Leetcode [Q.No.:- 2064]

M T W T F S S | Page No.: | Date: | YOUVA

⇒ Minimized maximum of Products distributed to any.

Ex ① :-

arr = { 11 , 6 } n=6.

i s

warehouse — 11 units of IPhone — 6 units of Samsung.

11i 6s 0 0 0 0 max = 11

Retail stores.

3i 3i 3i 2i 3s 3s max = 3 → minimized maximum

Ex ② arr = { 15 , 10 , 10 } n = 7

a b c → mx

5a 5a 5a 5b 5b 5c 5c

lo = ~~1~~ 5

hi = ~~15~~ ~~7~~ ~~8~~ 4

mid = ~~8~~ ~~4~~ ~~6~~ 5

stores = arr[i] / maxQ + 1 if (arr[i] % maxQ != 0)

Max Q = ~~8~~ ~~6~~ 5.

⇒ Code :- Psvm (int[] arr , int n , int m , int mx) {

arr.length → m ; max^m ele → mx

int lo = 1 , hi = mx ;

int ans = 0 ;

while (lo <= hi) {

int mid = lo + (hi - lo) / 2 ;

if (isPossible (mid , n , arr)) {

ans = mid ;

hi = mid - 1 ;

}

else lo = mid + 1 ;

}

Sout (ans) ;

}

⇒ Function :- Ps boolean isPossible (int maxQ , int n , int[] arr) {

int stores = 0 ;

for (int i = 0 ; i < arr.length ; i++) {

if (arr[i] % maxQ == 0) stores += arr[i] / maxQ ;

else stores += arr[i] / maxQ + 1 ;

}

if (stores > n) return false ;

else return true ;

}

Leet Code [Q.No:- 1539],

⇒ k^{th} Missing Positive Number [Binary Search + Math] → O(log n).

Ex (1):-

	0	1	2	3	4
arr = {	2 ,	3 ,	4 ,	7 ,	11 }
				hi	lo
No. of missed ele:- till that index.	1	1	1	3	3

k = 5.

missed = arr[mid] - (mid + 1)

if (missed < k) lo = mid + 1;

else hi = mid - 1.

k^{th} missing Number = arr[hi] + extra (Math)

extra = k - (missed no.s till hi).

= k - (arr[hi] - (hi + 1))

⇒ Observations :- The k^{th} missing No. is b/w arr[hi] & arr[lo].

k^{th} missing no. = arr[hi] + extra.

= arr[hi] + k - (arr[hi] - (hi + 1))

= ~~arr[hi]~~ + k - ~~arr[hi]~~ - (hi + 1)

= k + (hi + 1).

or.

k + lo

⇒ Code :-

```
Psvm {
    int lo = 0, hi = arr.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo)/2;
        int missed = arr[mid] - (mid + 1);
        if (missed < k) lo = mid + 1;
        else hi = mid - 1;
    }
    Sout (k + lo);
}
```

Recursion.

5 / Sep. / 2024.

What & Why ?

If you don't study recursion → Trees, DP, Graphs → DFS

Recursion is not a D.S, it is an algo.

↳ We believe in Recursion.

→ Recurrance Relations → Big Problem → Using Smaller Subproblems.

(Method)

⇒ It's Nothing but Function calling it self.

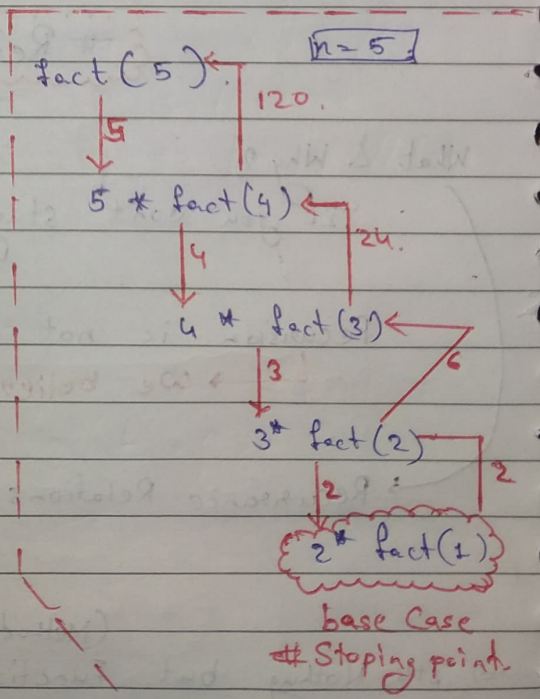
⇒ Function Calls :- ①

⇒ Code :-

```
Psv mango(){
    Sout("Hi, I am in mang");
}
Psv banana(){
    Sout("Hi, I am in banana");
    mango();
}
Psv apple(){
    Sout("Hi, I am in apple");
    banana();
}
JVM
Psv(m){
    Sout("Hi, I am in main");
    apple();
}
JVM
```

Output.

