Data Structures : Arrays

1. Access } S, T = O(1)
2. Set }

3. Traverse T = O(N), S = O(1)

4. Copy S, T = O(N)

Static Array : fixed size
next memory slot may not be empty.

Dynamic Array : allows O(1) insertion at end
OS allocates almost 2 times as much memory as needed.

5. Insertion :

- at beginning : O(N) } Static, Dynamic
- at end : O(1) : Dynamic, O(N) : Static
- In between : O(N) } Static, Dynamic

6. Remove : Same as Insertion

Question :

977. Squares of a Sorted Array

Input : nums = [-4, -1, 0, 3, 10]

Output : [0, 1, 9, 16, 100]

nums is an array in ascending order

Brute force way : Square the array and then sort it
This give Time = O(nlogn)
Space = O(n)

Method 2 : Using two pointers :

Compare the first and last value in the input array and put the greatest among them into a new array at last position.
Now shift the pointer

```
def sorted_squared(array)
    n = len(array)
    res = [0] * n
    i, j = 0, n-1
    for k in reversed(range(n)):
        if array[i]**2 > array[j]**2:
            res[k] = array[i]**2
            i += 1
        else:
            res[k] = array[j]**2
            j -= 1
    return res
```

Question :

896. Monotonic Array

An array is monotonic if it is either monotone increasing or monotone decreasing.

Input : [1, 2, 2, 3] Output : True

Input : [1, 3, 2] Output : False

```
def monotonic_array(array):

    n = len(array)
    if n == 0 : return True
    first, last = array[0], array[n-1] :

    if first > last :
            # decreasing
        for k in range(n-1) :
            if array[k] < array[k+1] :
                return False
    elif first == last :
        for k in range(n-1) :
            if array[k] != array[k+1] :
                return False
    else :
        for k in range(n-1) :
            if array[k] > array[k+1] :
                return False

    return True
```

RECURSION

Q. When to use recursion.

Ans When the given problem can be divided into smaller sub-problem.

Recursive Leap of Faith :

(1) Understand the problem → print 321123

(2) Identify subproblem → 2112
(3) Trust | Faith
(4) Link ① and ② → 321123
(5) Base condition → for 0, return

* Ways to write Base Condition :

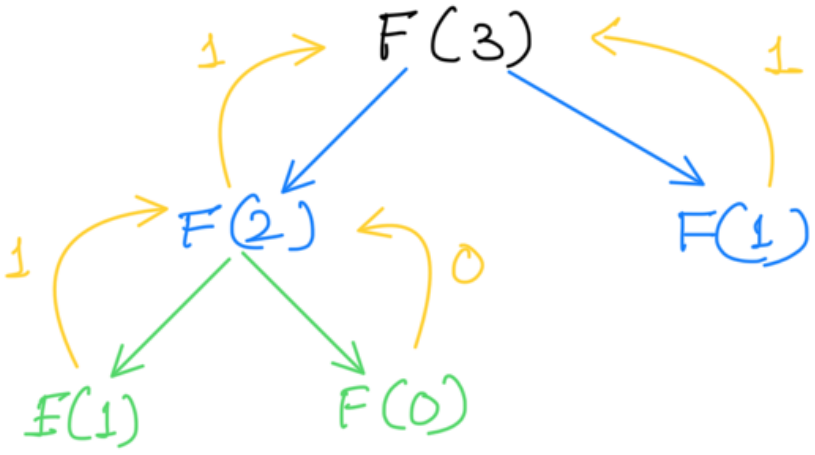
- last valid input
- first invalid input

* Recursion Tree :

Let take fibonacci series as example to draw the recursion tree. We know that by using Recursion Fibonacci series is given by:

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0$$
$$F(1) = 1$$

Let take $n = 3$



$$\text{So } F(3) = 1 + 1 = 2$$

A Recursion tree consists of ascending Phase and Descending phase.

