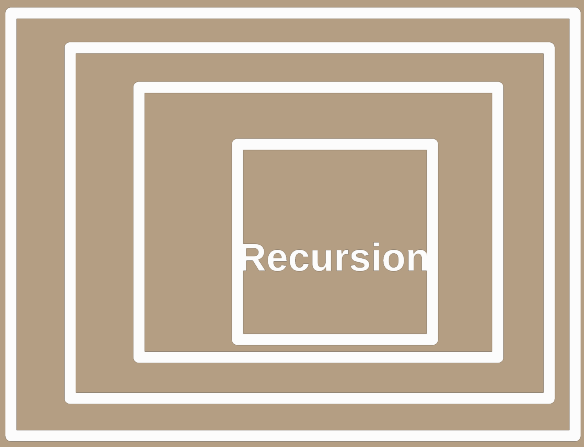
# Recursion




SOURAV SARKAR

## Recursion :-

### Identification :-

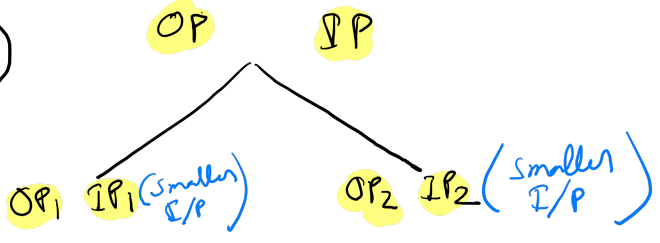
Recursion = choices + Decisions

| | a | b | c | |
|---|---|---|---|---|
| " " | ✗ | ✗ | ✗ | |
| a | ✓ | ✗ | ✗ | |
| b | ✗ | ✓ | ✗ | |
| c | ✗ | ✗ | ✓ | |
| ab | ✓ | ✓ | ✗ | |
| bc | ✗ | ✓ | ✓ | |
| abc | ✓ | ✓ | ✓ | |

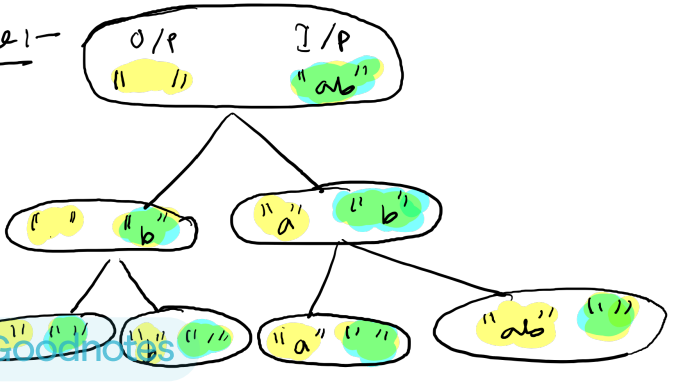
### Decision Tree :-

# no of Branches ⇓ # no of choices

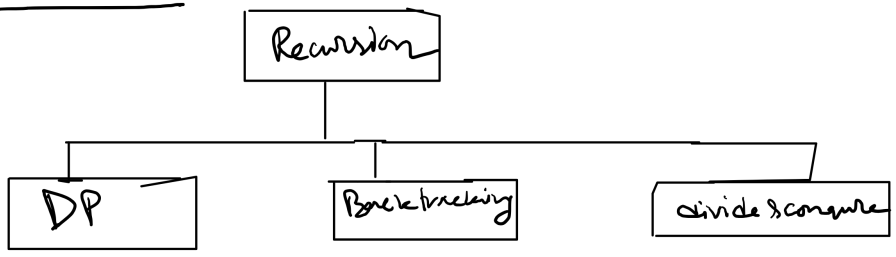
### I/P – O/P method




### Example :-



## Recursion Use:-




## Recursion Problem Pattern:-


## Recursion solving technique:-

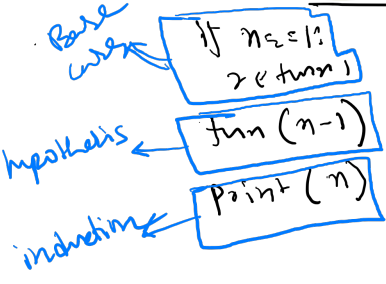
Simple question (1) Induction - Base - hypothesis

