Recursion → Iterations → Functions

Trees, Graphs, DP } used in recursion

using math

$f(x) = x^2$

$f(f(x)) = (x^2)^2$

$x = 2$

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

$f(f(2)) = 4^2 = 16$

- factorial

$n! = n \times n-1 \times \dots 1$

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

$(n-1) \times f(n-2)$

$(n-2) \times f(n-3)$

$1 \times f(0)$

$f(5) = 5 \times f(4) = 24$

$4 \times f(3) = 6$

$3 \times f(2) = 2$

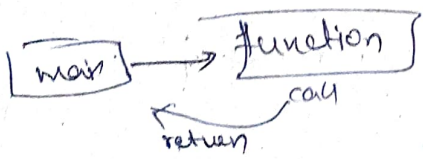
$2 \times f(1) = 1$

$1 \times f(0)$

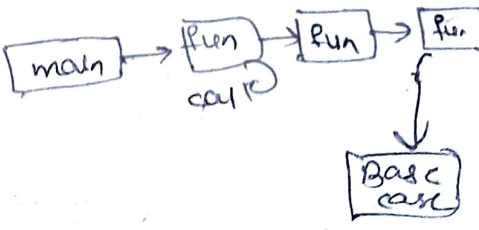
Recursion:- Recursion is a method of solving computational problem where the solution depends on solutions to smaller instances of the same problem.

We must know . Base case ($0! = 1$)

2 directions



in recursion



Recursion

1) Base case
2) work (n x f(n-1))
3) Inner function (f(n-1))

Problem - 1 print numbers from n to 1 (Decreasing order)

n=10 By recursion.

1) → f(n) → print from n to 1
10 * f(n-1) → 9 to 1
9 * f(n-1) → 8 to 1
8 * f(8) → 7 to 1

```
fun(int n)
{
  print(n);
  fun(n-1);
}
```

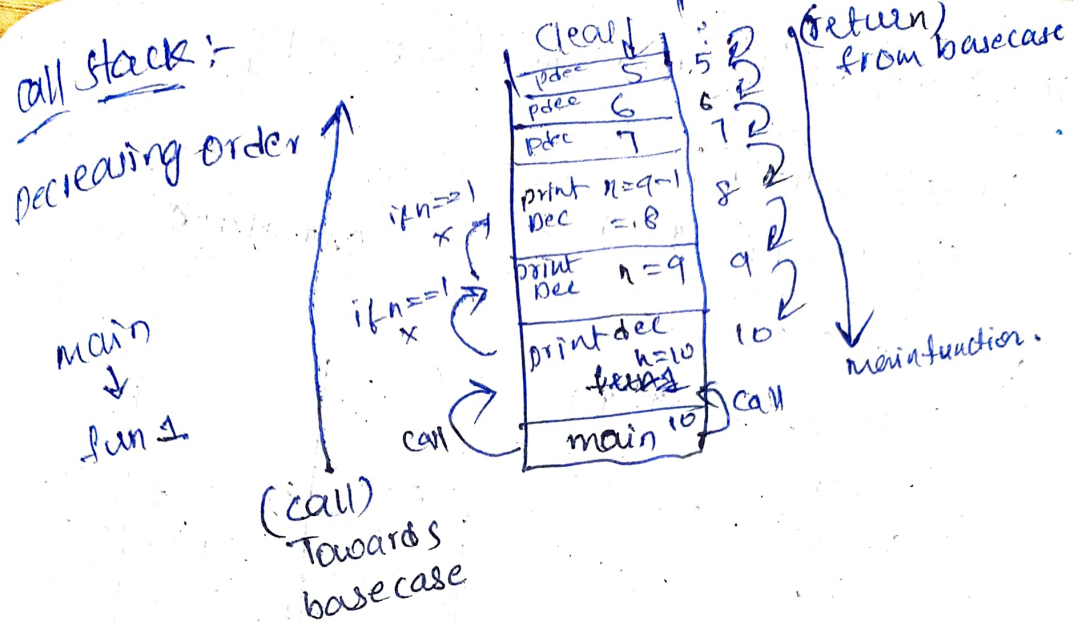
Up to Base case

Code :-

```
public class recursion {
    public static void printDec(int n) {
        Syso(n + " ");
        printDec(n-1);
        if (n==1) {
            Syso(n);
            return n;
        }
    public static void main {
        int n=10;
        printDec(n);
```

call stack :-

decreasing order



stack overflow :-

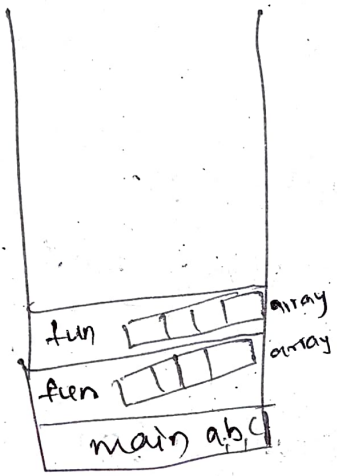
① parameters. [memory size is high]

② too many calls

(If we didn't write base case)

Our program crashes

it is called stack overflow



Problem 2

print numbers from n to 1 (increasing order)

```
fun (int n)
```

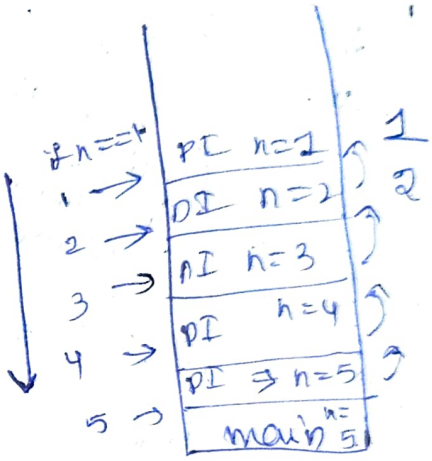
1 2 3 4 5 (Base case (n), int(n))

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

$f(n-2) + (n-1)$

```
public static void printInc (int n) {
    printInc(n-1);
    System.out.print(n + " ");
    if (n==1) {
        System.out.print(n + " ");
        return;
    }
}
PSVMS {
    int n=10;
    printInc(n);
```

call stack



Prob 3

print factorial of a number n.

