Recursion

Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instance of same problem.

$$n! = n \times (n-1) \times (n-2) \times \cdots 1$$
$$= n \times (n-1)!$$
$$(n-1)! = (n-1) \times (n-2)!$$
$$(n-2)! = (n-2) \times (n-3)!$$

#

```
Public class Factorial {
    Public static int fact(int n) {
        if(n==0) {
            return 1;
        }
        int smallAns = fact(n-1);
        return n*smallAns;
    }
Public static void main(String[] args){
    int n=3;
    int ans = fact(n);
    System.out.println(ans);
}
```

BASE CASE

main() n=3
↓
fact(3) 3×2 n=3
↓
fact(2) 2×1
↓
fact(1) – 1×1
↓
fact(0) – 1

Recursion and PMI]— They are basically Inter-Related.

```
# Public class SumOfNaturalNumbers {
    Public static Int sum (INT n) {
        if(n==1)
        { return 1;
        }
        return sum(n-1) + n;
    }
Public static void main(String[] args) {
    Int n=4;
    System.Out.println(sum(n));
    }
}
```

Calculate POWER

```
{
Public class Solution {
Public static Int power(int x, Int n) {
    if(n==0) {
        return 1;
    }
    return x * power(x, n-1);
}
```

x^n

$= x$

25

$2 \times 2 \times 2 \times 2 \times 2$

2×2^4

$2 \times 2 \times 2^3 \times$

$2^0 =$

Print Numbers —— find debug

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

6

1

Print(5)

↓

Print(3)

↓

Print(1)

1 2 3

~~5, 3, 1~~

Print(3)

↓

Print(2)

↓

Print(1)

Fibonacci Number

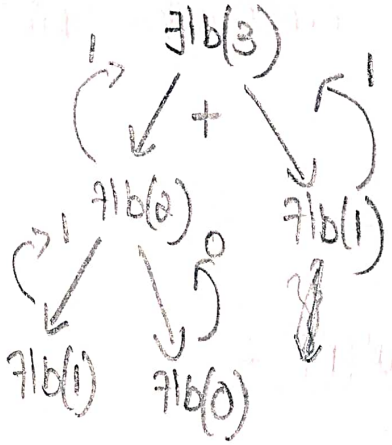
0 1 2 3 4 5 6 7

0, 1, 1, 2, 3, 5, 8, 13

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

```
Public class FibonacciNumber
{
    Public static int fib(int n){
        if(n==0 || n==1){
            return n;
        }
    return fib(n-1) + fib(n-2);
    }

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



Recursion and Arrays

→ Recursion STEPS

① Base class

② (n-1)/smaller Problem call

③ Update / Processing

2	4	1	6	9	5

Check SORTED Array

```
Public static boolean checkSorted(int input[]) {
  if(input.length <= 1)
  {
    return true;
  }

  int smallInput[] = new int[input.length - 1];
  for(int i=1; i< input.length ; i++) {
    smallInput[i-1] = input[i];
  }

  boolean smallAns = checkSorted(smallInput);
  if(!smallAns){
    return false;
  }

  if(input[0] <= input[1]) {
    return true;
  } else {
    return false;
  }

Public static void main(String[] args) {
  int input[] = {1,2,3};
  System.out.println(checkSorted(input));
}
```

I/P | 1 | 2 | 3 |

↑ T S/P = | 2 | 3 |

I/P | 2 | 3 |

↑ T S/P | 3 |

I/P | 3 |

```
Public static boolean checkSorted_2(int Input[])
{ if(Input.length<=1)
  { return true;
  }
  if(Input[0] > Input[1])
  { return false;
  }
  int smallInput[] = new int[Input.length-1];
  for(int i=1; i< Input.length ; i++){
    SmallInput[i-1] = input[i];
  }
  boolean smallAns = checkSorted_2(SmallInput);
  return smallAns;
}
```

SUM OF ARRAYS ——x——

```
Public class solution {
Public static int sum(int Input[]){
  if(input.length ==1){
    return input[0];
  }
int small[]= new int[Input.length-1];
for(int i=1; i< Input.length ; i++){
   small[i-1] = Input[i];
}
return input[0] + sum(small) ; } }
```

Check Number In Array

#

{ 9 8 7

