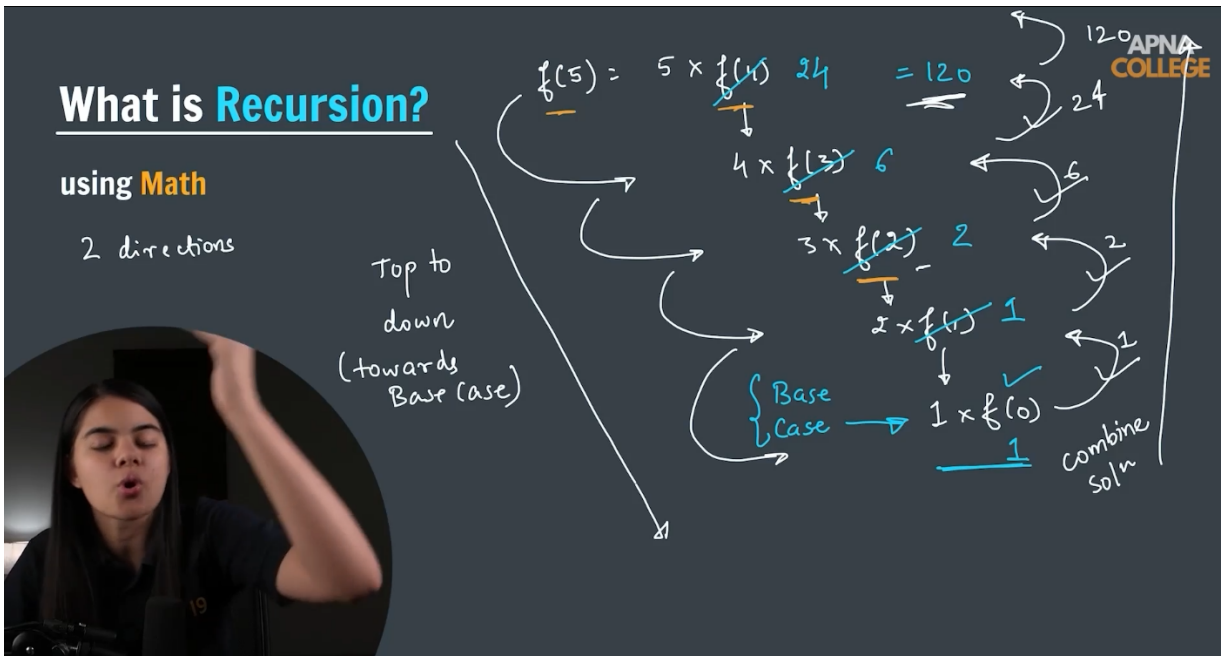
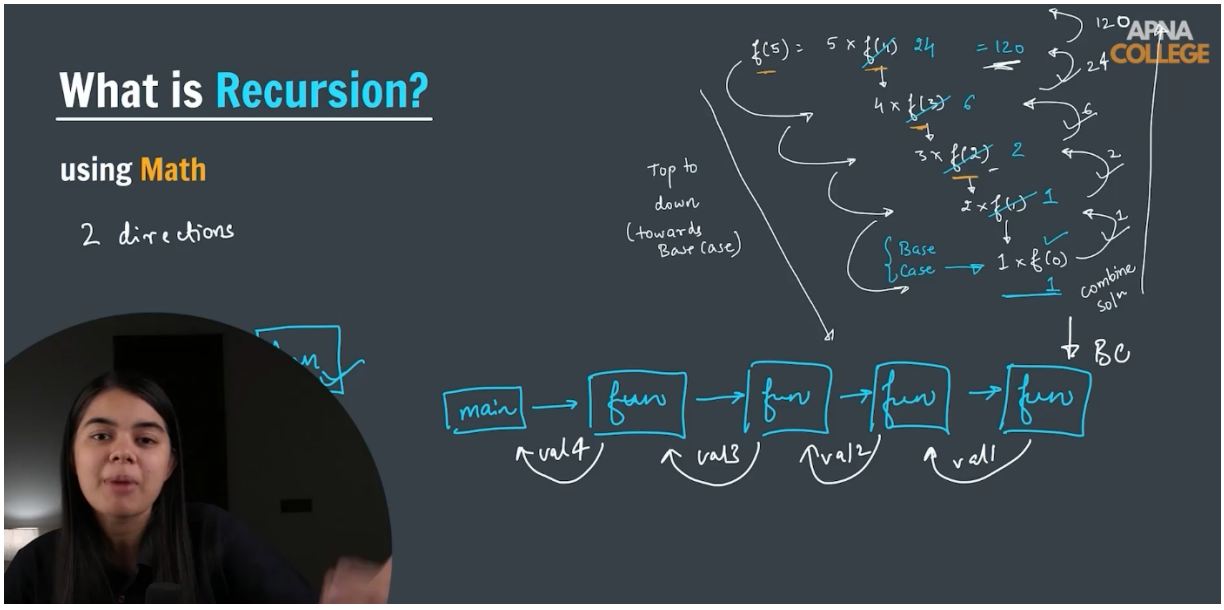
Recursion

- Recursive Functions are such functions which calls itself for smaller problems

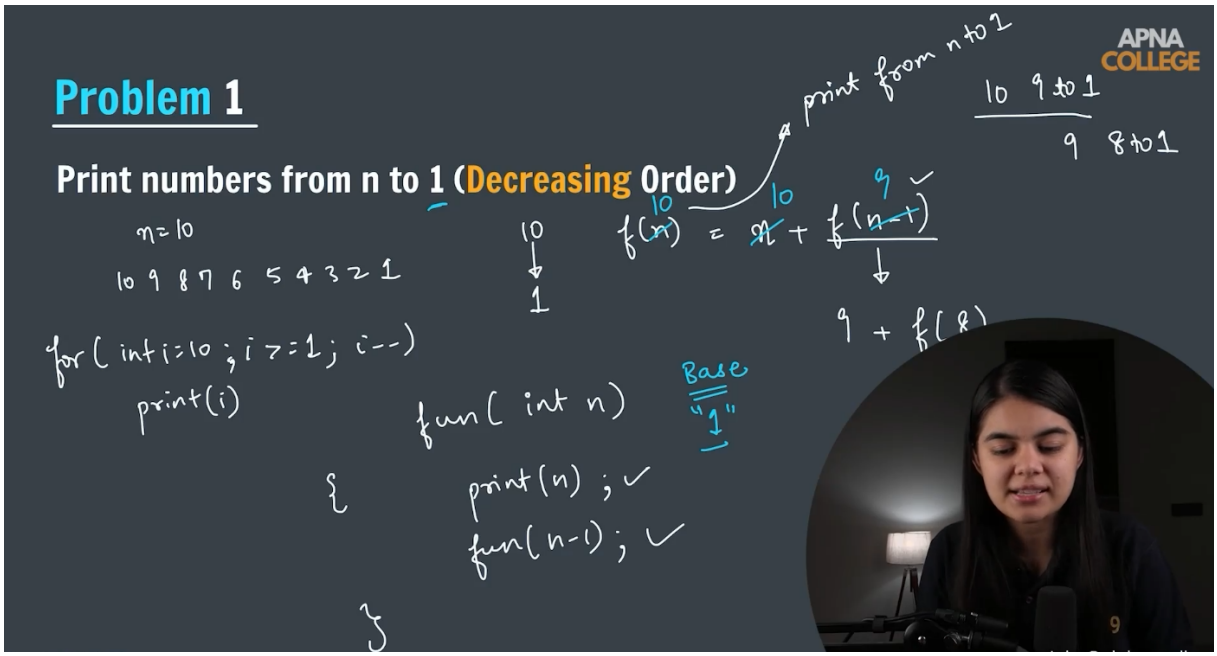
▼ What is Recursion?