$n! = n \times (n-1)!$ → $f(n) = n * f(n-1)$

$(n-1)\ f(n-2)$

```
int fac(int n) {
    FN-1 = fact(n-1);
    FN = n * FN-1
    return FN;
}
```

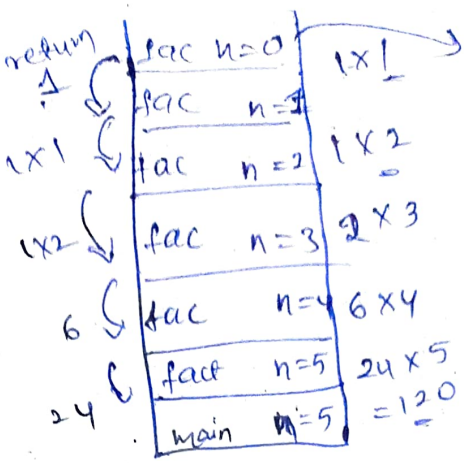
Base case = 0! = 1

```
if (n==0)
    return 1;
```

code :-

```
public static int fact (int n) {
    if (n==0) {
        return 1;
    }
    if fnm1 = fac(n-1);
    int fn = n * fact(n-1);
        return fn; }
```

psvms A[] {
int n=5;
sysO(fact(5))



n=0 ⇒ return 1

f(n-1

$f_n = n \times f_{n-1}$
return f_n.

Time complexity = O(n)
Space complexity = O(n)

print sum of ~~first~~ natural numbers.

n=5

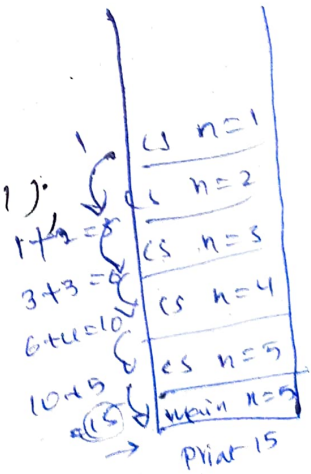
$f(n) = n + f(n-1)$

int sum (int n) {
S_{n-1} = sum(n-1)
$S_n = n + S_{n-1}$
return S_n.

— if (n==1)
return 1;

f(5) = 5 + f(4)
f(4 + f(3) 6
3 + f(2) 3
2 + f(1) 1
. . . .

public static int calcsum {
if (n==1)
return 1;
int snm1 = calcsum(n-1);
int sn = n + snm1;
return sn;

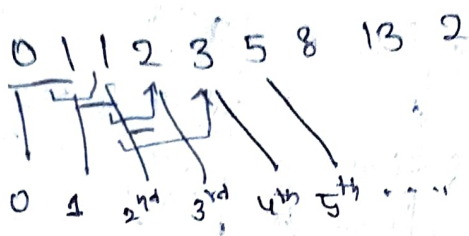


psvms {
int n=5
calcsum(5);

Time & space complexity = O(n)

Problem 5

print Nth fibonacci number.

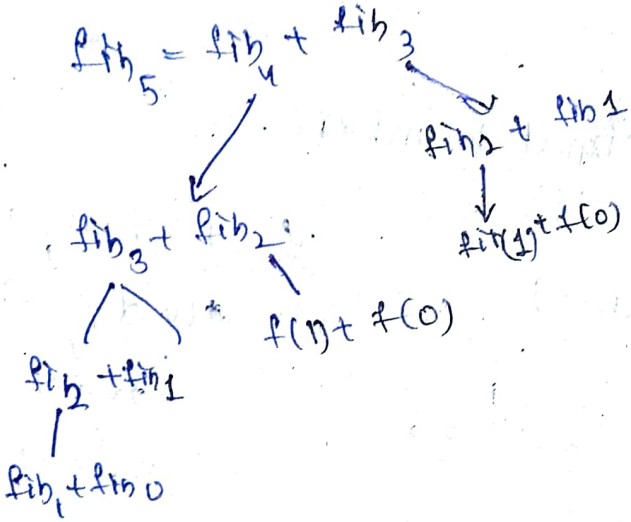


$fib_{n+2} = fib_n + fib_{n+1}$

$fib_n = fib_{n-1} + fib_{n-2}$

$fib_5 = fib_4 + fib_3$

$fib_4 \rightarrow fib_3 + fib_2$



Base case = 0, 1

fib(0) = 0

fib(1) = 1

Code

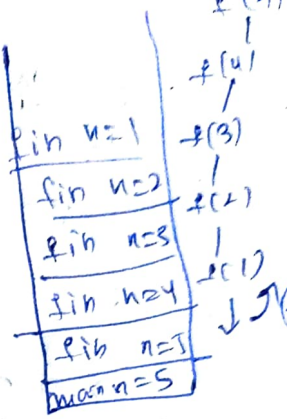
```
fib(int n) {
    if (n=0) -> 0
    if (n=1) -> 1
}  Base cases.
```