Medium question (2) Recursion Tree

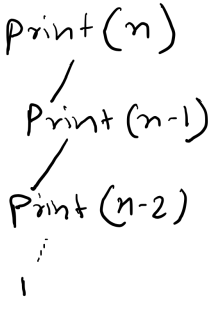
Hard question (3) choice Diagram

## Induction - Base - Hypothesis:-

-: Print 1 to n :-




print(n-1)+"n"



## Beauti of Hypothesis :-

1 — N

fun(n) → 1 .... n

fun(n-1) → 1 .... n-1

N - 1

fun(n) → n ..... 1

fun(n-1) → n-1 ..... 1

## Code 1 -

```
def fun(n)
    if n==1:
        print(n)
        return
    print(n)
    fun(n-1)
```

base case

induction

hypothesis

## Height of binary Tree :-

RBH :-

Induction → Base → Hypothesis

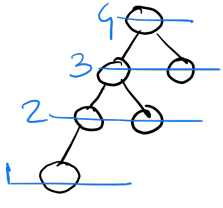
```
if not root:
    return 0
lh = fun(root.left)
rh = fun(root.right)
return 1+max(lh,rh)
```

Base case

hypothesis

Induction step

null = 0



# Sort an array using Recursion:—

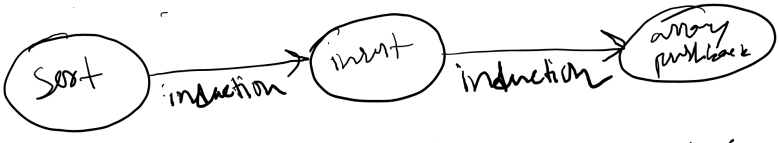
We will use IBH method (Induction → Base → Hypothesis)

**approach** —

Base case
if len(arr):
return;

Sort([1|2|3|4])
Sort([1|2|3])
Sort([1|2])
Sort([1])

⇒ Each step we have short arrays




```
def sort(arr):
    if len(arr)==1:
        return
    temp = arr[-1]
    arr.pop()
    sort(arr)
    insert(arr, temp)
```

```
def insert(arr, item):
    if len(arr)==0 || arr[-1] <= item:
        arr.push(item)
        return
    val = arr[-1]
    arr.pop()
    insert(arr, item)
    arr.push(val)
    return arr
```

→ base case
→ hypothesis
→ induction

**Explanation of insert**

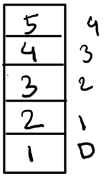
[0|1|5]  item=2

insert([0|1]) [5]

[0|1|2] [5]  push(5).

# Delete middle element from stack:-

## Induction - Base - hypothesis :-

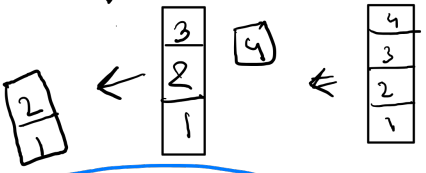



$k = \frac{size}{2} + 1$

$k = 3$

approach

we have to send smaller input




Solve(arr, k)

⇓

Solve(small arr, k-1)

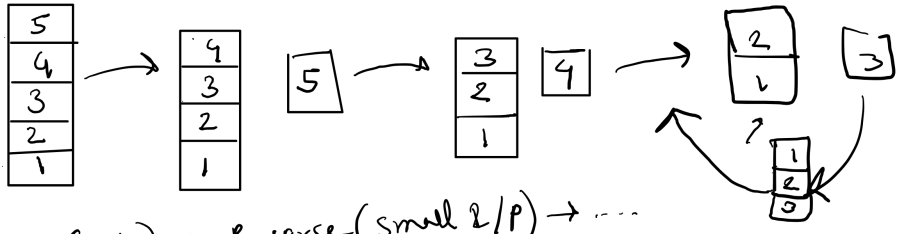
**Base case :-** if k == 1:
S.pop()

## Code :-

```
def delete(arr):
    if len(arr) == 0:
        return
    k = len(arr)//2 + 1
    Solve(arr, k)
    return S
```

```
def Solve(arr, k):
    if k == 1:                    # Base case
        S.pop()
        return
    temp = arr.top()
    S.pop()
    Solve(arr, k-1)
    S.push(temp)
    return
```

# Reverse Stack using Recursion:-



Reverse (I/P) → Reverse (small I/P) → ....