- Hi, I am in main
- Hi, I am in apple
- Hi, I am in banana
- Hi, I am in mango



⇒ Factorial Recurrence Relation:

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$

$6! = 6 \times 5!$

$8! = \overbrace{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1}^{7!}$

$8! = 8 \times 7!$

$n! = n \times n-1 \times n-2 \times n-3 \times \ldots\ldots n-1$

$n! = n \times (n-1)!$

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

$f(x) = x!$

$f(3) = 3!$, $f(2) = 2!$

⇒ Function Calls (2) :-

⇒ Code :-

```
Psv mango() {
    Sout("Hi, I am in mango");
}

Psv banana() {
    mango();
    Sout("Hi, I am in banana");
}

Psv apple() {
    banana();
    Sout("Hi, I am in apple");
    mango();
}

JVM

Psv main {
    Sout("Hi, I am in main");
    apple();
}

JVM
```

Output :-

- Hi, I am in main.
- Hi, I am in mango.
- Hi, I am in banana.
- Hi, I am in apple.
- Hi, I am in mango.

Q1. Make a function which calculates the factorial of n using recursion?

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

⇒ Code :-

```
Psvm() {
    Scanner sc = new Scanner(System.in);
    Sout("Enter n: ");
    int n = sc.nextInt();
    Sout(fact(n));
}
```

→ Base Case :- Need to give stoping point else it will be in ∞ loop.

→ Call :- It is calling it self.

function :-

```
Ps ▪ int fact(int n) {
    if (n==1) return 1; // Base case
    int ans = n * fact(n-1); // Call
    return ans;
}
```

8/ Sep/ 2024

Page No.: YOUVA

Q2. Print n to 1 → Using Recursion

↳ Take a number 'n' as input & print n to 1.

For ex:- n = 5 ⟶ Output => 5, 4, 3, 2, 1.

=> Code :-

```
psv print(int){
    if(n==0) return;
    Sout(n);
    print(n-1);
}

psvm {
    Scanner sc = new Scanner(System.in);
    Sout("Enter n : ");
    int n = sc.nextInt();
    print(n);
}
```

n = 6

↳ Output :-

6
5
4
3
2
1

Q3. Print 1 to n (extra parameter) → Recursion

Note :- From here wherever, I use 'n' imagine that I have take it as input.

M-1 (We don't prefer to use)

=> Code :- // Using Global Variable

```
Static int n;
psv print(int x){
    if(x>n) return; // Base case
    Sout(x); // Work
    print(x+1); // Call
}

psv m {
    n = sc.nextInt();
    print(1);
}
```

M-2

=> Code :- Extra Parameter

```
psv print(int x, int n){
    if(x>n) return; // Base case
    Sout(x); // Work
    print(x+1, n); // Call.
}

psv m {
    int n = sc.nextInt();
    print(1, n);
}
```

=> Recursion :-

```
fun() {
    base Case;
    work;
    Call;
    Work;
}
```

```
fun() {
    base Case;
    Call;
    Work;
    Call;
```

Qu. Print 1 to n (After recursive call) M-3.

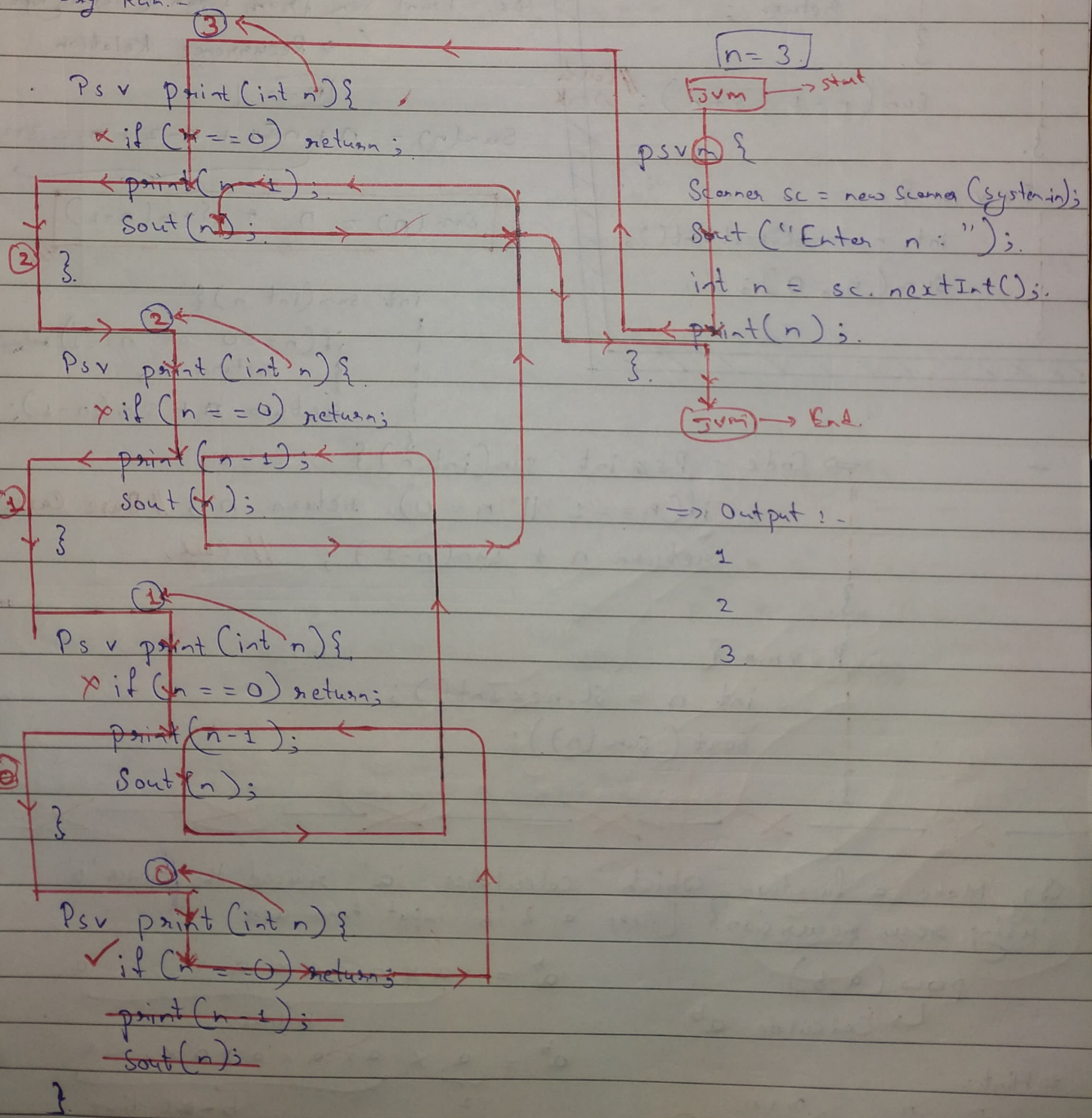
=> Code :-

```
psv print(int n){
    if (n==0) return; // Base Case.
    //Sout(n); // work
    print(n-1); // Call
    Sout(n); // Work.
}
```