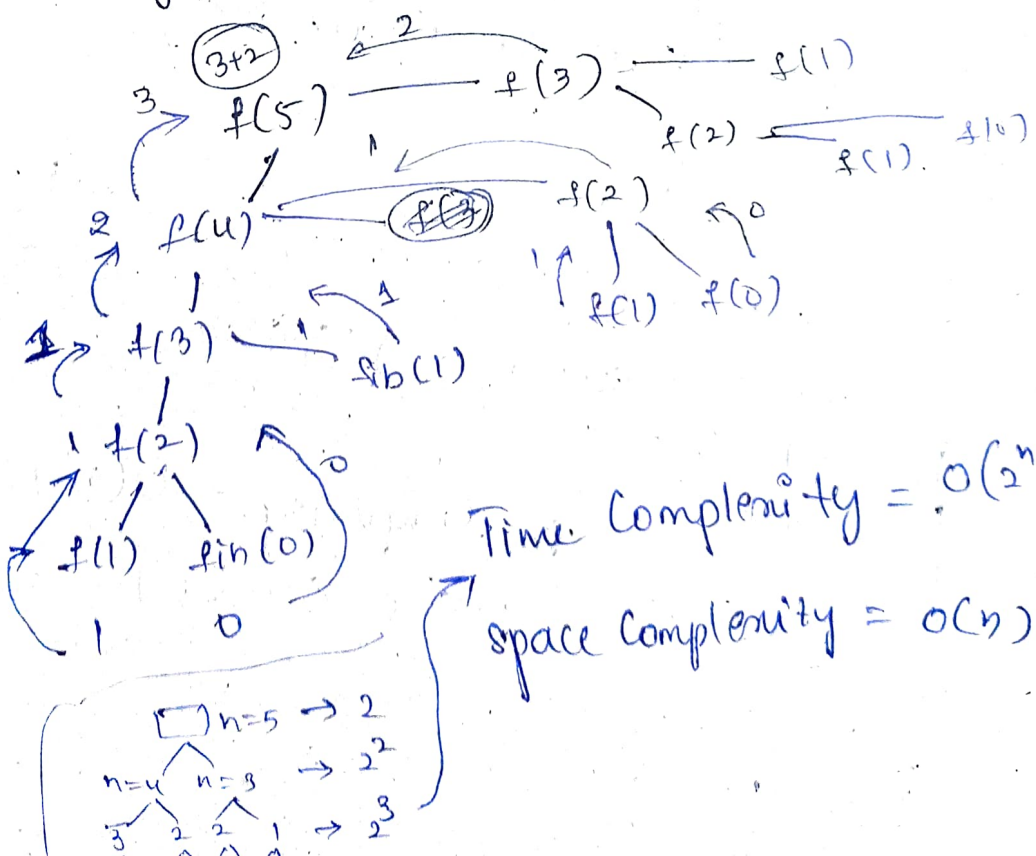
$fib_{N-1} = fib(n-1)$

$fib_{N-2} = fib(n-2)$

$fib_N = fib_{N-1} + fib_{N-1} + fib_{N-2}$

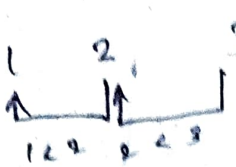
```
public static int fib(int n) {
    if n = 0 {
        return 0 ;
    if n=1 {
        return 1 ;.
    (or)
    if (n==0 || n==1){
        return n
    }
int fibnm1 = fib(n-1); // one reccursive cay
int fib(nm2 = fib(n-2); // 2nd recurvise call
int fibn = fib(n-1) + fib(n-2)
return fibn;
ps vms A[] {
    int n=5
    syso (fib.(n));
```



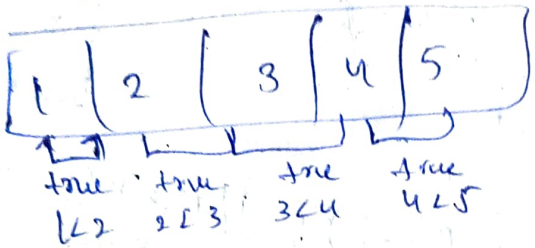


Q-6

Check if a given array is sorted or not (ascending)



sorted



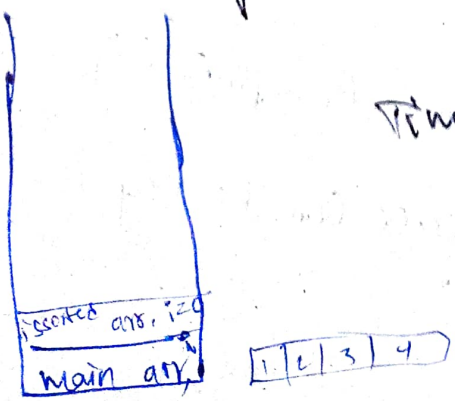

```
isSorted(int arr[], int i)
{
arr[i] <= arr[i+1]
isSorted(arr, i+1)
```

```
public static boolean isSorted(int arr[], int i)
{
    if (i == arr.length - 1)
    {
        return true;
    }
    if (arr[i] > arr[i+1])
    {
        return "not sorted" false;
    }
    return isSorted(arr, i+1);
}
```

```
psvms
{
    int arr[] = {1, 2, 3, 4, 4};
    sysout(isSorted(arr, 0)) // first call
}
```

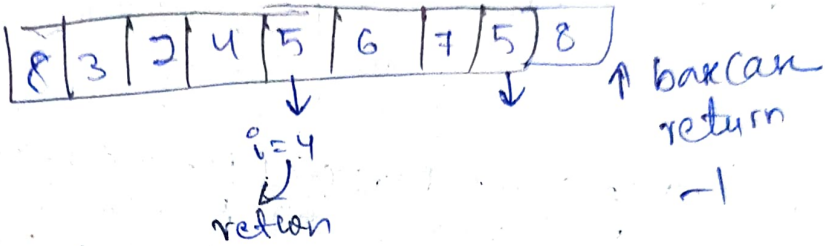
Time Complexity = O(n)

S.C = O(n)



problem - 7

write a fcnct to fnd occurence of a element in an array.