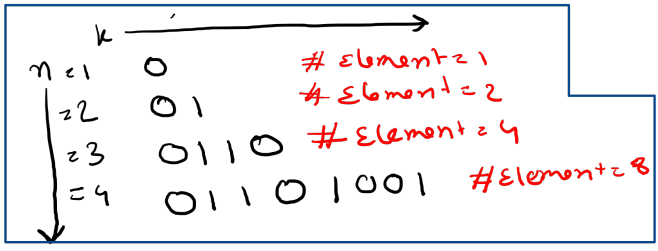
insert ( Reverse O/P, item)

```
def insert ( stack, item):
    if len(stack)==0:
        stack.push(item)
        return
    val = stack.top()
    stack.pop()
    insert ( stack, item)
    stack.push(val)
    return stack
```

```
def reverse(stack):
    if len(stack) == 1:
        return
    val = stack.top()
    stack.pop()
    reverse(stack)
    insert(stack, val)
    return stack
```

# Kth Symbol in grammar :-

Formula of no of Element $= 2^{n-1}$




Observation :- n=3 0110 rev

n=4 0110 1001

mid

Code :-

```
def KthElement(n,k)
    if n==1 and k==1:
        return 0
    mid = 2**(n-1)//2

    if k<=mid:
        return kthElement(n-1,k)
    else:
        return 1-kthElement(n-1,k-mid)
```

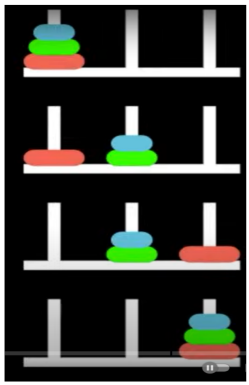
# Recursive problem solving approach :-

Lets f(n) be a recursive function

1. shows f(1) works
2. Assume f(n-1) works
3. shows f(n) work using f(n-1)

## Tower of hanoi :-

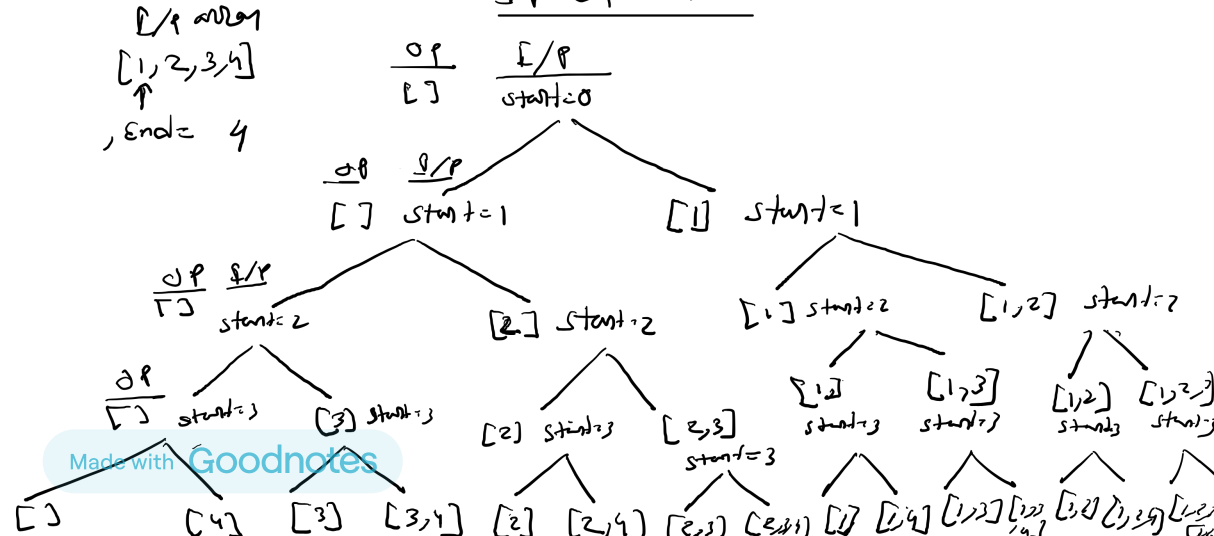
```
def tower_hanoi(n, source, aux, des):
    if n==1:
        print("....")
        return
    tower_hanoi(n-1, source, des, aux)
    print
    tower_hanoi(n-1, aux, src, des)
```