→ If you want to print 1st n to 1 & then 1 to n.

```
psv main {
    int n = sc.nextInt();
    print(n);
}
```

Dry Run :-



=> Output :-

1

2

3

Q5. Print sum of from 1 to n [Parameterised] → Recursion

n = 5 ⟶ Sum = 1 + 2 + 3 + 4 + 5
or
5 + 4 + 3 + 2 + 1.

using for loop:

```
Sum = 0;
for(int i = 1; i <= n; i++){
    Sum += i;
}
```

[M-1]

Sum(5, 0) → Sum(4, 5) → Sum(3, 9) → Sum(2, 12) → Sum(1, 14) → Sum(0, 15)

Base case. Calculate Sum.

⇒ Code :-

```
Ps v sum(int n, int s){
    if(n == 0){ //Base Case
        Sout(s);
        return;
    }
    Sum(n-1, s+n); // Call & work.
}
Ps v m {
    int n = sc.nextInt();
    sum(n, 0);
}
```

Q6. Print sum from 1 to n (Return Type).

⟶ Recurrence Relation.

[M-2]

Sum(n) = n + n-1 + n-2 + n ... 3 + 2 + 1

Sum(n) = n + Sum(n-1)

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

⇒ Code :-

```
Ps sv int sum(int n){
    if(n == 1 || n == 0) return n; // Base Case
    return n + sum(n-1); // call.
}
Ps v m {
    int n = sc.nextInt();
    Sout(sum(n));
}
```

Q7. Make a function which calculates 'a' raised the power 'b' using recursion. [Take 'a' & 'b' input from user.

pow(a,b)
↳ Calculate a^b

$a^b = a \times a \times a \times a \ldots a$ (b times)

$a^b = a \times a \times a \times a \times a \times \ldots a$ (b-1 times), with the leading a = 1

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

Hint :-

$x^m \cdot x^n = x^{m+n}$

pow(a,b) = a * pow(a,b-1);

M-1

⇒ Code :- PS# int pow(int a, int b){
if(b==0) return 1;
return a * pow(a, b-1);
}

psvm{
~~int a = sc.nextInt("Enter base");~~
~~int b = sc.nextInt();~~
int a = sc.nextInt(); // [Enter base]
int b = sc.nextInt(); // [Enter power]
Sout(a + "raised to the power" + b + "is" + pow(a,b));
}

Day Run :-

M	T	W	T	F	S	S

Page No. | YOUVA
Date:

$(3)^5$ → 3×3^4 → 3×3^3 → 3×3^2 → 3×3^1 → 3×3^0 → Base case.

T.C = O(b)
↓
Exponent as in a^b.

Q8. Power function [logarithmic]

$2^{64} = 2^* 2^{63}$
$2^{63} = 2^* 2^{62}$
$2^{62} = 2^* 2^{61}$
⋮
$2^1 = 2^* 2^0$

64 calls [Bekar & Very Slow Slow]

* Hint : $x^{m+n} = x^m \cdot x^n$

$a^b \rightarrow$ T.C = $O(\log_2 b)$

$2^{64} = 2^{32} * 2^{32}$
$2^{32} = 2^{16} * 2^{16}$
$2^{16} = 2^8 * 2^8$. # $a^b = (a^{b/2})^2$
$2^8 = 2^4 * 2^4$.
$2^4 = 2^2 * 2^2$.
$2^2 = 2^1 * 2^1$.