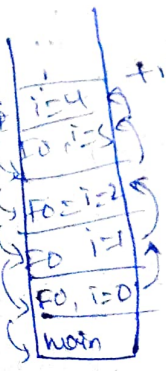


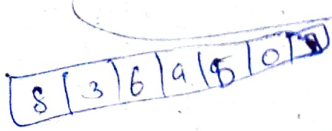

```
first occurance (arr, i)
    arr[i] == key
        return i
    return FO (arr, i+1)
```

code

```
public static int firstoccurance (arr[], int key, int i)
    if (i == arr.length); rtn -1;
    if (arr[i] == key) {
        return i;
    return, firstocurre (arr, key, i+1);
```

```
psvms A()
{
int arr[] = { 8, 3, 6, 9, 5 ... 5, 3}
sout (occurnce (arr, key, 0));
```



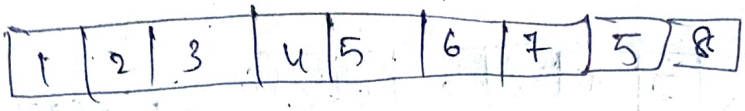


worst complexity = O(n)

space complexity = O(1)

Problem - 8

WAF to print the last occurence of an element-




```
public static int lastOcc(int arr[], key, i) {
    if (i == arr.length); { return -1;
    int isFound = lastOccure(arr, key, i+1)

    if (isfound != -1 && arr[i] == key)
        return isfound;
    }
    // check with self

    (if (arr[i] == key) {
        return i;)
    }
    return isfound;
```

```
psvmSA() {
    int arr[] = {2, 4, 6, 8, 5, 6, 7, 5}
    Sys(LastOccure(arr, key, 0);

public static int LO(int arr[], key, i) {
//Base case.  if (i == arr.length) {
                return -1;
    int isfound = Occ(arr, key, i+1);
    if (isfound == -1 && arr[i] == key) {
                return i;}

    return isfound;
```

problem 9.

print x^n for $2^{10} = 1024$ $\quad x^n = x \times x^{n-1}$

$x \times x^{n-2}$

$x \times x \times x^{n-3}$

$pow(x, n) = x * pow(x, n-1)$

```
pow(x, n) {
    n == 1 → x*n1
    return x * pow(x, n-1)
}
```

```
public static int power(int x, int n)
{
    if (n == 0) {
        return 1;
    }
    int nm1 = pow(x, nm1);
    int pn = x * nm1;
    return pn; || or return x * pow(x, n-1)
}

psvm sA(?) {
    sysO(pow(2, 10));
```

Base case = 0

$x^n = 2^0 = 1$

x*n 1

pow x=2	n=0
pow x=2	n=1
pow x=2	n=2
pow x=2	n=3
Base case. pow x=2	n=4
pow x=2	5
main x, n	2 5

$x^n = 2 \times 1 = 2$

$x^n = 2 \times 2 = 4$

$x^n = 4 \times 2 = 8$

$x^n = 8 \times 2 = 16$

$x^n = 16 \times 2 = 32$

Time complexity:-

$O(n)$

not optimised

Problem 10

print x^n in $O(\log n)$

even = $x^{n/2} \times x^{n/2}$

odd = $x \times x^{n/2} \times x^{n/2}$

$x^n \ (2^{10}) = x^{n/2} \times x^{n/2}$

case 1 (even) = $(2^5) \times 2^{10/2}$

case 2 (odd) = $x^n (2^5) \times x \times x^{n/2} \times x^{n/2}$

$2 \times 2^2 \times 2^2 = 2^{1+2+2}$.

$2^{10} = 2^5 \times 2^5$

$2 \times 2^2 \times 2^2$ | $2 \times 2^2 \times 2^2$

2×2 2×2 | 2×2 2×2.

$O(\log n)$

$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$

$\log_2 n = k$

Code

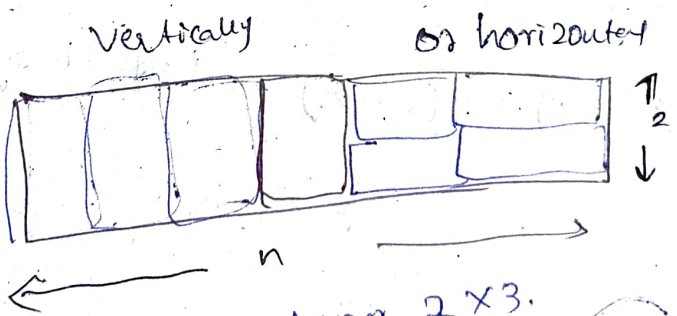
```
public static int optimizedpow(int a, int n){
    if (n==0){
        return 1;
    int halfpow = opt(a,n/2);
int halfpow = optimizedpow(a,n/2) * optimizedpow(a,n/2);

    if (n%2!=0)
        halfpow = a x halfpow;

    return halfpow;

psvm(){
    int a=2, int n=5.
        syso(optimizedpow(a,n));
```

```
if (n==0) {
    return 1;
int halfpow = opt(a, n/2);
int optimpowsq = halfpow x halfpow;
    if (n%2 != 0)
    halfpow = a x halfpow;
    return halfpow;
```

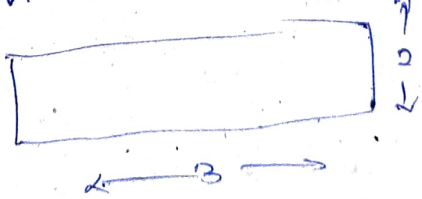
Problem 11 (Amazon)

Tiling problem:

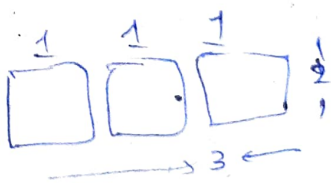
Given a "2x2" board and tiles of size "2x1", count the number of ways to file the given board using the 2x1 tiles. (A tile can be placed either horizontaly or vertically)



Ex n=3 floor 2x3.