* Recursion Approaches :

(1) 0 to N
(2) N to 0

Example :

Given n, find 1+2+3+. . . . . n :

0 to N :

```
function sum ( curr , n )
{
  if curr = n : return n

  return curr + sum (curr+1 , n)
}
```

N to 0 :

```
function sum(n)
{
  if n=0: return 0
  return n + sum(n-1)
}
```

* Complexity Analysis of Recursive Solutions :

Time Complexity : [nodes] x [work done per node]

Space Complexity : Depth

Questions :

779. K-th Symbol in Grammar

n = 1
n = 2
n = 3
n = 4

0
0 1
0 1 1 0
0 1 10 10 0 1

Replace 0 with 01 and 1 with 10

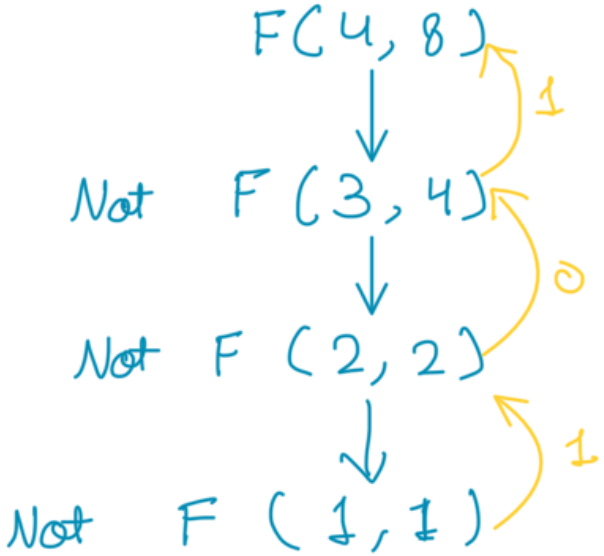
Express the solution of a problem as a function of the solutions to smaller instances of the same problem

Observation :

1) nth row first half is same as previous row

2) nth row second half is NOT of previous row.

For (n=4, k=8) Recursion tree =



Since k is greater than N we add Not

Base Case : If n==1 return 0

T = O(N) , S = O(N)

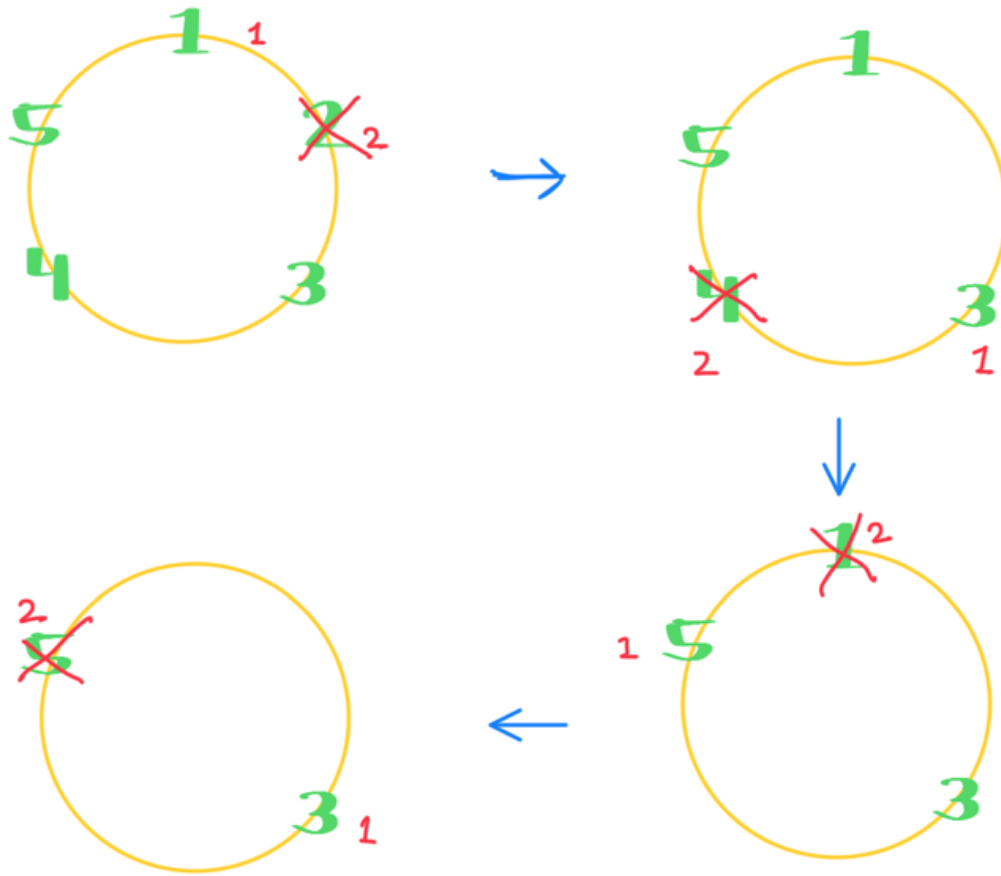
CODE :