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



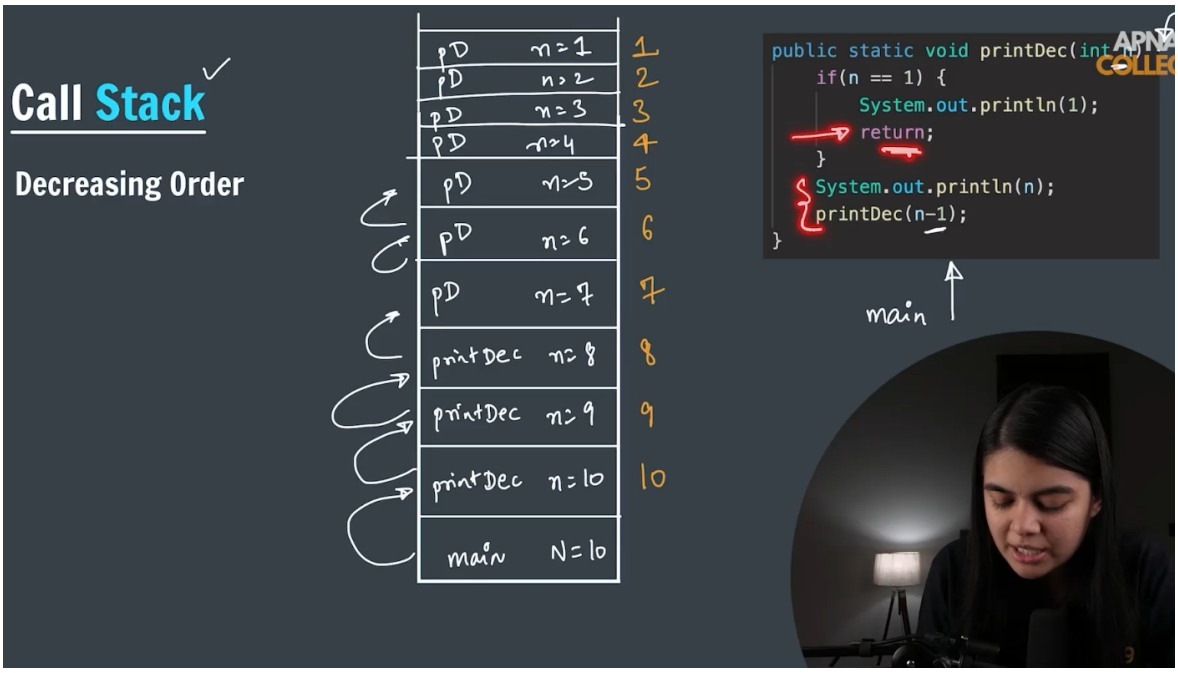


▼ Print numbers from n to 1 (Decreasing Order)




```
  package Recursion;

public class Print_decending_numbers{
    public static void printdec(int n){

        if (n == 1 ){           //Base condition in case of the last remaining number
            System.out.print(n);
            return;
        }

        System.out.print( n + " ");
        printdec(n-1);          //Calling the function again and leaving the remaining task to it to complete

    }
    public static void main (String Args[]){
        int n = 10;         //Input Value
        printdec(n);
    }

}
```

- Call Stack :



▼ Stack Overflow :

- It is a condition which occurs when a function (recursive function) is creating large number of small array or small numbers of very large array in which case our allotted stack of memory get full and is called as Stack Overflow
- A stack overflow is **a type of buffer overflow error that occurs when a computer program tries to use more memory space in the call stack than has been allocated to that stack**

▼ Print numbers from n to 1 (Increasing Order) :

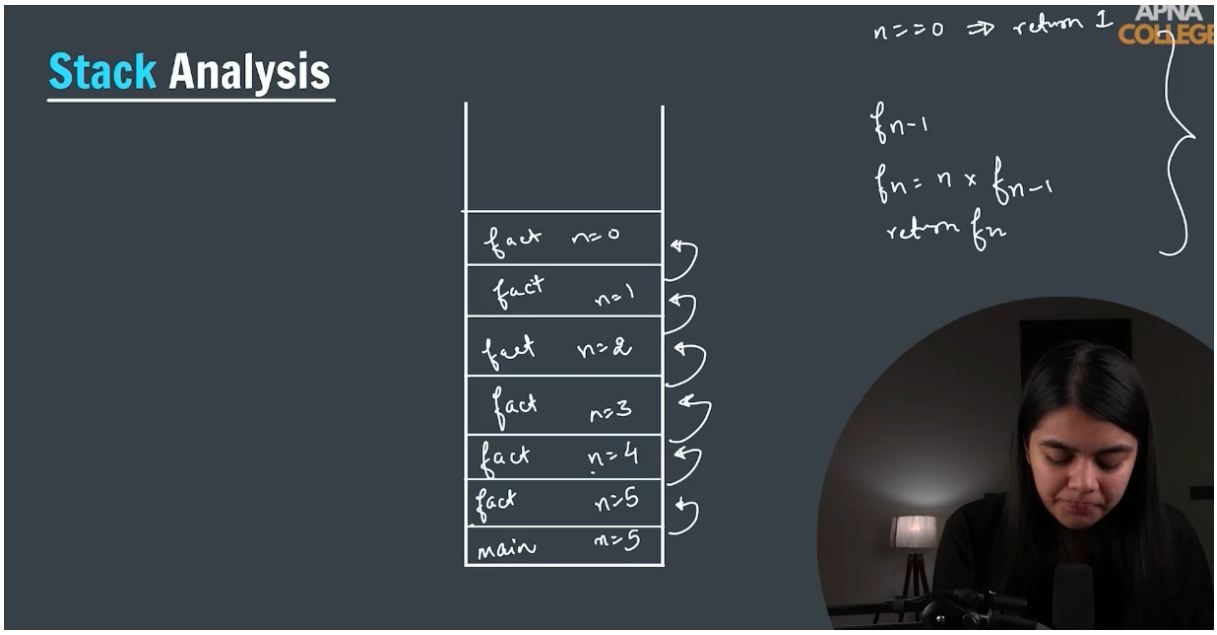
```
package Recursion;

public class Print_increasing_numbers {
    public static void printInc(int n){
        if (n == 1) {
            System.out.print( n + " ");
            return;
        }
        printInc(n-1);
        System.out.print( n + " ");
    }
    public static void main (String Args[]){
        int n = 10;
        printInc(n);
    }
}
```