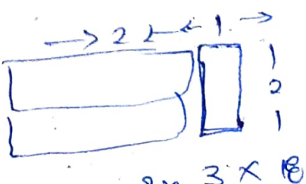


(c1)



2x3x1

(c2)



2x3x1

(c3)

2x3x1

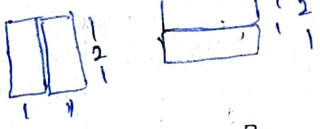
for n=3 way = 3

① base case
② work
③ call innerf(n)

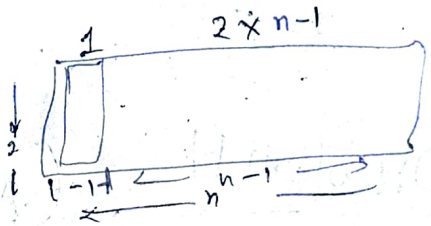
{ if n=0 && n<
ways = 1 }

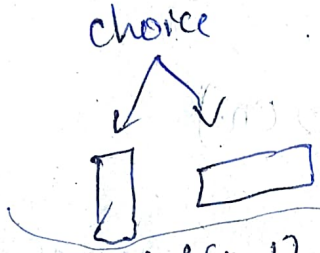
n=0 2x0 ways = 1 (No tile, tailighed.)

n=1 2x1  ways = 1

n=2 2x2  ways = 2.

n=3 3x2 ways = 3.

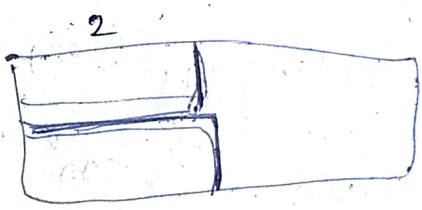




Q1:

fill 2xn-1 → f(n-1)

Vertical → f(n-1)



← n →

| 2 | (n-2)

Q2 f(n) = 2xn.

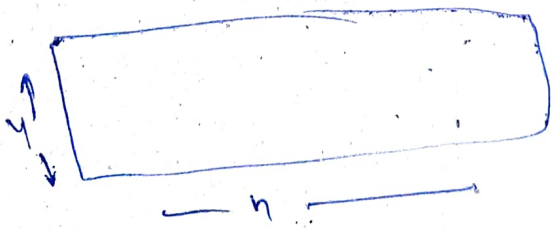
f(n-1) = 2x n-1

horizontal = f(n-2) = 2x n-2.

f(n-1) + f(n-2) = total ways

---

for uxn.



Code

```
public static int tilingprob(int n) {  // floor size 2xn.
    // base case
    if (n==0 || n==1) {
        return 1;
    }
    // work
    // vertical choice
    int verticalTiles = tilingprob(n-1);  (fnm1)
    // horizontal choice.
    int horizontalTiles = tilingprob(n-2);  fnm2
    int totways = fnm1 + fnm2;
    return totalways;
}
psvms A {
    syso(tiling prob(3));
```

Problem 12 Google Amazon.

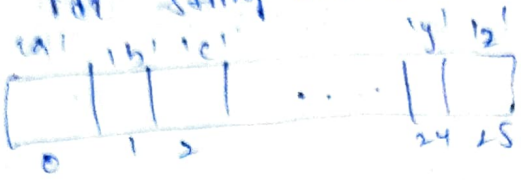
Remove Duplicates in a string.

('a', '-', '2', 'A', '@', '1' comes in hashset)

"appnacollege"
"empty string"
"apncoleg"

new string (sb)

idx string index.



map(boolean) 26

'a' - 'z'

'a' - 'a' = 0

'b' - 'a' = 1

'c' - 'a' = 2

'd' - 'a' = 3

index = char - 'a' (current)
(map)

① Base case idx = 0 to n (str length)

② work to do char → present in map ✓ (or) ✗ → new str + add

③ idx → idx+1

f(idx) → f(idx+1);

code

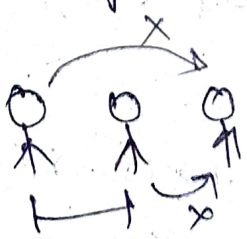
```
pubic static void remove duplicate (String str, int idx
                                    StringBuilder sb, boolean
                                    map)
{
if (idx == str.length()){
    sysO.pln(sb);
    return;
}
// work to do.
char curchar = str.charAt(idx);
if( map[curchar - 'a'] == true){
        // duplicate.
    remove duplicate( str, id+1, sb, map);
}
```

```
else {
    curr map[currchar - 'a'] = true;
    removeduplicate(str, idx+1, newstr.append(currchar), map);
  }
}

psvmsA() {
    String str = "appnacollege";
    remove duplicates(str, 0, newStringBuilder(""), newboolean[26]);
}
```

Problem 13 (Goldman sachs)