6 calls. = $\log_2 64$

$a^b = a^{b/2} * a^{b/2}$.
int ans = pow(a, b/2);
pow(a,b) = ans * ans;
↓
Wrong.

$2^5 = 2^2 * 2^2 * 2$
$3^7 = 3^3 * 3^3 * 3$ → Extra when b is odd
$3^8 = 3^4 * 3^4$

M-2

⇒ Code :- ps int pow2(int a, int b){
if(b==0) return 1;
int ans = pow2(a, b/2);
if(b%2==0) return ans * ans;
else return ans * ans * a;
}

Psvm{ int a = sc.nextInt();
// int b = sc.nextInt();
Sout("a +" raise to the power" + b + "is" + pow2(a,b));
}

Day Run :-

3^9
↓
$(3^4)^2 * 3$
↓
$(3^2)^2 * 3$
↓
$(3^1)^2 * 3$.
↓
$(3^0)^2 * 3$. ← Base case.

[#LeetCode 509.]

Q9. Write a function to calculate the n^{th} fibonacci Number using recursion.

n =	0	1	2	3	4	5	6	7	8	9	10	11	.....
	0	1	1	2	3	5	8	13	21	34	55	89	.....

$$fibo(n) = fibo(n-1) + fibo(n-2)$$

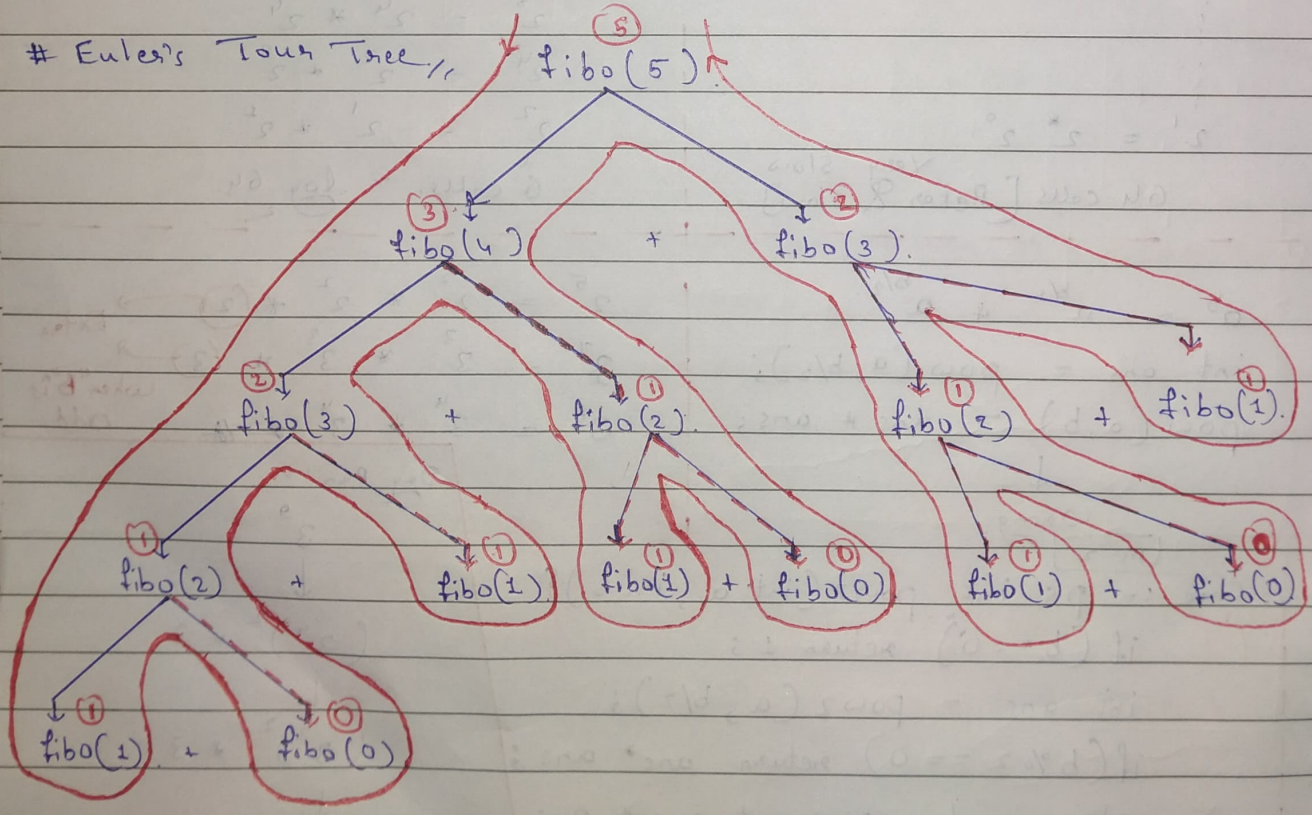
Base Case → if (n <= 1) return n;