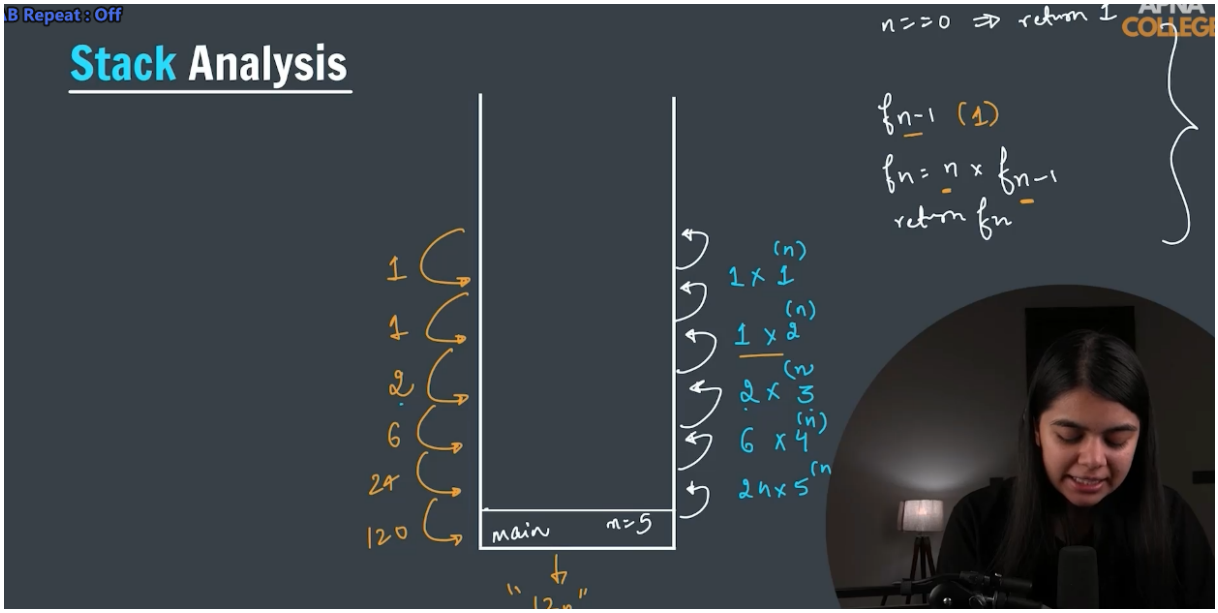
▼ Print Factorial of a number n :

- We will assume that one part of the equation would get automatically calculated and we just have to calculate

```
package Recursion;

public class Find_factorial_of_n {
    public static int fact (int n) {
        if (n==0) {     //Base case
            return 1;
        }
        int fnm1 = fact (n-1); //fnm = fact - 1
        int fn =n * fact(n-1);
        return fn;
    }
```

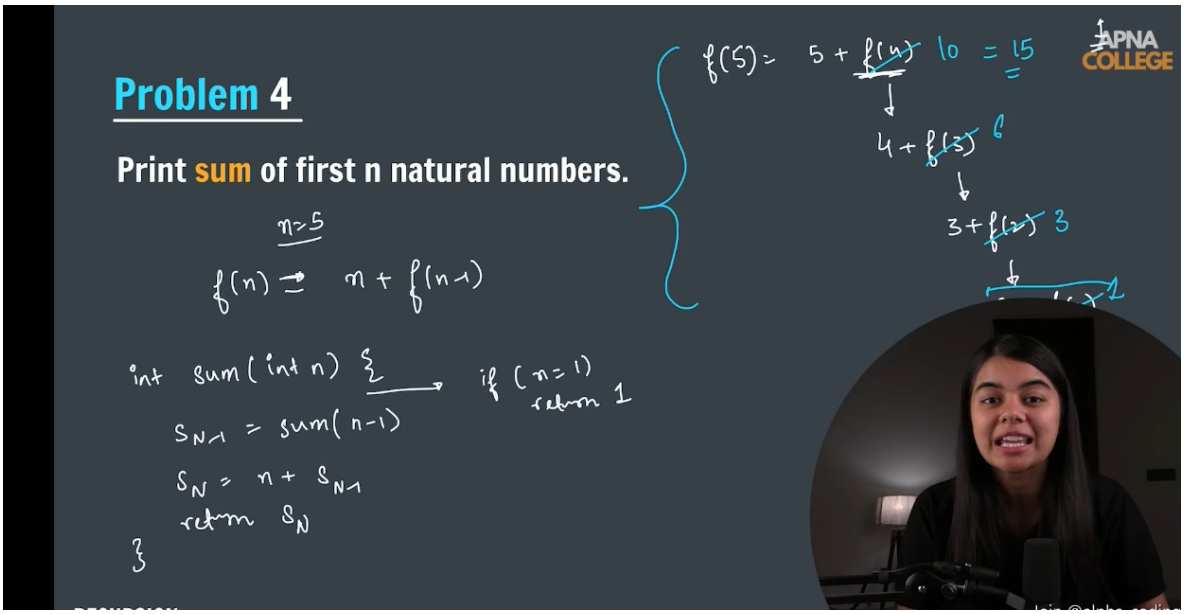
```
public static void main (String args[]){
    int n = 5;
    System.out.println(fact(n));
}
}
```





▼ Print sum of first n natural numbers.

