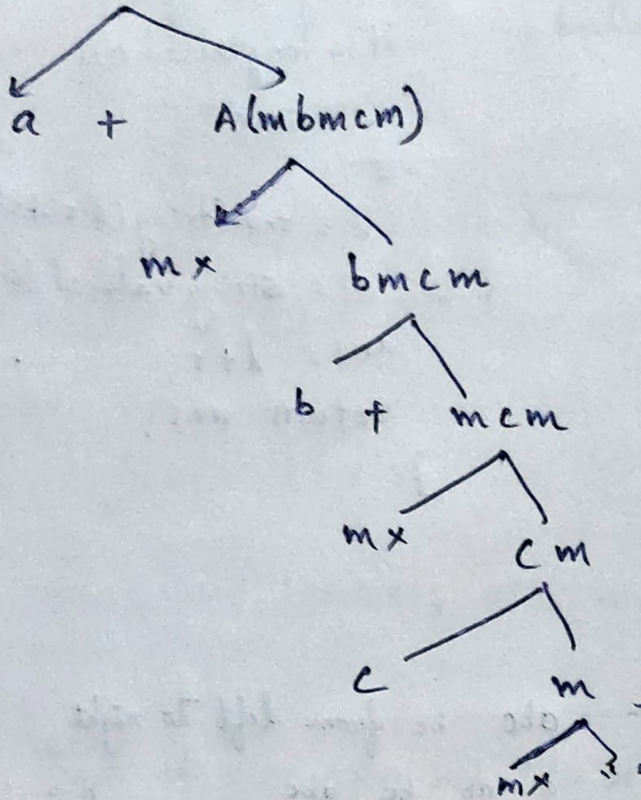
	0	1	2	3	4	5
	0	1	1	2	3	5



```
A(n) {
if (n<=1) {
return n;
}
else
{
return A(n-1) + A(n-2);
}
```

→ Remove 'm' from String:

ambmcm → o/p: abc



```
A(string s)
{
char ch = s.charAt(0);
 if(ch=='m'){
 return A(s.subString(1)).
 }
else {
 return ch + A(s.subString
                   (1));
}
}
```

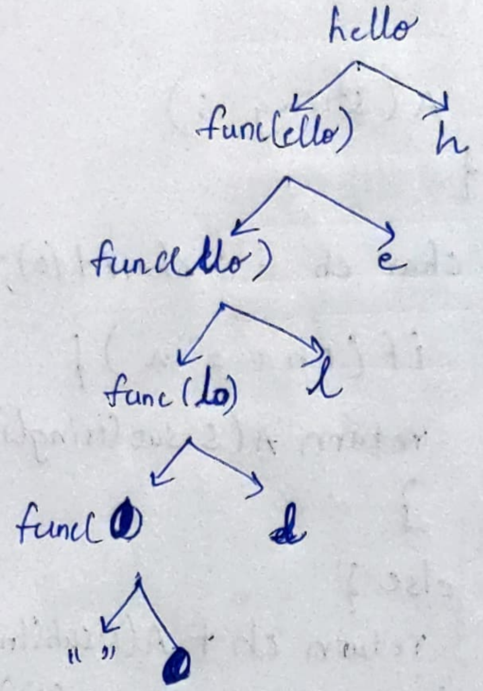
→ First and Last Occurrence of a character in a String:

String s= "abaacdaefaah", element = 'a'   o/p: first: 0
                                                   Last : 10

```
func (String s, int first, int last, char elem, int index)
{
        char ch = s.charAt (index);
        if (ch == element)
        {
            if (first == -1) {
                first = index;
        }
        func( s, first, last, eleme, index++);
}
```

```
if(index==s.length())
{
s.o.pln(first);
last= last!=-1?last:first}
s.o.pln(last);
}
```

String Reversal: "hello" → "olleh"



```
revString (string s){
    if(s.length()== 0){
        return " ";
    }
    l = revString(s.substring(1));
    r = String.valueOf(s.charAt(0));
    ans = l+r;
    return ans;
}
```