⇒ Code :-

```
ps int fibo(int n) {
    if(n <= 1) return n;
    return fibo(n-1) + fibo(n-2);
}

psvm {
    int n = sc.nextInt();
    sout(fibo(n));
}
```

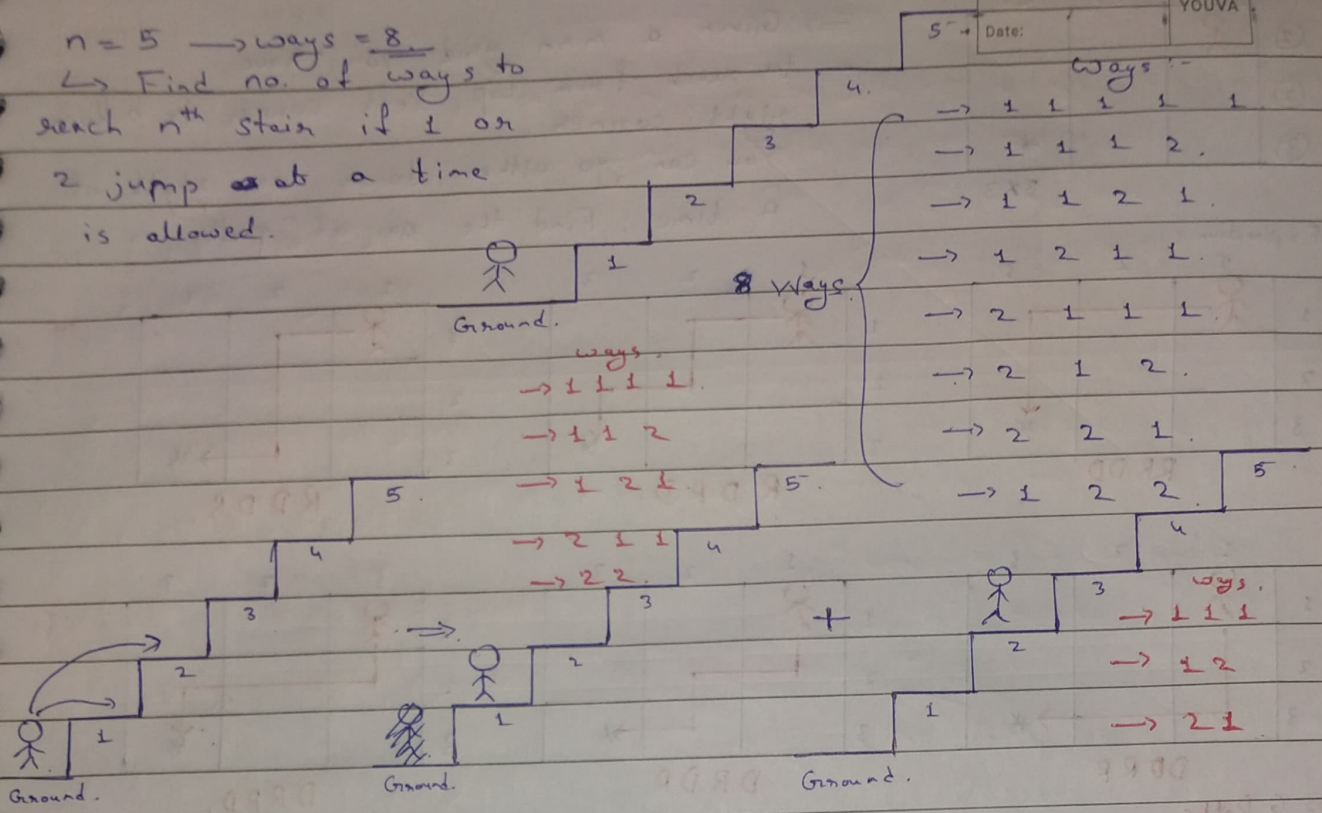
Euler's Tour Tree :-



Stair Path.

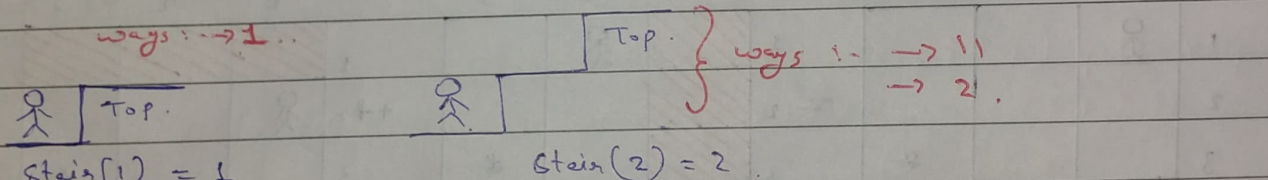
n = 5 → ways = 8

↳ Find no. of ways to reach n^{th} stair if 1 or 2 jump at a time is allowed.



Stair(5) = Stair(4) + Stair(3).