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

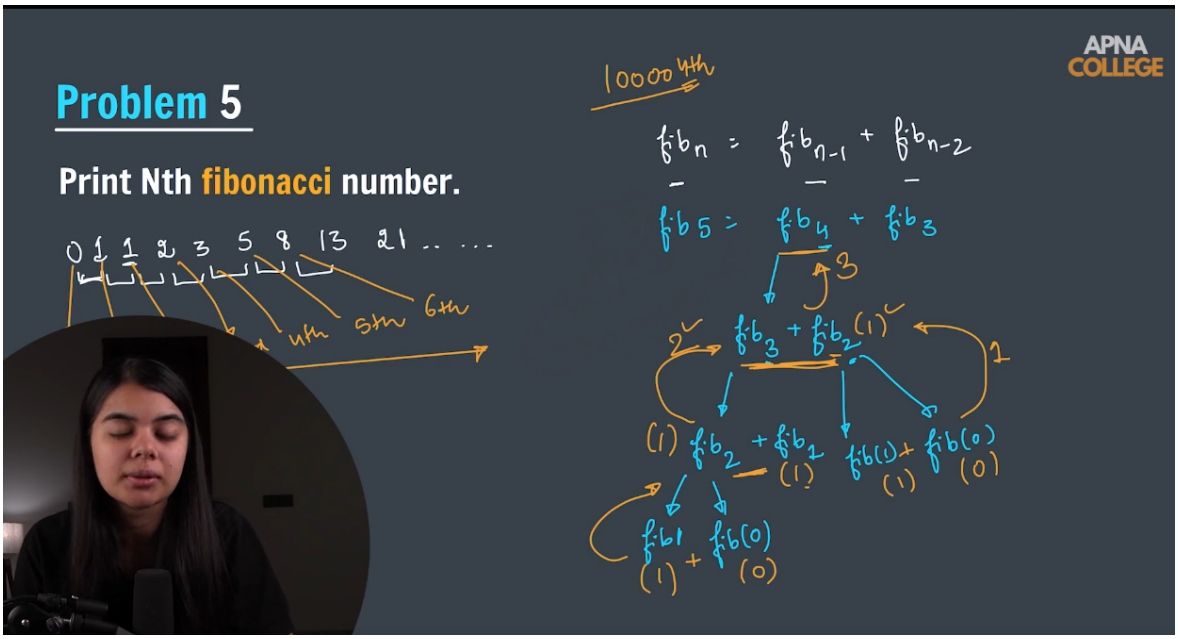


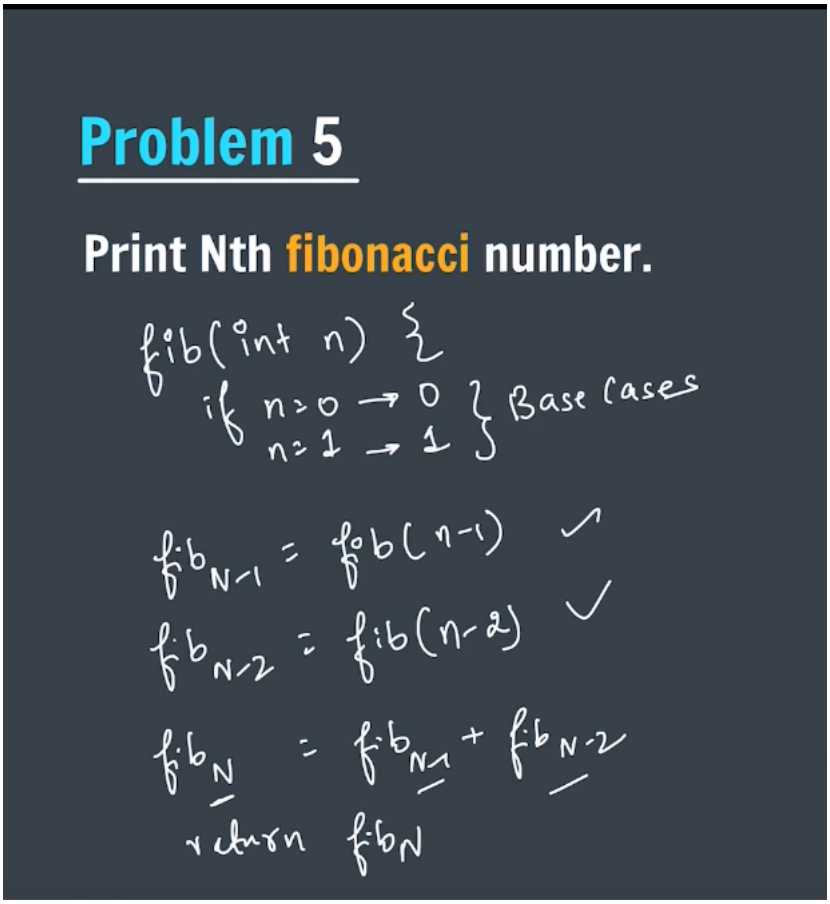

```
package Recursion;

public class Print_sum_of_first_n_natural_numbers {

    public static int clacSum (int n){
    if ( n==1){                // Base Case
        return 1;              //It is because we know that the sum of the 1 natural numbers would always be 1
    }
    int Snm1 = clacSum(n-1);
    int Sn = n + Snm1;
    return Sn;
    }
    public static void main (String args[]){
        int n = 5;
        System.out.println(clacSum(n));
    }
}
```

- Time Complexity : O (n)
- Space Complexity : O (n)

▼ Print Nth Fibonacci number.

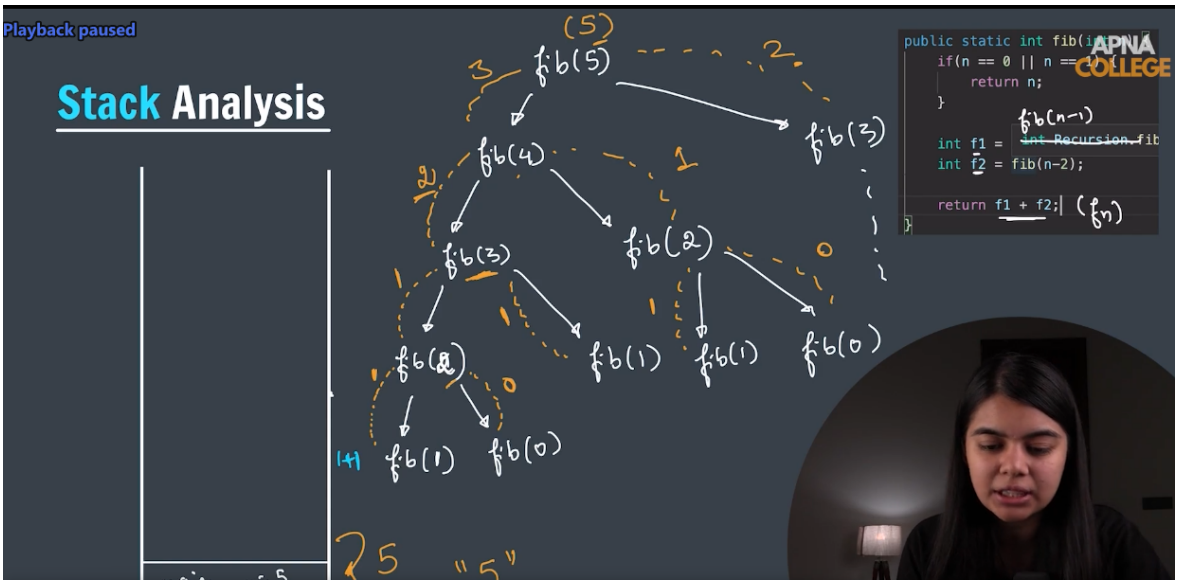
- A series of numbers in which each number (*Fibonacci number*) is the sum of the two preceding numbers. The simplest is the series 1, 1, 2, 3, 5, 8, etc.
- To find the sum of fib (n) we can have the sum as fib (n-1) + fib (n-2)






```
package Recursion;

public class Print_Nth_fibonacci_number {

    public static int fib(int n ){
```

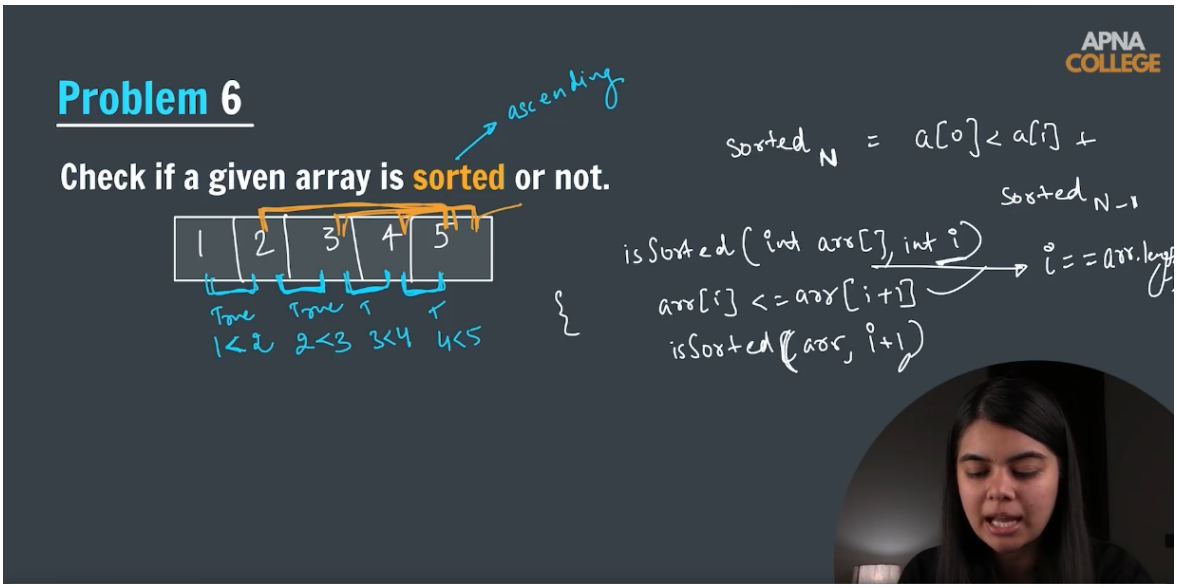
```
        if (n==0 || n ==1){
            return n ;
        }
        int fnm1 = fib(n-1);
        int fnm2 = fib(n-2);
        int fn = fnm2 - fnm1;

        return fn;

    }
    public static void main (String args[]){
        int n = 25;
        System.out.println(fib ( n ));
    }
}
```



- Time Complexity : O (n)
- Space Complexity : O (2^n)

▼ Check if a given array is sorted or not.