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

    if k <= mid
        return Kth_symbol(n-1, k)
    else:
        return int(not kth_symbol(n-1, k-mid))
```

Question :

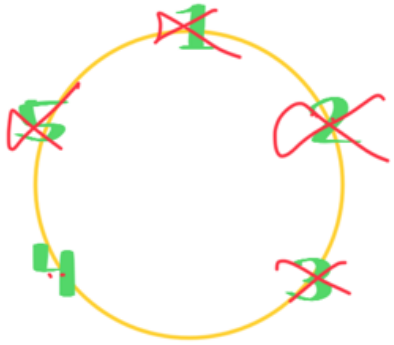
1823. Find the winner of the Circular game / Josephus Problem

Input n = 5 , k = 2



Output = 3

Similarly n = 6 , k = 5 Output = 1



n = 5

k = 3

n = 5 k = 3 Output = 4

3 Approaches :

1st approach → $T = O(n^2)$, $S = O(n)$

2nd approach → $T = O(n)$, $S = O(n)$

3rd approach → $T = O(n)$, $S = O(1)$

Some points :

1. When you have something circular consider % (modulo) → remainder after division

Let n = 5 k = 7 , Let try by using an array

0	1	2	3	4
1	2	3	4	5

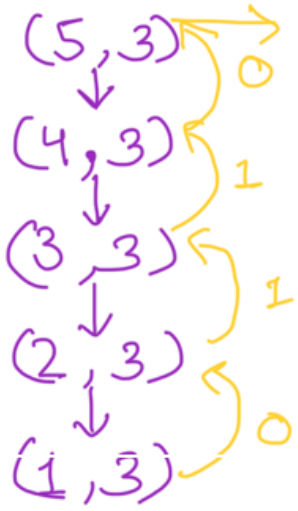
$[0 + 7 - 1] \% 5 = 1$

So index 1 will be eliminated at first place

Approach 1 :

```
def findTheWinner(n, k):
    arr = [i+1 for i in range(n)]
    def helper(arr, start_index):
        if len arr == 1:
            return arr[0]
        index_to_remove = (start_index + k - 1) % len(arr)
        del arr[index_to_remove]
        return helper(arr, index_to_remove)
    return helper(0, arr)
```

Approach 2 : Lets take n = 5 and K = 3



[sp + k] % n

CODE ↓