Friends pairing problem

Given n friends, each one can remain single or can be paired up with some other friends. Each friend can be paired only once. Find out the total number of ways in which friends can remain single or can be paired up



n=1 (single) ways = 1

n=2 a, b with others ways = 2

(a, b) both pair

n=3 a b c , [a, b, c]

{(ab), c}, {a(b,c)}, {b(a,c)}

ways = 4

① Base case n=1 ways = 1 | n=1 || n=2
n=2 ways = 2 | ways = n

② work to do.. choice level — permutation and combination

[n] → single, pair

single → □ (n-1)

pair → □□ (n-2)

(n-1) (n-2)

pair choices × 1 pair ways.

totways = f(n-1) + (n-1) × f(n-2)

Code:-

```
public static int friendspairing (int n) {
    // base case
    if (n==1 || n==2) { return n;

    // choice
    // single.
    int fnm1 = friendspairing (n-1);

    // pair
    int fnm2 = friendspairing (n-2);
    int pairways = (n-1) * f(n-2);
    int totways = fnm1 + pairways.
    return totoways;
```

psvm str []
ISYSO friendpairing (3)

(or)

```
p s v int friendspairing (int n)
    if (n==1 || n==2) {
        return n;
    }
return friendspairing(n-1) + (n-1) * friendspairing(n-2);
psvm sa[]
syso(friendspairing(3));
```

problem 14 (paytm)

Binary strings problem

print all binary strings of size N without consecutive ones.

01010 !! invalid.
↳ consecutive 1 → one after 1

010101 → ✓ valid string
0000 → valid string

1) Base case.

n=0 → " "
n=1 → "1" or "0";
n=2 → "00" "01" "10" ~~"11"~~ (consecutive 1's)

It is equals to n chairs

n size n=2 — — 0,1

Binary string

0 0 ✓
0 1
1 0
1 1 ✗
0

n=3

5 strings:

1) 0 0 0 — Lastplace = 0
2) 0 0 1 — LP = 1
3) 0 1 0 — LP = 1
4) 1 0 0
5) 1 0 1

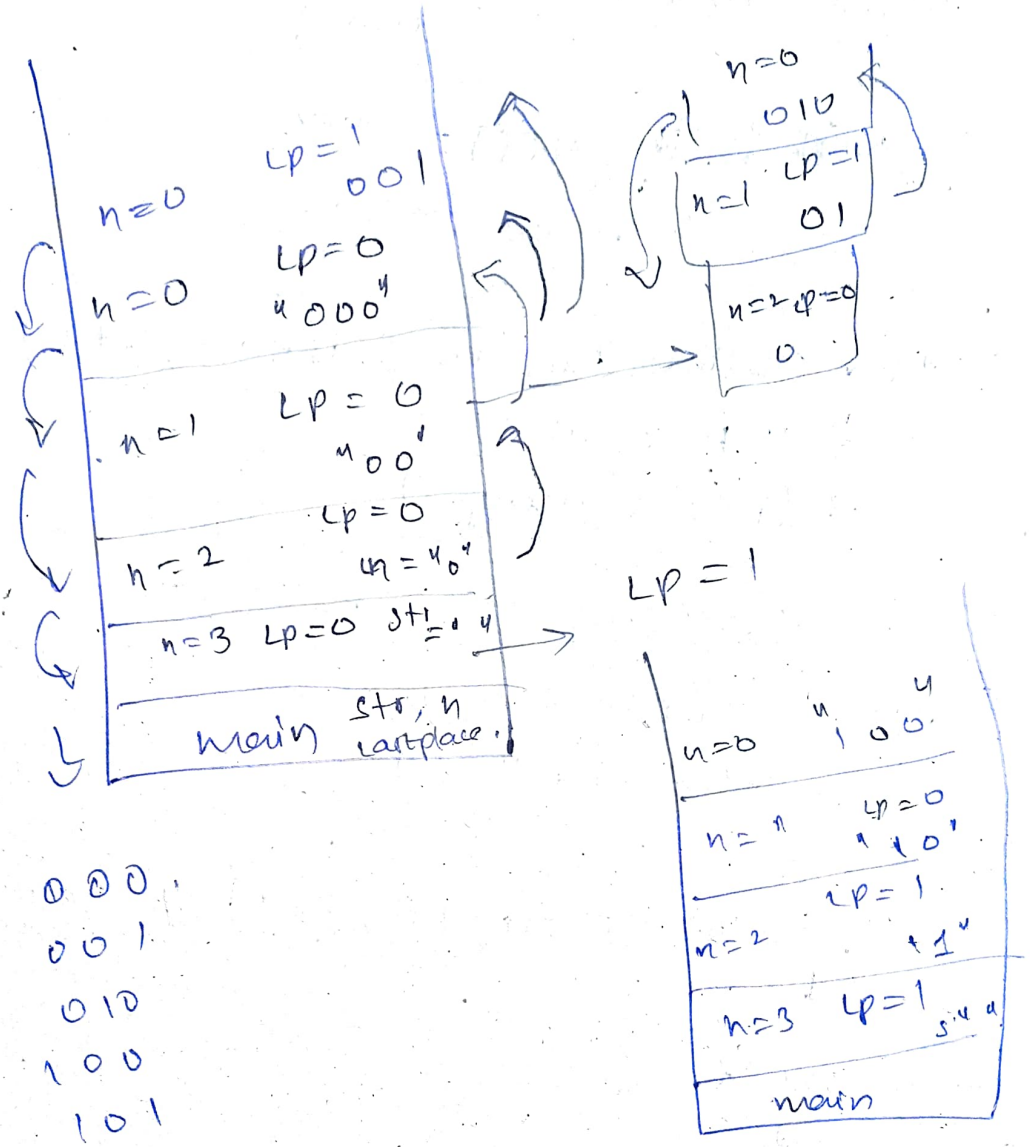
1) Base case n=0 "empty string"
2) work = Lastplace + 0 & 1
f(n), f(n-1)