- Sorted N = a [0] < a [0] + Sorted N-1 (i.e. the remaining array is also sorted)
- Every time we would select the first two elements of the array and check weather they are sorted if yes then we mark it as true and move to the next part . Similarly we select the first two elements of the remaining array and repeat




```
package Recursion;

public class Check_if_a_given_array_is_sorted_or_not {

    public static boolean isSorted (int arr[],int i ){
        if (i == arr.length-1 ){            //i.e. i is equal to the last elment of the array then mark as true.
            return true;
        }

        if (arr[i] > arr[i+1]){
            return false;
        }
        return isSorted(arr, i+1);

        //This denotes that if at any point the value of i is greater than i+1 (in case of assending) Then the array is not sorted

    }
    public static void main (String args[]){
        int arr [] = {1,2,3,4};
        System.out.println(isSorted(arr, 0));
    }

}
```

▼ Problem 7 : Write a Function to find the first occurrence of an element in an array

- Checking from 0 index for the desired value and then moving forward
- If the key is not available then return -1

```
package Recursion;

public class Write_a_Function_to_find_the_first_occurrence_of_an_element_in_an_array {

    public static int firstOccurence(int arr[], int key, int i ){
        if (i == arr.length){
            return -1;
        }
        if (arr[i] ==  key){
            return i;
        }
        return firstOccurence(arr, key, i+1);
    }
    public static void main (String args[]){
        // int key = 5;
        int arr[] = {8,3,6,9,5,10,2,5,3};
        System.out.println(firstOccurence(arr, 5 , 0));
    }
}
```

▼ Problem 8 : Write a Function to find the last occurrence of an element in an array

```
package Recursion;

public class Write_a_Function_to_find_the_last_occurrence_of_an_element_in_an_array {

    public static int lastOccurence(int arr[], int key, int i ){
        if (i ==  arr.length){
            return -1;
        }

        int isFound = lastOccurence(arr, key, i+1);
        if (isFound == -1 && arr[i] == key){
            return i;
        }
        return isFound;
        }

    public static void main (String args[]){
        // int key = 5;
        int arr[] = {8,3,6,9,5,10,2,5,3};
        System.out.println(lastOccurence(arr, 5 , 0));
```