```
def findTheWinner(n,k)
    def josephus(n):
```

```
if n == 1:
    return 0
return (josephus(n-1) + k) % n
return josephus(n) + 1
```

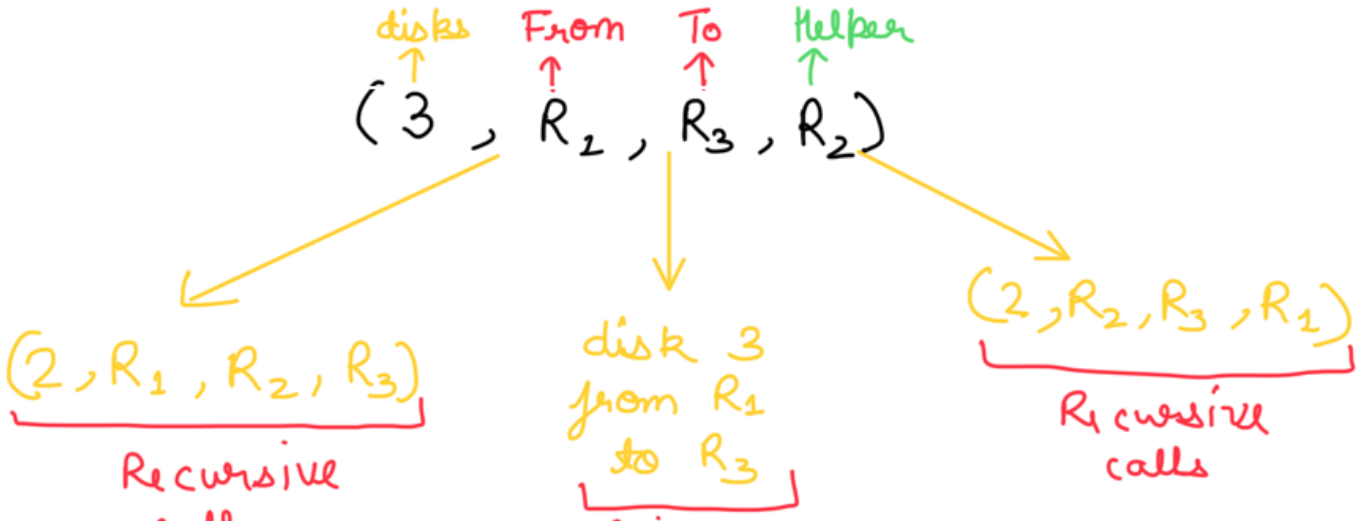
Iterative approach :

```
def findTheWinner(n,k)
    survivor = 0
    for i in range (2, n+1)
        survivor = (survior + k) % i
    return survivor +1
```

Question :

We have three rods and N disks. The objective of the puzzle is to move the entire stack to another rod. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, find & return the total moves.
Note: The discs are arranged such that the top disc is numbered 1 and the bottom-most disc is numbered N. Also, all the discs have different sizes and a bigger disc cannot be put on the top of a smaller disc. You can only move 1 disk at a time.

Lets take 3 disks as example



call

print statement

CODE 7

```
def toh(N, fromm, to, aux):
    count = 0
    def helper(N, from, to, aux):
        nonlocal count
        if N == 1:
            print("move disk" + str(N) + " from rod" + str(fromm) + " to rod" + str(to))
            count += 1
            return
        helper(n-1, from, aux, to)
        print("move disk" + str(N) + " from rod" + str(fromm) + " to rod" + str(to))
        count += 1
        helper(n-1, aux, to, from)
    helper(N-1, fromm, to, aux)
    return count
```

Space Complexity : $O(N)$
Time Complexity : $O(2^N)$

Question : Power Sum

```
def powersum(array, power=1):
```

```
sum = 0
for i in array:
    if type(i) == list
        sum += powersum(i, power+1)
    else:
        sum += i
return sum ** power
```

BACKTRACKING

Is an algorithmic approach to find solution to problems that involve many possible paths

→ Solutions are built step by step

→ If a path does not lead to a solution/ violates constraints than that particular path is abandoned

→ Controlled recursion

→ Makes changes in place (pass by reference)

Blueprint / Pseudocode:

```
function helper {
    if solved → save the solution/
                Print
                return

    for choice in choices {
        if is_valid(choice) {
            choose
            helper()
            revert choice
        }
    }
}
helper()
```

if limited choices (no need of for loop)

eg. powerset / subsets problem
- exclude
- include

*When to use Backtracking :

(1) If a problem requires every possible path

(2) There are multiple solutions and you want all of them

*Pruning : Cutting down the search space by removing options that are clearly incorrect or not promising.

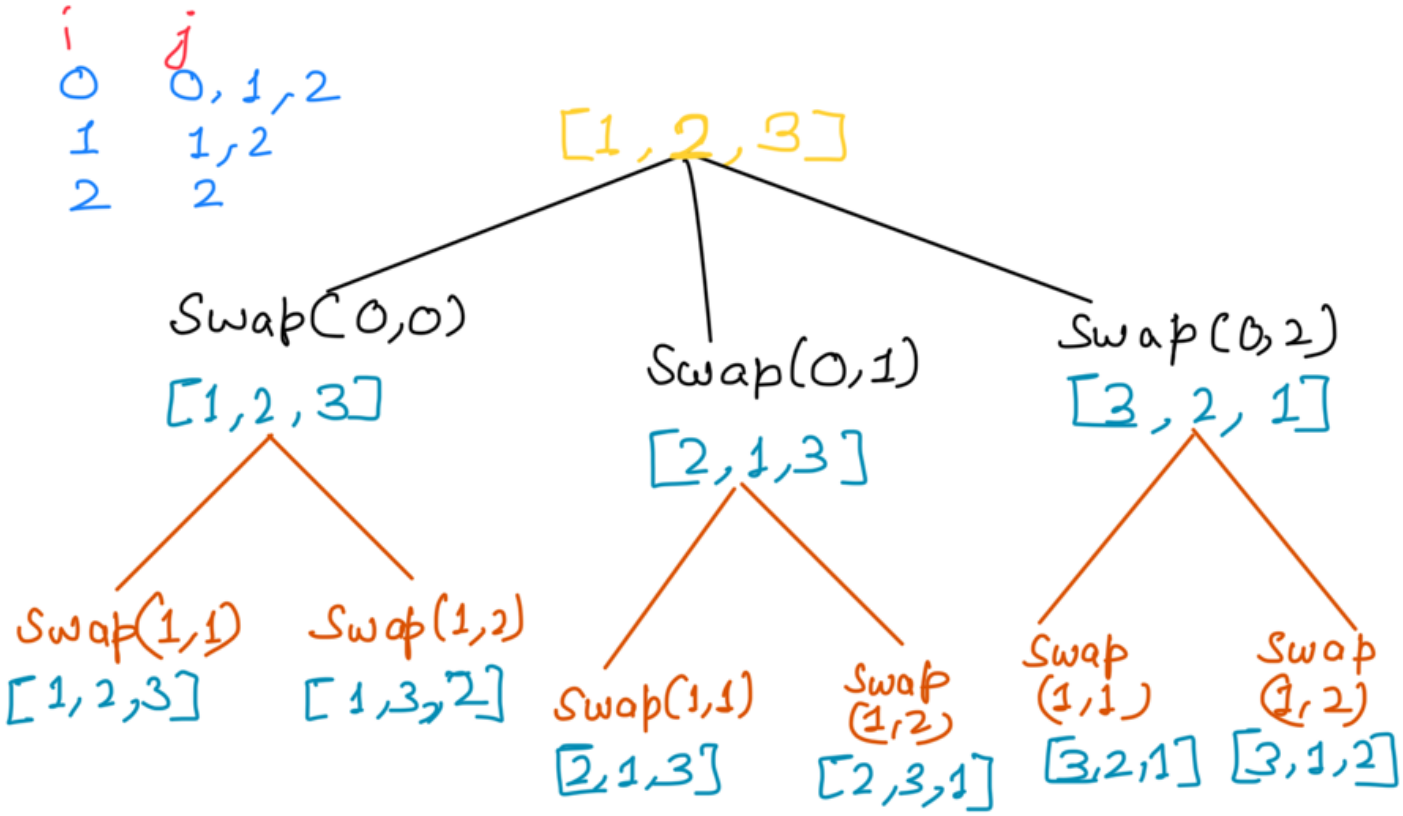
QUESTION :

QUESTION :

46. Permutations.

Input: nums = [1,2,3]

Output : [[1,2,3], [1,3,2], [2,3,1], [2,1,3], [3,1,2], [3,2,1]]



PSEUDOCODE :