Code:-

```
public static void printBinStrings(int n, int last_place, String str) {
    if (n == 0) {
        sysO (str);
        return;
    }
    // work
    if (last place == 0) {
        str += "0";
        // str.append("0"); str + "0";
        printBinstr(n-1, 0, str.append("0"));
        printBinstr(n-1, 1, str.append("1"));
                                    str + "1";
    }
    else {
        printBinst(n-1, 0, str.apend("0"));
```

(or)

```
printBinStrg(n-1, 0, str.append("0"));
    if (Last place == 0) {
        printstrings(n-1, 1, str.append("1"));
    }

ipSvmA() {
    printstrings(3, 0, new StringBuilder(""));
```



000
001
010
100
101

Q-1 for an integer array of size N you have to find all occurances of an element(key) and print them. (index)

Use recursive function to solve this.

ip = {3, 2, 4, 5, 6, 2, 7, 2, 2} key = 2

o/p = 1 5 7 8 (index numbers)

→ base case if i == arr.length
return;

work : if arr[i] == key
return arr[i].

```
PS v PrintIndex (int arr[], int key, int i)
  { // Base case.
  if (i == arr.length)
    {return}
  if (arr[i] == key)
    sysO (i + " ")
  }
  printIndex (arr, key, i+1) // recursion call
  }

PS vm st.
  {
  int arr[] = {1, 2, 2, 3, 2, 2};
  printarr (arr, 2, 0);
```

Question 2.

convert the number into string.

1234 / 10 → 10/1234

1972 → one nine seven two.

```
public class Main {
    static String digits[] = { "zero", "one", "two",
                               3, 4, 5, 6, 7, 8, 9 };

    public static void printdigits(int nbr) {
        if (nbr == 0)
            return;
        int lastdigit = number % 10;   // last digit = 4
        printdigits(number/10);        // 123
        sysO(digits[lastdigit] + " ");
                digits[4] = four.
    }
    psvm() {
        printdigits(1234);
```

2002

two zero zero two.

write a program to find the length of a string

eg:- "abacb";

length = 5.

1) base case. if i >= length
space
return;

for (int i=0; i<str

~~static string digits~~

public static ~~void~~ int length (String str ~~[]~~

if (str.length() == 0)

{ return 0; }

~~retu~~ return length (str.substring(1) + 1);

}

psvm SA() {

string ("abcd");

Question-4

s = "abacb"

output = 4 (a, ab, abc, ...)

4 substrings end with same element so reject

Accept it

ex: s = aba

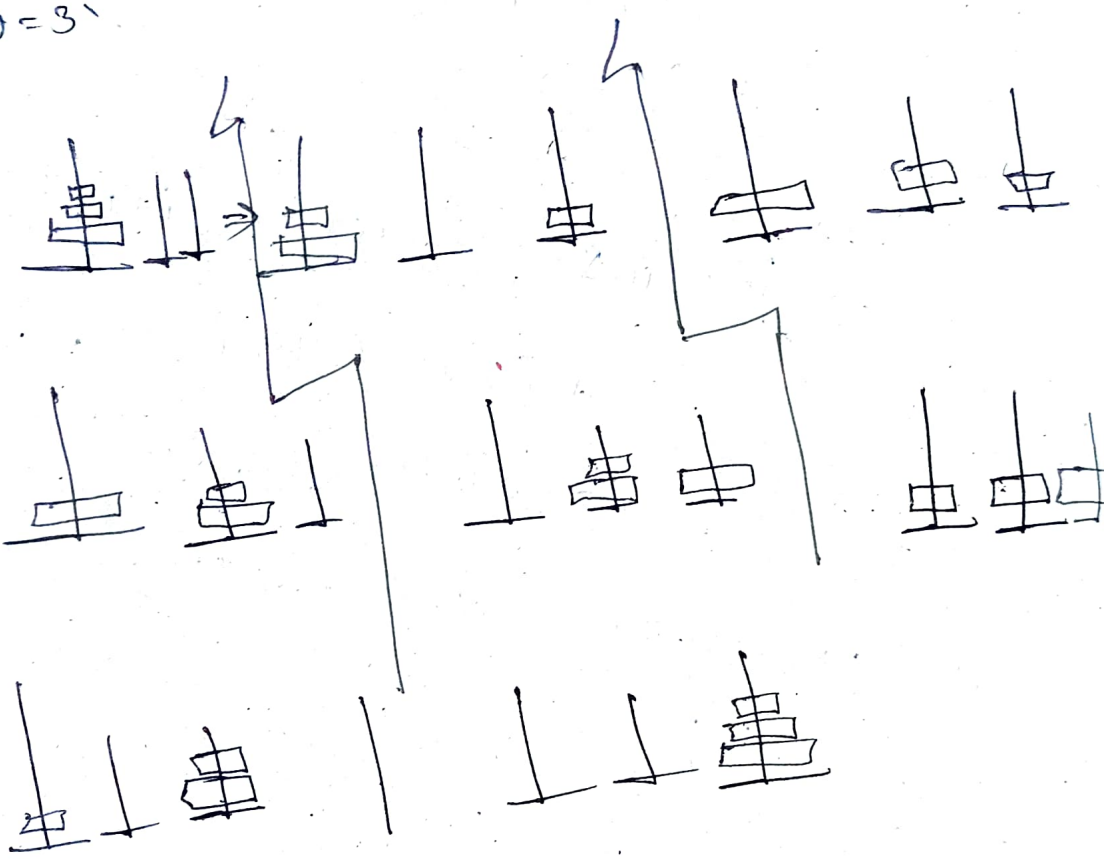
o/p = 4

substrings = a, b, a and aba.