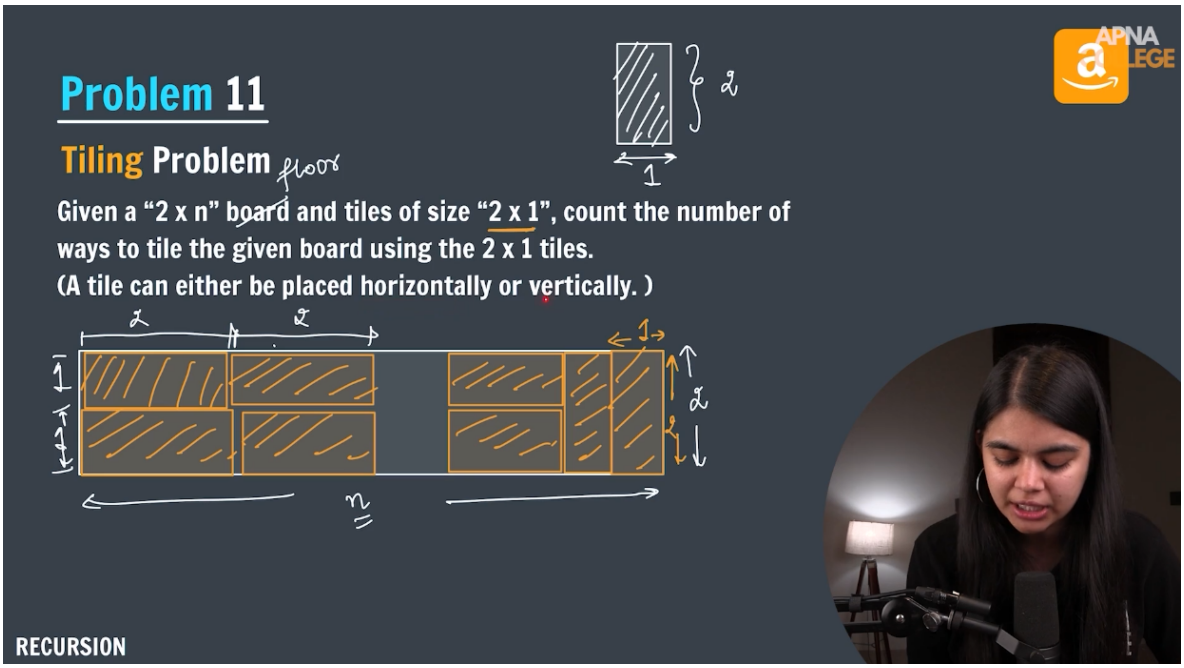
```
    }
}
```

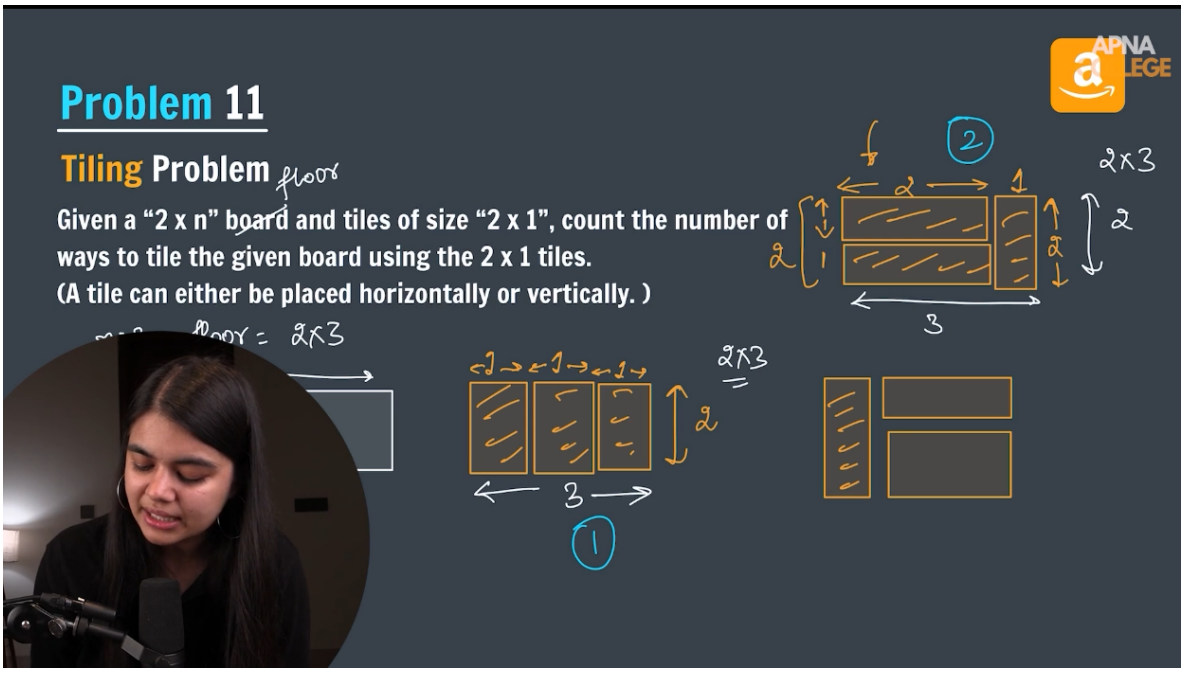
▼ Problem 9⭕ : Print x^n

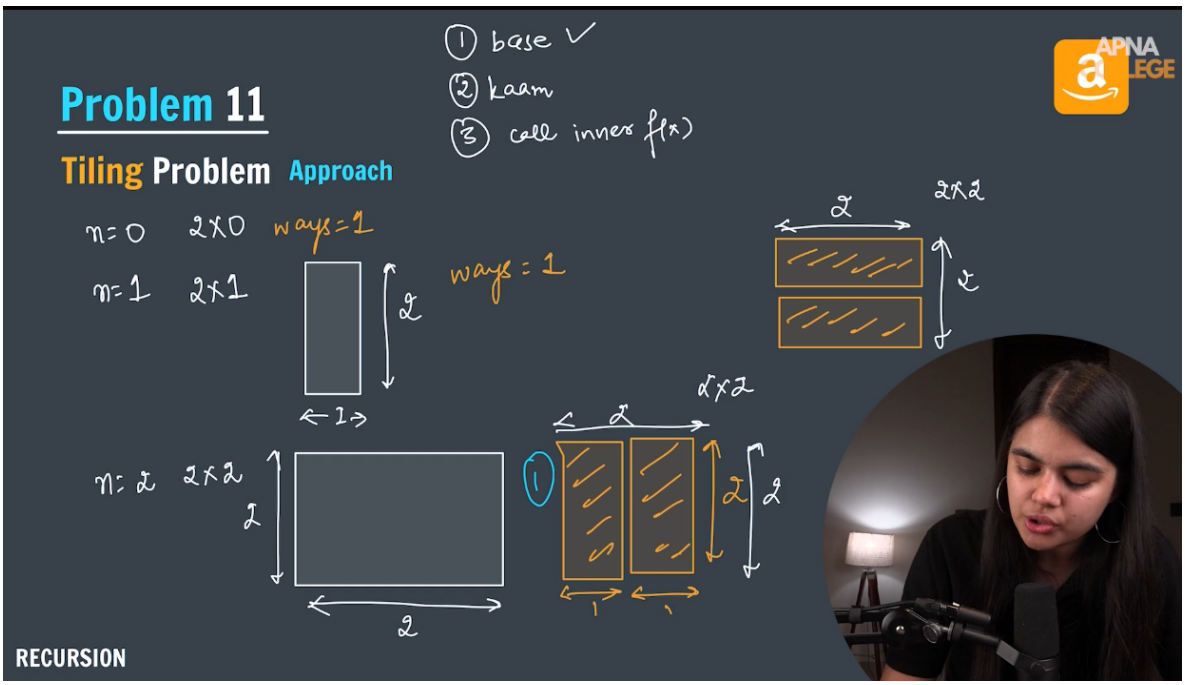
- pow (x, n) = x * pow (x, n-1)
- Skipped for current
- ⭕⭕

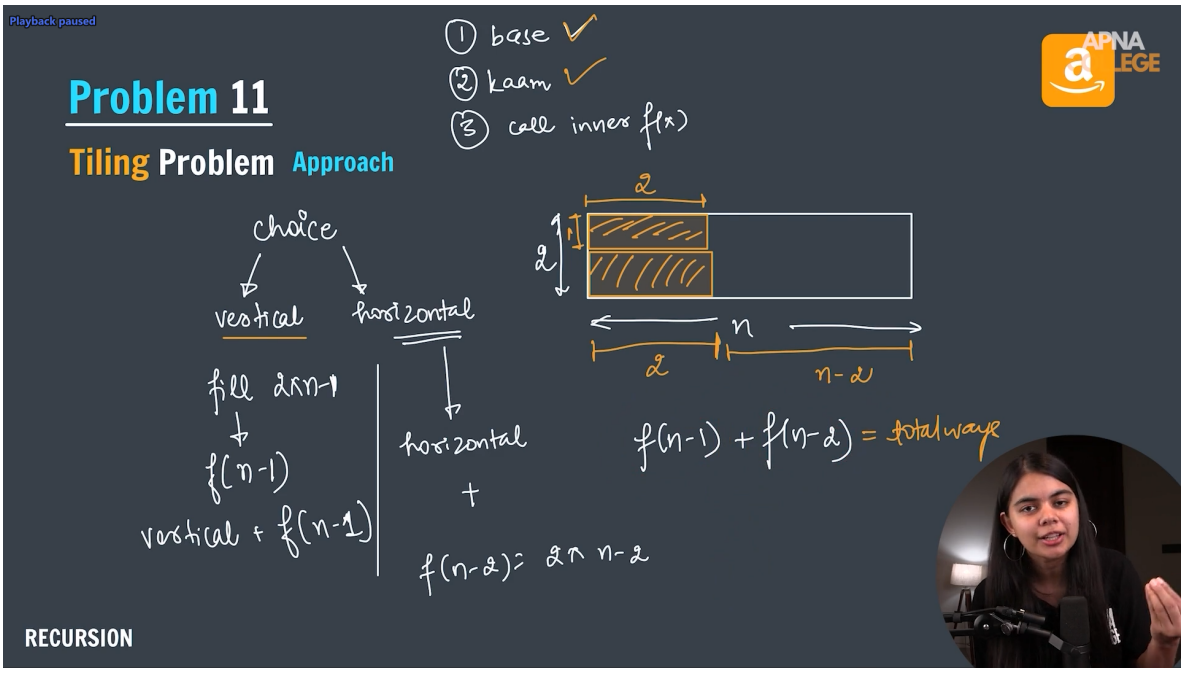
▼ Tilling Problem :

Given a "2 x n" board and tiles of size "2 x 1", count the number of ways to tile the given board using the 2 x 1 tiles. (A tile can either be placed horizontally or vertically.)