```
Public static boolean checkNumber(int Input[], int x)
{
  if(Input.length<=1 && Input[0]!=x){
    return false;
  }
  if(Input[0]==x){
    return true;
  }
  int small[]= new int[Input.length-1];
  for(int j=1; j<Input.length; j++)
  {
    small[j-1] = Input[j];
  }
  return checkNumber(small[], x);
}
```

Recursion and Array

```
// This function checks if the array is sorted from startIndex to end.
Public static boolean checkSortedBetter(int input[], int startIndex)
{ if(startIndex >= input.length-1)
  {
    return true;
  }
  if(input[startIndex] > input[startIndex+1])
  { return false;
  }
  boolean smallAns = checkSortedBetter(input, startIndex+1);
  return smallAns;
}

Public static void main(string[] args) {
  int input[] = {1,2,3};
  System.out.println(checkSortedBetter(input,0);
}
```

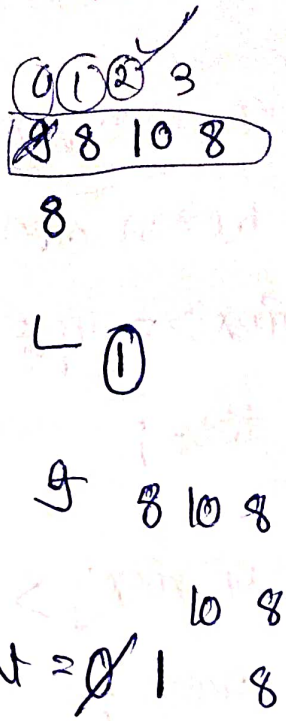
Recursion and helper function ——X——

```
→ Public static boolean checkSortedBetter(int input[]){
      return checkSortedBetter(input,0);
}
```

→ Upar vala function bula
HO JAA AA GAA.

FIRST INDEX OF NUMBER

```
#
{
  static int count=0;
Public static int firstIndex(int input[], int x)
{
  if(input.length<=1 && input[0]!=x) {
    return -1;
  }
  if(input[0]=x) {
    return count;
  }
  int small[] = new int[input.length-1];
  for(int i=1; i<input.length; i++) {
     small[i-1] = input[i];
    }
    count++;
   return firstIndex(small, x);
  }
}
```



~~LAST INDEX OF NUMBER~~

```
Public class solution {
Public static int FirstIndex(int Input[], int x, int startIndex) {
 if(startIndex == Input.length) { //// input
   return -1;
 }
 if(input[startIndex] == x) {
   return startIndex;
 }
 return FirstIndex(input, x, startIndex+1);
}

Public static int FirstIndex(int Input[], int x){
   return FirstIndex(input, x, 0);
}
```

→ LAST INDEX OF NUMBER

```
# {Public static int LastIndex(int Input[], int x) {
     return lastIndex(Input, x, input.length-1);
   }
   Public Static int lastIndex (int Input[], int x, int Index) {
    if(Index == -1) { return -1; }
    if (Input[Index] == x) { return Index; }
    return lastIndex(Input, x, index-1);
   }
}
```

All Indices of Number

x = 8

```
# {
Public static int[] allIndexes(int input[], int x){
    return allIndexes(input, x, 0);
}

Public static int[] allIndexes(int input[], int x, int startIndex){
  if(startIndex >= input.length){
    int ans[] = new int[0];
    return ans;
  }

int ans[] = allIndexes(input, x, startIndex+1);
  if(input[startIndex] == x)
  { int temp[] = ans;
    ans = new int[temp.length+1];
    ans[0] = startIndex;
    for(int i=0; i<temp.length; i++){
      ans[i+1] = temp[i];
    }
    return ans;
  }
  else {
      return ans;
  }
}
```

input =

0	1	2	3	4
9	8	7	8	8

(5)

0+1

0 → 1 → 2 → 3 → 4 → 5 ans[]

Mko Nca Aaya Samj Nceche Sa.

9	8	7	8	8