## Subsets :-

I/P array
[1,2,3,4]
↑
, end = 4

### IP-OP method



# all subsets :-

[ ], [4], [3], [3,4], [2], [2,4], [2,3], [2,3,4], [1], [1,4], [1,3], [1,3,4], [1,2], [1,2,4], [1,2,3], [1,2,3,4]

## Code :-

```python
class Solution(object):
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        result = []
        def sub(nums,start,res):
            if len(nums)==start:
                result.append(res)
                return
            sub(nums,start+1,res+ [nums[start]])
            sub(nums,start+1,res)
        sub(nums,0,[])
        return result
```

base case

hypothesis

We have two choices taking a element or not taking

sub(nums, start+1, res+[nums[start]]) we are taking nums[start]

sub(nums, start+1, res) we are not taking nums[start]

By using res + [nums[start]], each recursive call works with an independent copy of res, so it won't affect other recursive calls.

~~We are not taking~~ res.append(nums[start]) because res is sharing variable for each recursion and we need copy

sub(nums, start+1, res)

## all unique subsets :-

Same solution like last problem just I have to sort the nums and we have to use hashset to remember it came before or not.

### 90. Subsets II

Solved

Medium · Topics · Companies

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

**Example 1:**

```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

**Example 2:**

```
Input: nums = [0]
Output: [[],[0]]
```

Python · Auto

```python
class Solution(object):
    def subsetsWithDup(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """

        result = []
        seen = set()
        nums.sort()
        def sub(nums,start,res):
            if len(nums)==start:
                if tuple(res) not in seen:
                    result.append(res)
                    seen.add(tuple(res))
                return

            sub(nums,start+1,res+ [nums[start]])
            sub(nums,start+1,res)
        sub(nums,0,[])
        return result
```

The code you provided has almost everything in place, but there's one small issue: the input list nums needs to be **sorted** initially to ensure duplicates are adjacent. Without sorting, subsets with duplicate elements may appear in different orders, causing the set to consider them as different subsets.

We are checking that we already add to our result or not if not then we are adding as result

## Permutation with Spaces :—

I/P: ABC

O/P: A B C , AB C, A BC, ABC

Edge case :—

_ ABC _

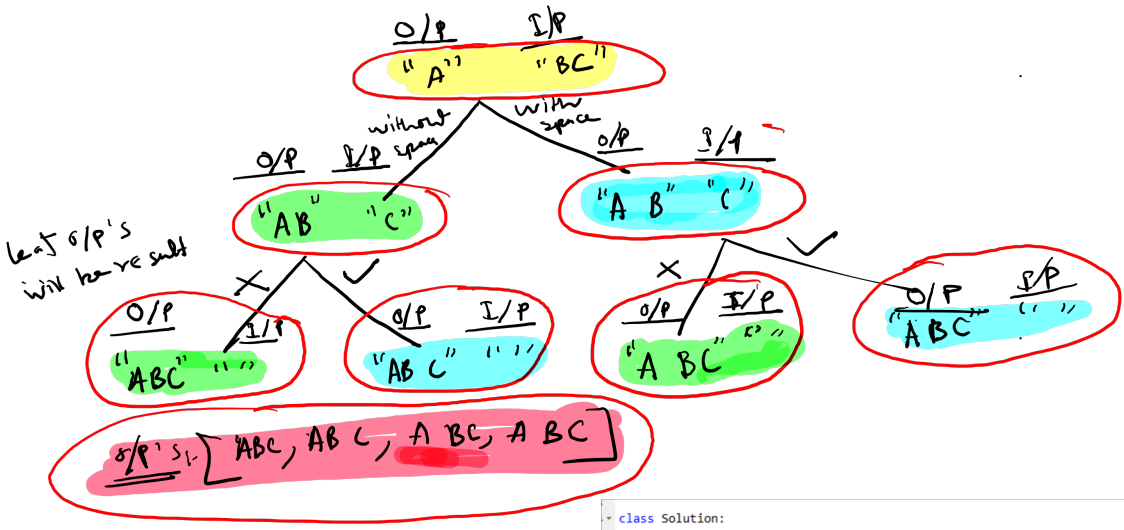
↑ ↑

We don't have to put before and after the String

We will use Input - output method, we will make decision tree.

Step 1 :— we will start from 1st index and we will add first index character to output.

Suppose I/P = "ABC"