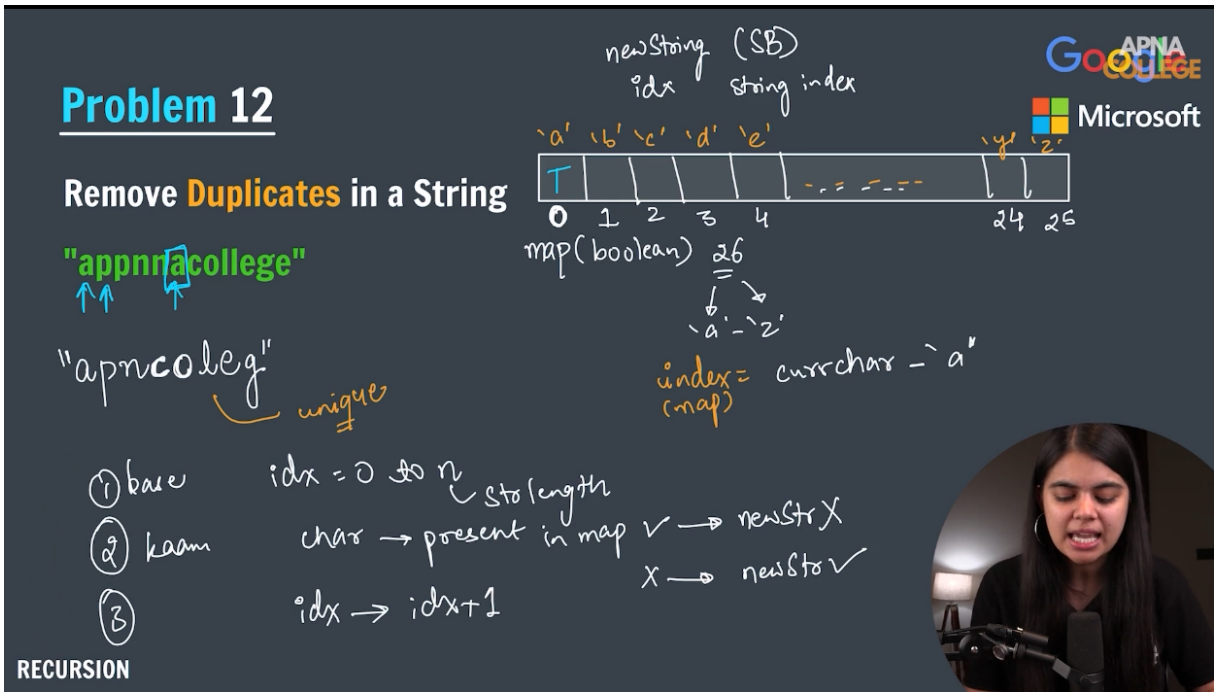

Problem 11
Tiling Problem
floor
Given a "2 x n" board and tiles of size "2 x 1", count the number of ways to tile the given board using the 2 x 1 tiles.
(A tile can either be placed horizontally or vertically.)
2x3
2
1
3


Problem 11
Tiling Problem Approach
1 base ✓
2 kaam
3 call inner f(x)
n=0 2x0 ways=1
n=1 2x1
ways=1
n=2 2x2
2x2
RECURSION



▼ Remove Duplicates in a String

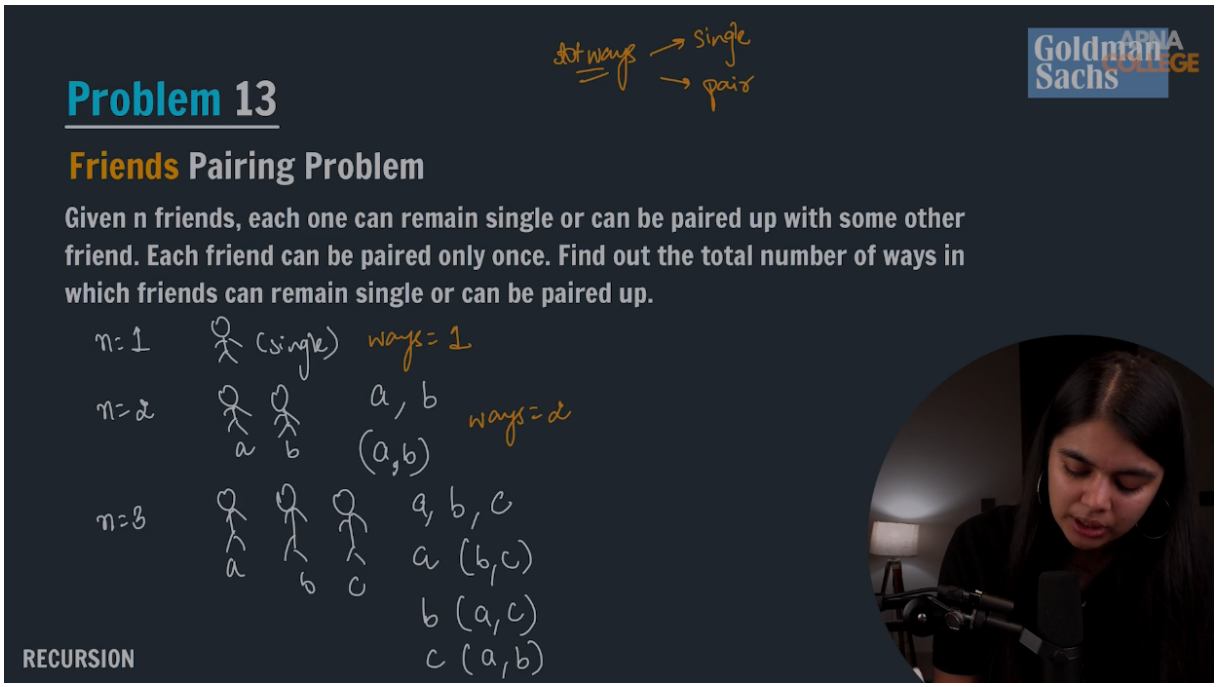
- With a condition that all characters are Small letters
- Remove duplicate for other conditions such as Capital Letters , Symbols and others can be solved using Hash Map
- “apnacollege” = apncoleg
- Create a new String using StringBuilder
- Create a int idx which would track the index value of the String
- Create a ‘map’ Array which would store True/False (Boolean Type) , the array would store letters form a, b, c, d upto z with index values of 0 to 25.
- If at any point any letter is found in the string , for eg ‘ a ’ the in the map array the index value corresponding to the letter i.e. ‘ 0 ’ in this case would be marked as true.
- Moving forward each letter would be compared to the map array and if for any letter the value is already true , then it would not be added.
- Type Conversion : Using it we can convert characters into integers
  - i.e. ‘ a ’ - ‘ a ’ = 0
  - ‘ b ’ - ‘ a ’ = 1
  - Current Char - ‘ a ‘



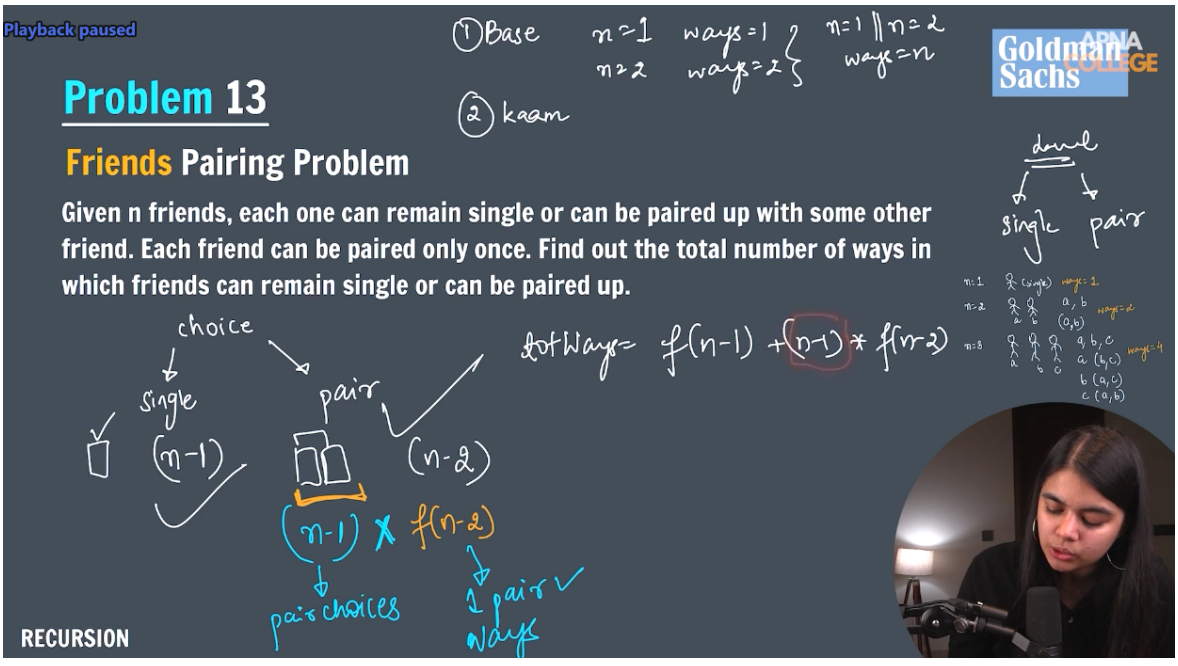

```
package Recursion;

public class Remove_Duplicates_in_a_String {
    public static void removeDuplicates(String str , int idx , StringBuilder newString , boolean map[]){

        //Base Case
        if (idx == str.length()){
            System.out.println(newString);
            return;
        }
        //Checking if the character is a duplicate or not by checking and assigning true to the characters in the map array and skippin

        char currChar = str.charAt(idx);

        if( map[currChar-'a'] == true ){
            //If the value is already true then the char is a duplicate
            removeDuplicates(str, idx + 1, newString, map);

        } else {

            map[currChar - 'a' ] = true;
            removeDuplicates(str, idx + 1, newString.append(currChar), map);
        }
    }
    public static void main (String args[]){

        String str = "apnacollegeisthebest";
        removeDuplicates(str,0,new StringBuilder(""), new boolean[26]);
    }
}
```