```
Public static int[] allIndices
  (int[] arr, int x, int idx, int fsf) {
if(idx == arr.length)
{
  return new int[fsf];
}
if(arr[idx] == x) {
int[] iarr = allIndices(arr, x, idx+1, fsf+1);
iarr[fsf] = idx;
return iarr;
}
else
{ int[] arr = allIndices(arr, x, idx+1, fsf);
  return iarr;
}
}
```

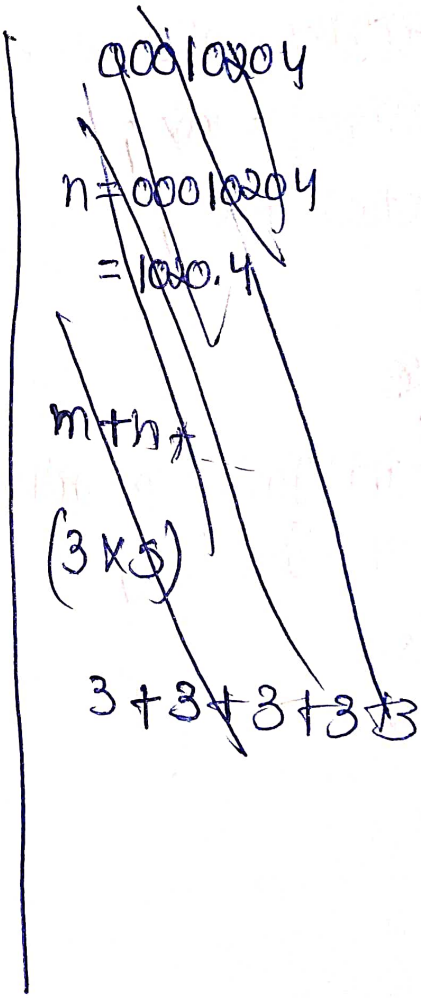
```
allIndexes(input, x, 0)

allIndexes (int input[], int x, int startIndex){
if(startIndex == input.length)
 { int ans[] = new int[0];
   return ans[];
}

int ans =
return (input, x, startIndex + 1);
if (input[startIndex] == x]
{  int temp[] = ans[];
   ans[] = new int[temp.length+1];
   ans[0] = startIndex
   for(i=0; i< temp.length+1 ; i++){
    ans[i] = temp[i];
    }
    return ans;
}
else {
    return ans;
    }
```

MULTIPLICATION (RECURSIVE)

```
Public class solution {
    Public static int multiplyTwoIntegers(int m, int n)
    {
        if(m == 0 || n == 0)
            return 0;
        else
            return m + multiplyTwoInteger(m, n-1);
    }
```



~~count zero~~

```
Public class solution {
    Public static int countzeroRec(int input) {
        if(input % 10 == 0)
            count++;
        if(input == 0) {
            return 1;
        }
        countzeroRec(input/10);
        return count;
    }
```

— Geometric sum

$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^k}$

$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^5}$

```
1
Math.Pow(2, k)
```

#

```
Public class solution
{
  Public static double FindGeometricSum(int k)
  {
    if(k==0)
    {
      return 1;
    }
    return FindGeometricSum(k-1) + 1/Math.pow(2,k);
  }
}
```

— Check Palindrome (recursive)

```
Public class solution{
  static int J;
  Public static boolean IsStringPalindrome(String input){
    J = input.length()/2;
    return IsStringPalindrome(input,0);
  }

  Public static boolean IsStringPalindrome(String input, int i){
    int k = input.length()-1;
    if(i==J){
      return true;
    }
    if(input.charAt(i) == input.charAt(k-i)){
      return IsStringPalindrome(input, i+1);
    }
    else{ return false;
    }
}}
```

racecar

#

```
Public static boolean IsStringPalindrome(String input) {
    if (input.length() <= 1) {
        return true;
    }
    if (input.charAt(0) == input.charAt(input.length() - 1)) {
        return IsStringPalindrome(input.substring(1, input.length() - 1));
    }
    else {
        return false;
    }
}
}
```

Sum of digits (Recursive)

```
{
Public static int sumofDigits(int input) {
    if (input == 0) {
        return 0;
    }
    int k = input % 10;
    return k + sum of digits(input/10);
}
```