Stair(n) = Stair(n-1) + Stair(n-2). Exactly same like fibonacci



=> Code :- Ps int stair(int n){

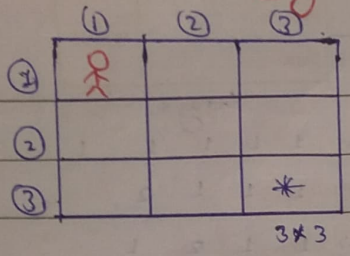
```
    if (n <= 2) return n;
// if (n == 1) return 1;
// if (n == 2) return 2;
    return stair(n-1) + stair(n-2);
}
```

→ You can use this two also instead of first both are same only.

```
psvm {
    int n = sc.nextInt();
    Sout (stair(n));
}
```

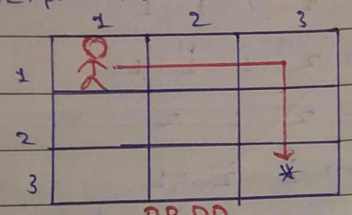
→ Rat in maze.

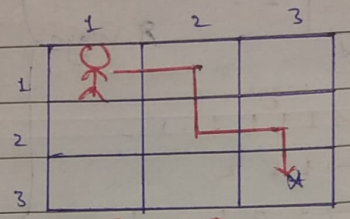
Maze Path [Very Imp]

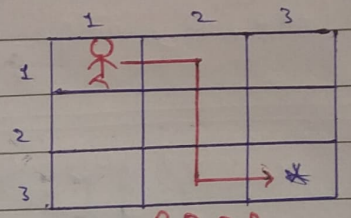


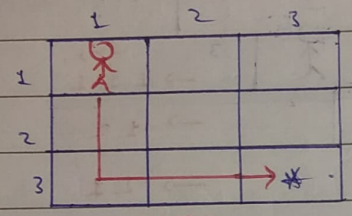
=> Given a m x n grid you have to reach from top left corner to bottom right corner. You can go either down or right at a time. Find the no. of paths.

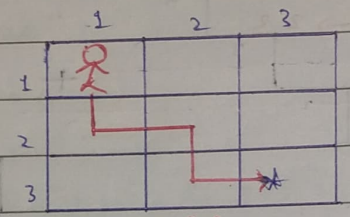
=> Explanation :-

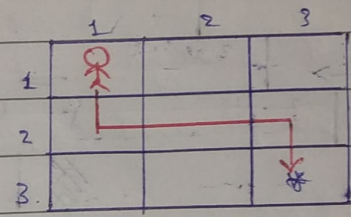








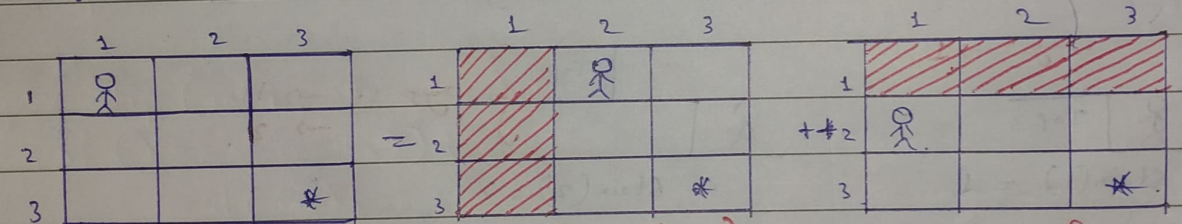




=> 6 Paths //

=> Logic :-

M-1

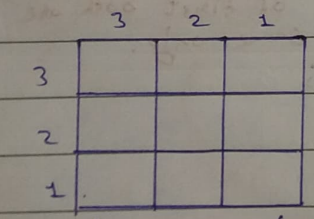


(1,1) → (3,3) = (1,2) → (3,3) + (2,1) → (3,3)

total ways = rightways + downways

Base Case → if (row == m or col == n) return 1;

M-2



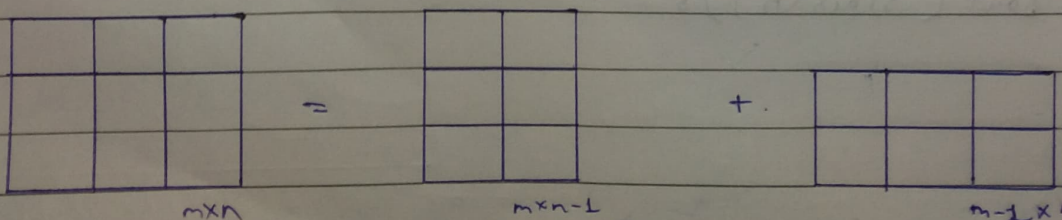
right → col--

down → row--

(m,n) → (1,1)

or

(1,1) → (m,n)



[M-1]

=> Code :-

```
Ps int maze(int row, int col, int m, int n){
    if(col == n || row == n) return 1;
    // if(row == m && col == n) return 1;
    // if(row > m || col > n) return 0;
    int rightWays = maze(row, col+1, m, n);
    int downWays = maze(row+1, col, m, n);
    return rightWays + downWays;
}
```

[M-2]

```
Psvm{
    int n = sc.nextInt();
    int m = sc.nextInt();
    //Sout(maze(1,1,m,n));
    Sout(maze2(m,n));
}
```

```
ps int maze2(int row, int col){
    if(col == 1 || row == 1) return 1;
    int rightWays = maze2(row, col-1);
    int downWays = maze2(row-1, col);
    return rightWays + downWays;
}
```

Pre In Post :-

```
fun(){
    base case
    work
    fun()
    work
    fun()
    work.
}
```

=> Predict the output.