▼ Friend Pairing Problem :

Given n friends, each one can remain single or can be paired up with some other friend. 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.



- Base Case : no. of friends = n = 1,2 = no. of ways to pair them , that would also be 1 , 2
- Choice at every level = Either to be alone 1 , Either be in pair 2

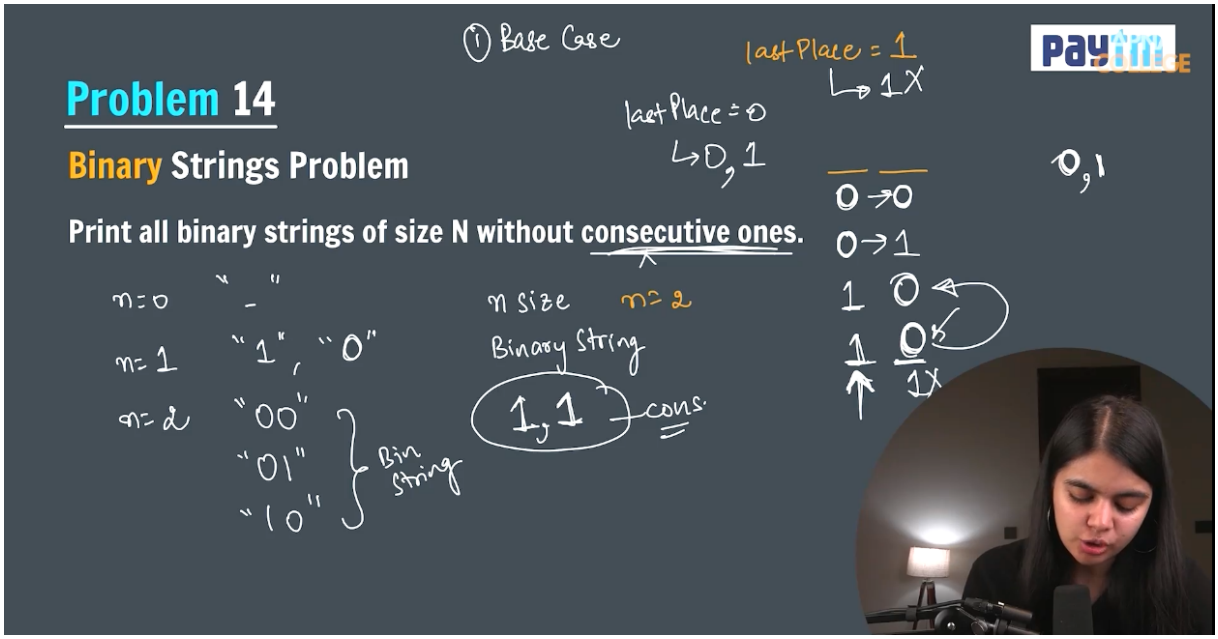



```
package Recursion;

public class Friend_Pairing_Problem {
    public static int friends_pairing(int n){
        //BaseCase
        if (n==1 || n==2){
            return n;
        }
```

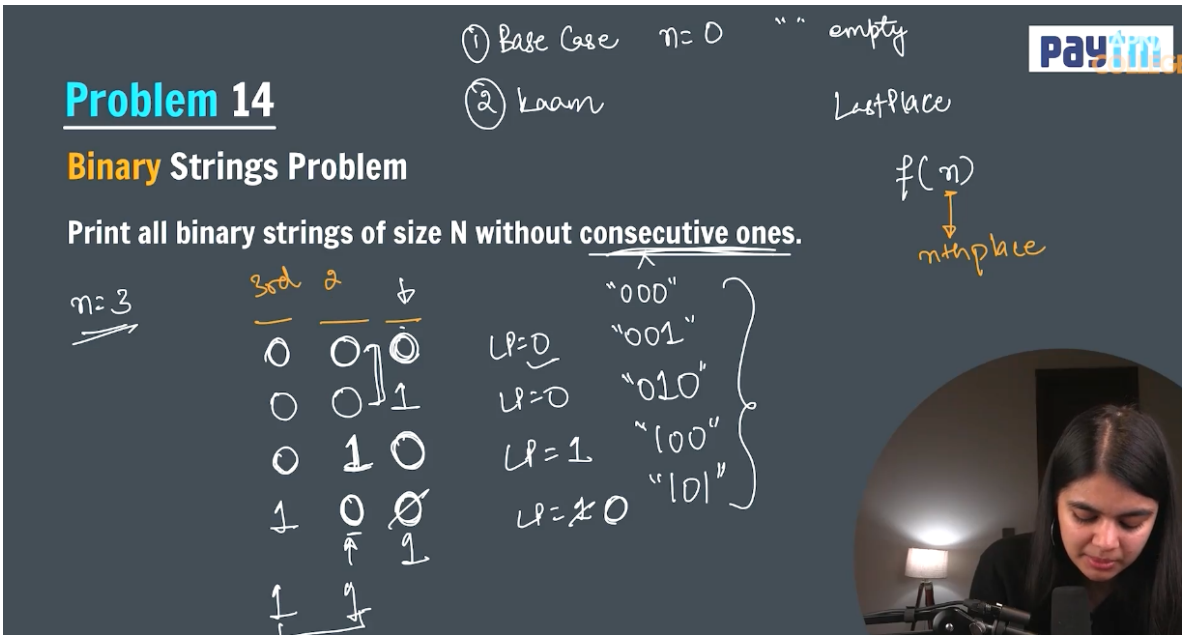
```
        //Choice
        //Single
        int fnm1 = friends_pairing(n-1);

        //Pair
        int fnm2 = friends_pairing(n-2);
        int pairWays = (n-1) * fnm2;

        //Total Ways
        int totways = fnm1 + pairWays;
        return totways;
    }
    public static void main(String args[]){
        System.out.println(friends_pairing(3));
    }
}
```

▼ Problem 14 : Binary Strings Problem :

Print all binary strings of size N without consecutive ones.



Establish a last place variable using which we can see if the current last digit was 0 or 1 . If it is 1 then our only option is to place 0 but if last digit is 0 then we can place either 0 or 1.



▼ ⭕ Stack Analysis - Binary String