```python
class Solution:

    def permutation(self, s):
        # code here
        result = []

        # Helper function for recursive generation
        def sub(nums, start, res):
            # If we've reached the end of the string, add the result
            if start == len(nums):
                result.append(res)
                return

            # Recursive case 1: Add the next character without space
            sub(nums, start + 1, res + nums[start])

            # Recursive case 2: Add the next character with a space
            sub(nums, start + 1, res + " " + nums[start])

        # Start the recursion from the first character
        sub(s, 1, s[0])

        # Sort the result in lexicographical order before returning
        return sorted(result)
```

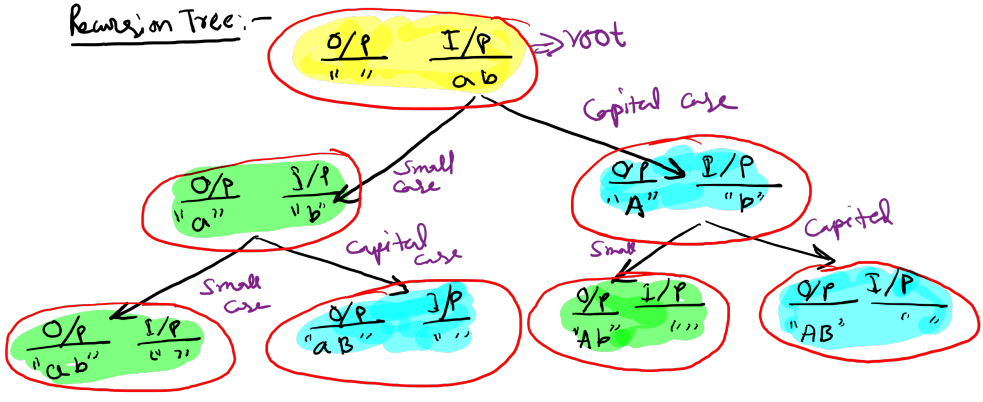
Base Case

hypothesis

start from 1st index and o/p having first index character

**Permutation with Case Change :-**

I/P : ab
O/P : AB Ab ab aB




```python
class Solution:

    def permutation(self, s):
        # code here
        result = []

        # Helper function for recursive generation
        def sub(nums, start, res):
            # If we've reached the end of the string, add the result
            if start == len(nums):
                result.append(res)
                return