=> Code :-

```
Psv pip(int n){
    if(n == 0) return;
    Sout(n + " ");
    pip(n-1);
    Sout(n + " ");
    pip(n-1);
    Sout(n + " ");
}
```

```
Psvm{
    pip(3);
}
```

=> You can dry run this using call stack technique. we will use different method.

Dry Run :-

pip(3) :

3 pip(2) 3 pip(2) 3.

pip(2)

2 pip(1) 2 pip(1) 2.

pip(1)

1 pip(0) 1 pip(0) 1.

pip(0)

Base Case → Nothing

Output => 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

Traversing an array using recursion.

→ Print all the elements of an array.

⇒ Code :-

```
psv print(int i, int[] arr){
    if(i == arr.length) return;
    Sout(arr[i] + " ");
    print(i+1, arr);
}
```

```
psvm {
    int[] arr = {4,7,1,3,8};
    print(0, arr);
}
```

Skip a Character

↳ Remove all occurrences of 'a' from a string.

```
String s = "Raghav";
String ans = "Rghv";
```

→ for printing String using recursion.

⇒ Code :-

```
psv print(int i, String s){
    if(i == s.length()) return;
    Sout(s.charAt(i));
    print(i+1, s);
}
```

```
psv skip(int i, String s, String ans){
    if(i == s.length()){
        sout(ans);
        return;
    }
    if(s.charAt(i) != 'a') ans += s.charAt(i);
    skip(i+1, s, ans);
}
```

```
psvm {
    String s = "Raghav";
    // print(0, s);
    skip(0, s, "");
}
```

skip(0, "Raghav", "");

↓

skip(1, "Raghav", "R");

↓

skip(2, "Raghav", "R");

↓

skip(3, "Raghav", "Rg");

↓

skip(4, "Raghav", "Rgh");

↓

skip(5, "Raghav", "Rgh");

↓

skip(6, "Raghav", "Rghv");

Traversing an array using recursion.

T.C = O(n)

→ Print all the elements of an array.

```
⇒ Code :- psv print(int i, int[] arr){
    if(i == arr.length) return;
    Sout(arr[i] + " ");
    print(i+1, arr);
}

psvm {
    int[] arr = {4,7,1,3,8};
    print(0, arr);
}
```

Skip a Character #T.C => O(n).

↳ Remove all occurrences of 'a' from a string.

```
String s = "Raghav";
String ans = "Rghv";
```

→ for printing using string recursion.

```
⇒ Code :- psv print(int i, String s){
    if(i == s.length()) return;
    Sout(s.charAt(i));
    print(i+1, s);
}
```

skip(0, "Raghav", "");
↓
skip(1, "Raghav", "R");
↓
skip(2, "Raghav", "R");
↓
skip(3, "Raghav", "Rg");
↓
skip(4, "Raghav", "Rgh");
↓
skip(5, "Raghav", "Rgh");
↓
skip(6, "Raghav", "Rghv");

```
psv skip(int i, String s, String ans){
    if(i == s.length()){
        Sout(ans);
        return;
    }
    if(s.charAt(i) != 'a') ans += s.charAt(i);
    skip(i+1, s, ans);
}

psvm {
    String s = "Raghav";
    // print(0, s);
    skip(0, s, "");
}
```

Subsets :-

→ Print subsets of a string with unique characters.

{1, 2, 3}

↓

{{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}

Power set is set of all subsets.

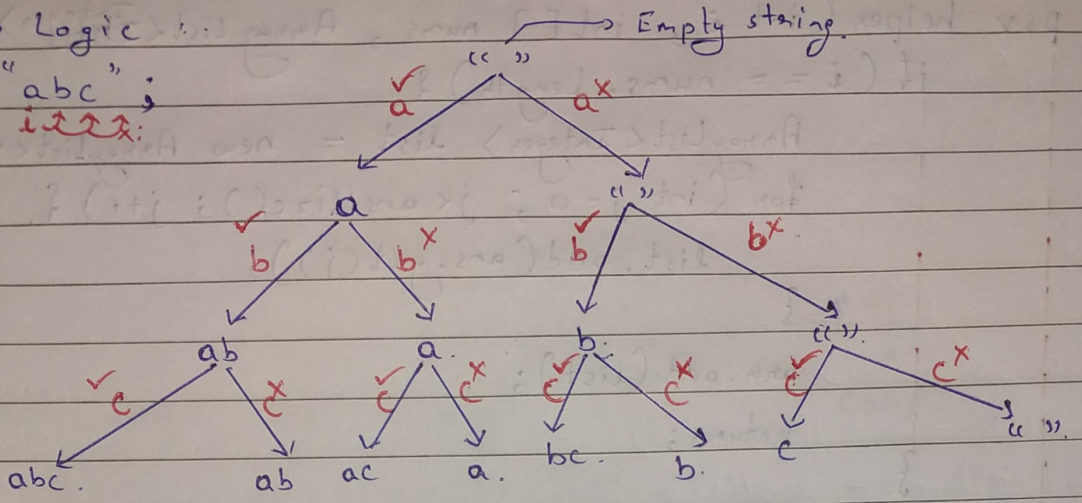
String s = "abcd";

ϕ, a, b, c, d, ab, ac, ad, bc, bd, cd, abc, abd, acd, bcd, abcd.