* SubSequence of a String:-

abc i.e from left to right

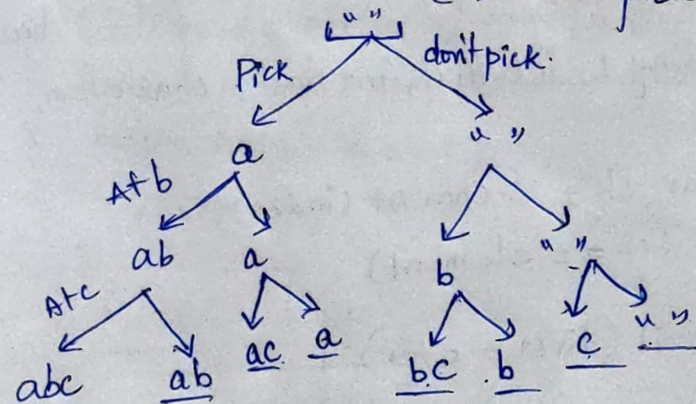
a ab bc abc

b ac

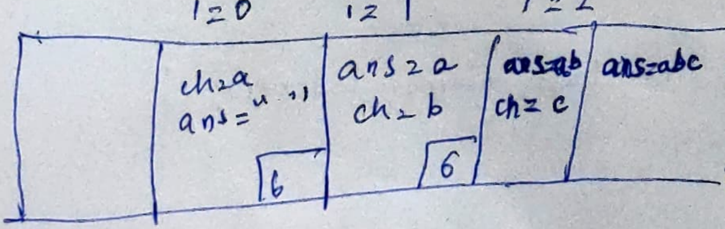
c

b a, ca, cba ✗

→ first start with empty string, at each instance we will have 2 options Pick the current character (or) dont pick the character.



```
subseq(str, index, newStr)
{
1   if (index == str.length())
    {
2   S.O.pln (newStr);
3   }return;

4   char current = str.charAt(index);
    //tobe included
5   subseq(str, index+1, newStr+current);
    //not tobe included
6   subseq(str, index+1, newstr);
```



→ Remove Duplicater in a String.

S = "abccccddab" → o/p:- abcd.

```
       a   b   c   d   e
SArr[] [ F | F | F | F | F | .. ]  → boolean array
         0   1   2   3
```

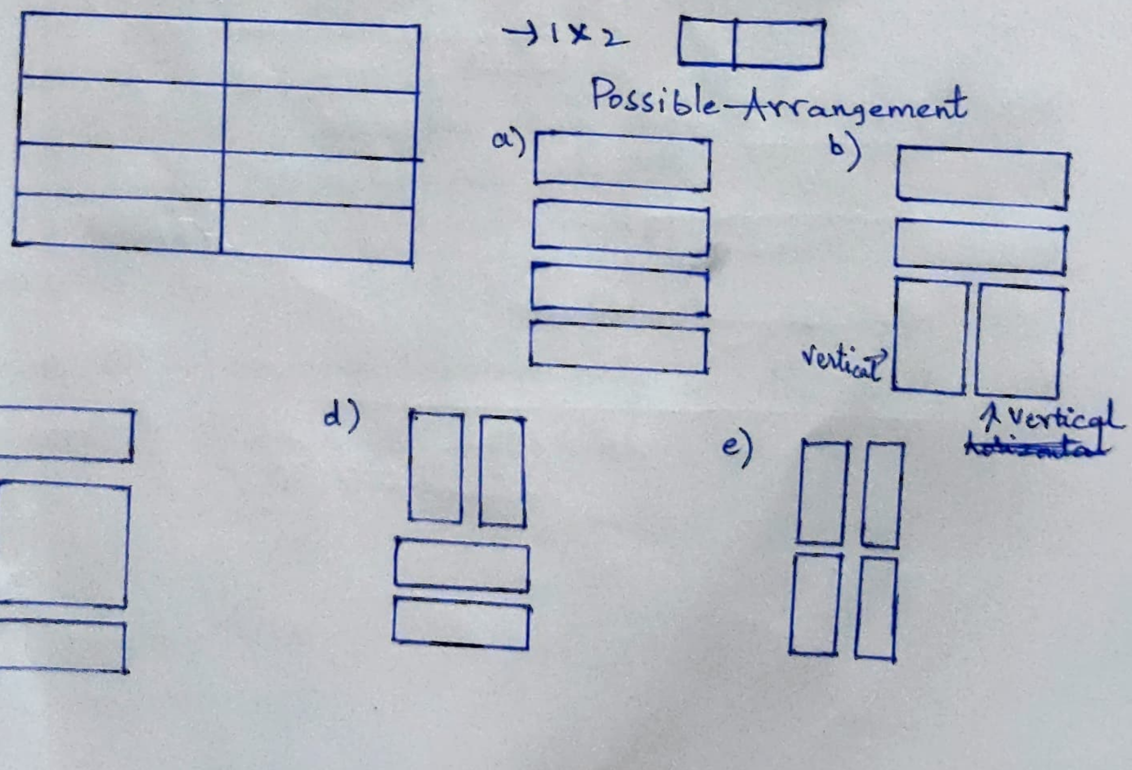
```
                          SArr[]
func(String, 0, " " ") {
    char ch = s.charAt(index);
    if(SArr[ch-'a'] = = 'F')
    {
        ans + = ch;
        SArr[ch-'a'] = 'T';
    }
    func(string, index+1; ans, SArr)
}
```

base condition

```
if (index = = s.length()){
    s.o.pln(ans);
    return;
}
```

→ Given a floor of size n×m and tiles of size 1×m. The problem is to count the number of ways to tile the given floor using 1×m tiles. A tile can be placed either horizontally (or) Vertically. Both n and m are positive integers and 2<=m.

n×m.   n=4, m=2   Tile→ 1×m


→1×2
Possible-Arrangement
a)
b)
vertical
↑ vertical
~~horizontal~~
c)
d)
e)