            # Recursive case 1: Add the next character without space
            sub(nums, start + 1, res + nums[start].lower())

            # Recursive case 2: Add the next character with a space
            sub(nums, start + 1, res + nums[start].upper())

        # Start the recursion from the first character
        sub(s, 0, "")
```

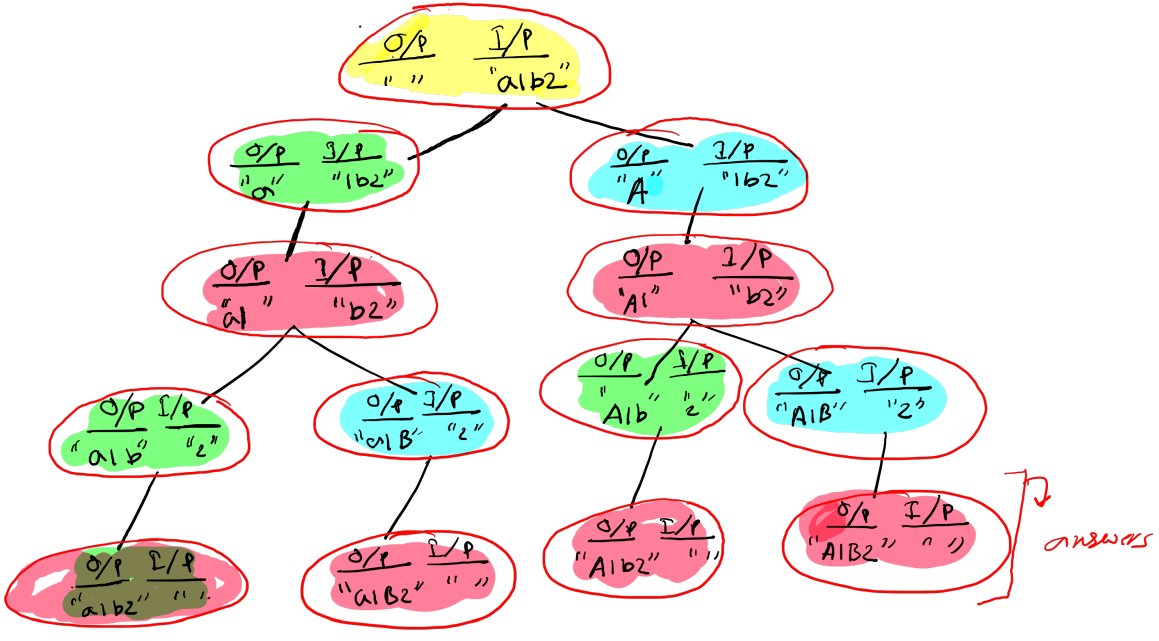
Base case

Recursive part
Sub(upper)
Sub(lower)

Sub(S, 0th index, Empty String)

## Letter case Permutation

**Input:** s = "a1b2"
**Output:** ["a1b2","a1B2","A1b2","A1B2"]




### 784. Letter Case Permutation

Solved

Medium · Topics · Companies

Given a string `s`, you can transform every letter individually to be lowercase or uppercase to create another string.

Return *a list of all possible strings we could create*. Return the output in **any order**.

**Example 1:**

```
Input: s = "a1b2"
Output: ["a1b2","a1B2","A1b2","A1B2"]
```

**Example 2:**

```
Input: s = "3z4"
Output: ["3z4","3Z4"]
```

**Constraints:**

Python · Auto

```python
class Solution(object):
    def letterCasePermutation(self, s):
        """
        :type s: str
        :rtype: List[str]
        """
        result = []
        def permutation(s,start,res):
            if start == len(s):
                result.append(res)
                return

            if s[start].isalpha():
                permutation(s, start+1, res+(s[start]).lower())
                permutation(s, start+1, res+(s[start]).upper())
            else:
                permutation(s, start+1, res+s[start])