M T W T F S S | Page No.: | Date: | YOUVA

=> Logic :-

s = "abc";



2 + 4 + 8 ...

tnc = $2^1 + 2^2 + 2^3 \ldots\ldots 2^n \simeq 2 \cdot 2^n$ → T.C = O(2^n).

=> Code :-

```
Static ArrayList<String> arr = new ArrayList<>(); // Global.
psv printSubsets(int i, String s, String ans){
    if(i == s.length()){
        // Sout(ans);
        arr.add(ans);
        return;
    }
    // char ch = s.charAt(i);
    printSubsets(i+1, s, ans); // Not Take.
    ans += s.charAt(i);
    printSubsets(i+1, s, ans); // Take
}
psvm{
    String s = "abcd";
    arr = new ArrayList<>(); // reset Good habit when you use global variable.
    printSubsets(0, s, "");
    Sout(arr);
}
}
```

⇒ Subset of array [LeetCode Q. No. :- 78]

⇒ Code :-

```
{
    static List<List<Integer>> arr;
psv helper(int i, int[] nums, ArrayList<Integer> ans){
        if(i == nums.length){
            ArrayList<Integer> list = new ArrayList<>();
            for(int j=0; j<ans.size(); j++){
                list.add(ans.get(j));
            }
            arr.add(list);
            return;
        }
        helper(i+1, nums, ans); // Not Take
        ans.add(nums[i]);
        helper(i+1, nums, ans); // Take
        ans.remove(ans.size()-1);
}
psvm{
        int[] nums = {1, 2, 3};
        arr = new ArrayList<>();
        ArrayList<Integer> ans = new ArrayList<>();
        helper(0, nums, ans);
        Sout(arr);
}
}
```

---

* Permutations :- → Find all permutations of an ~~Array~~ String given all elements of the String are unique

String s = abc.

n! permutations

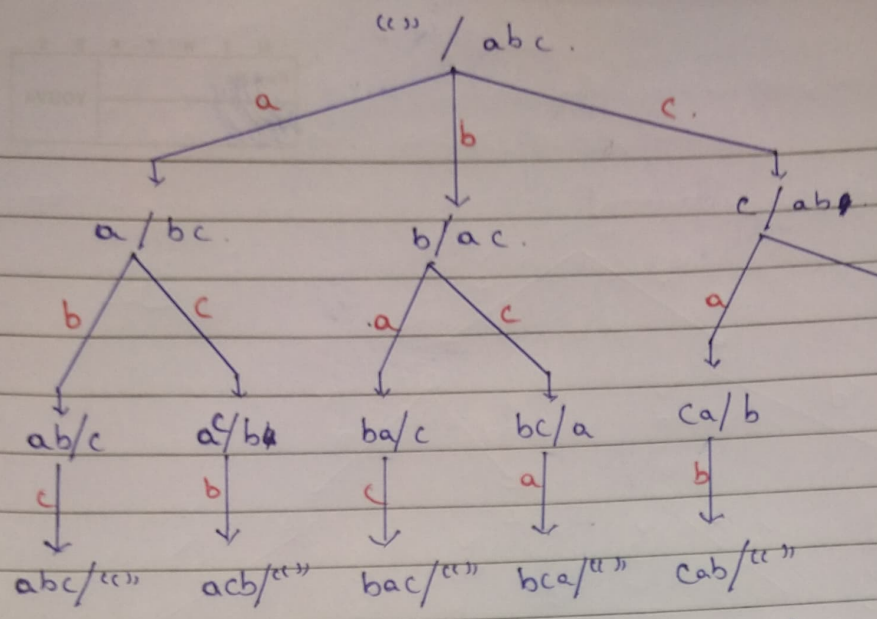
↓

abc, acb, bac, bca, cab, cba

abcdef → abdef

a b c d e f

0 to i-1 (Left) | i | i+1 to end (Right)



T.n.c = n(n-1)(n-2)...
= n!
T.C = O(n!)

=> Code :- psv printPermutations (String ans, String s) {

```
    if (s.length() == 0) {
        sout(ans);
        return;
    }
    for(int i=0; i<s.length(); i++) {
        char ch = s.charAt(i);
        String left = s.substring(0, i);
        String right = s.substring(i+1);
        printPermutations(ans + ch, left + right);
    }
}

psvm {
    String s = "abcd";
    printPermutations("", s);
}
```