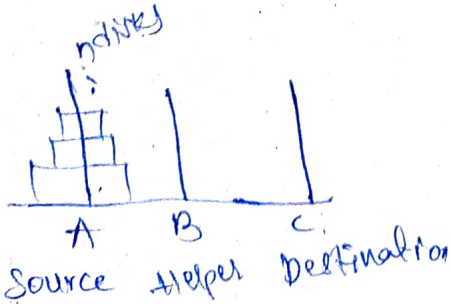
```
public class soln {
    static int countSubst(String str, int i, int j
                          , int n)
    {
        if (n == 1) {
            return 1;
        if (i <= 0) {
            return 0;
        int res = countSub(str, i+1, j, n-1)
                + (countSub(str, i, j-1, n-1)
                - countSub(str, i+1, j-1, n-2);
        if (str.charAt(i) == str.charAt(j)) {
            res++;
        return res;
    psvm sA {
        String str = "abc";
        int n = str.length;
        Sys.o(countSub(str, 0, n-1, n));
```

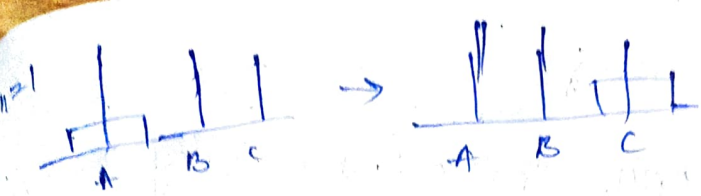
Tower of HANOI (Important)

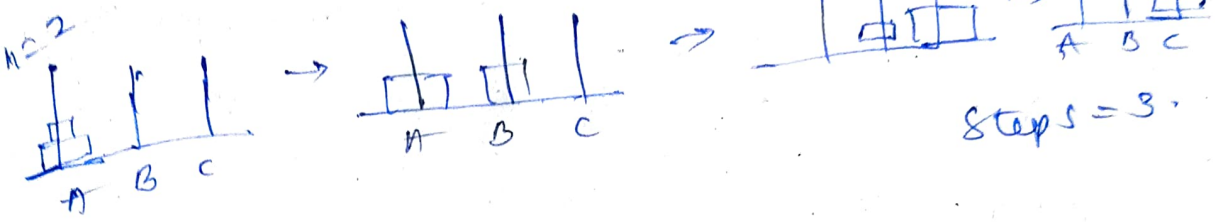
for N = 3

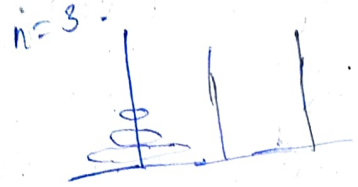


Rules 1) only one disk transfered in 1 step
2) Smaller disks are always ket on top of larger disks.

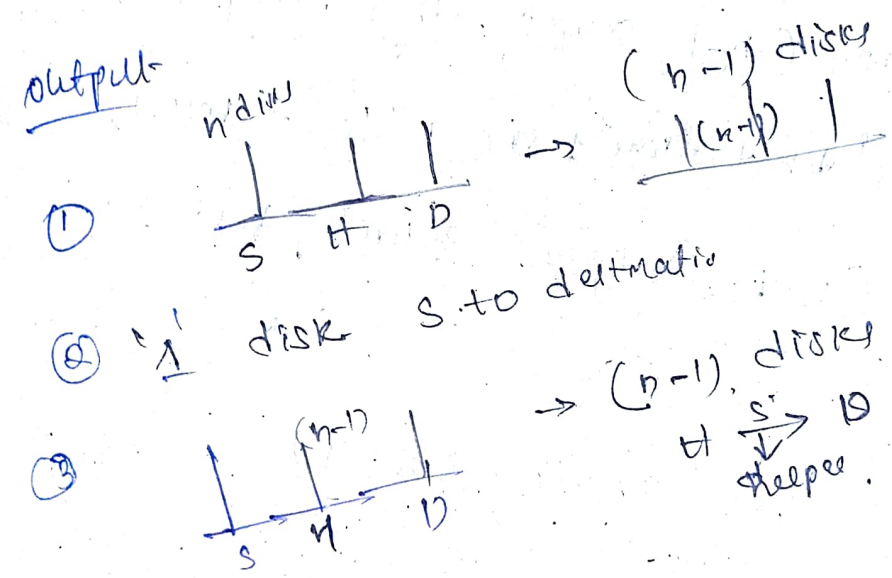








output-



Code:-

```
Public static void TowerofHanoi (int n, String source,
                                 String helper, String dest)

    // base case.
```

Code

```
public class recursion {
public static void TowerofHanoi(int n, String source,
                                String helper, String dest) {

    // Base case
    if (n == 1) {
        System.out.println("transfered" + n + "from" + src + "to" + dest);
        return;
    }

    towerofHanoi(n-1, source, dest, helper);
    System.out.println("transfered" + n + "from" + src + "to" + dest);
    towerofHanoi(n-1, helper, source, dest);
}

public static void main(String[] args) {
    int n = 3;
    towerofHanoi(n, "S", "H", "D");
}
}
```

S → D, S → H, D → H, S → D, H → S, H → D, S → D

Time Complexity $O(2^n - 1) \approx O(2^n)$

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

$T(n-1)$ $\quad 2T(n-2) + 1$

$2T(n-3) + 1 \quad \ldots \quad T(1) = 1$