        permutation(s, 0, "")
        return result
```

Saved

base case

if alphabets recursive all for lower, higher

if digit just we have to add

# Print N-bit binary numbers having more 1s than 0s

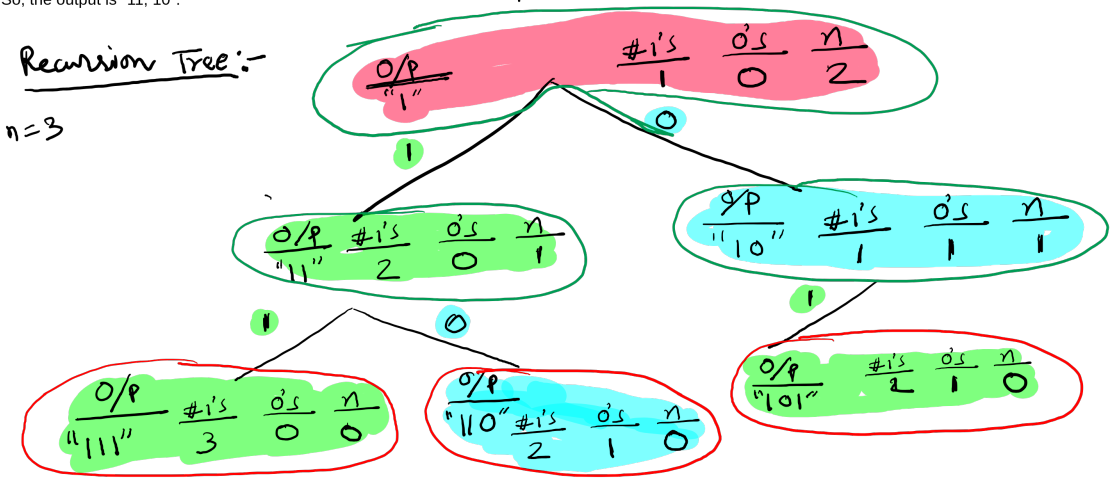
Given a positive integer **n**. Your task is to generate a string list of all **n-bit binary numbers** where, for any prefix of the number, there are **more or an equal** number of 1's than 0's. The numbers should be sorted in **decreasing order of magnitude**.

**Input:**
n = 2
**Output:**
{"11", "10"}
**Explanation:** Valid numbers are those where each prefix has more 1s than 0s:
11: all its prefixes (1 and 11) have more 1s than 0s.
10: all its prefixes (1 and 10) have more 1s than 0s.
So, the output is "11, 10".

Trick here :-
we have to find the prefixes of n digits where '#1's > #0's
So we have to start with 1.....

Recursion Tree :-

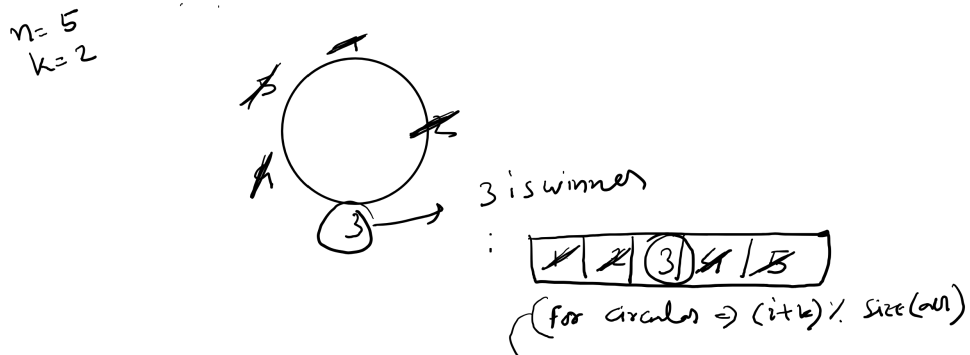
n=3




```python
class Solution:
    def NBitBinary(self, n):
        # code here
        result = []

        def rec(s,no_of_0, no_of_1,n):
            if n==0:
                result.append(s)
                return

            rec(s+"1",no_of_0,no_of_1+1,n-1)
            if no_of_1 > no_of_0:
                rec(s+"0",no_of_0+1,no_of_1,n-1)

        rec("1",0,1,n-1)
        return result
```

## Josephus Problem(Find the winner in circular game)

There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in **clockwise order**. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend.
The rules of the game are as follows:
**Start** at the 1st friend.Count the next k friends in the clockwise direction **including** the friend you started at.
The counting wraps around the circle and may count some friends more than once.
The last friend you counted leaves the circle and loses the game.If there is still more than one friend in the circle, go back to step 2 **starting** from the friend **immediately clockwise** of the friend who just lost and repeat.Else, the last friend in the circle wins the game.
Given the number of friends, n, and an integer k, return *the winner of the game*.




```
class Solution(object):
    def findTheWinner(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: int
        """
        def rec(arr, i , k):
            if len(arr)==1:
                return arr[0]
            remIdx = (i+k)%len(arr)
            arr.pop(remIdx)
            return rec(arr, remIdx, k)
        k = k-1
        arr = []
        for i in range(n):
            arr.append(i+1)
        return rec(arr,0,k)
```