```
function perm {
  if i == len(array) - 1 :
      add to result

  for j = i to len-1 {
      swap nums[i] and num[j]
      perm(i+1)
      swap nums[i] and num[i]
```

3

3 perm( )

CODE:

```
def permute(nums):
    n = len(nums)
    res = []

    def helper(index):
        if index = n-1:
            res.append(nums[:])
            return

        for j in range(index, n):
            nums[index], nums[j] = nums[j], nums[index]
            helper(index+1)
            nums[index], nums[j] = nums[j], nums[index]

    helper(0)
    return res
```

47. Permutations II

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

CODE:

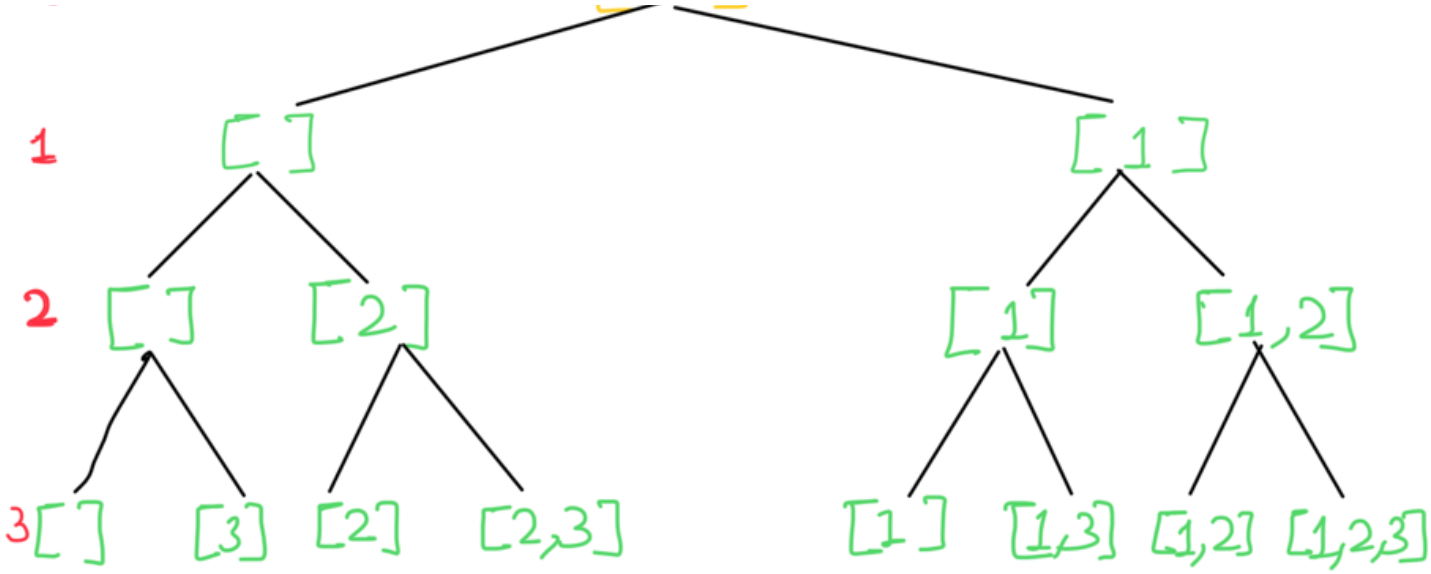
```
def permuteunique(nums):
    n = len(nums)
    res = []

    def helper(index):
        if index == n-1:
            res.append(nums[:])
            return

        hash = {}

        for j in range(index, n):
            if nums[j] not in hash:

                hash[nums[j]] = True
                nums[index], nums[j] = nums[j], nums[index]
                helper(index+1)
                nums[index], nums[j] = nums[j], nums[index]

    helper(0)
    return res
```

QUESTION:

78. Subsets →

Input: nums = [1, 2, 3]

Output: [[], [1], [2], [1,2], [3], [1,3], [2,3], [1,2,3]]



Now we are going to traverse this tree by using depth first search

CODE:

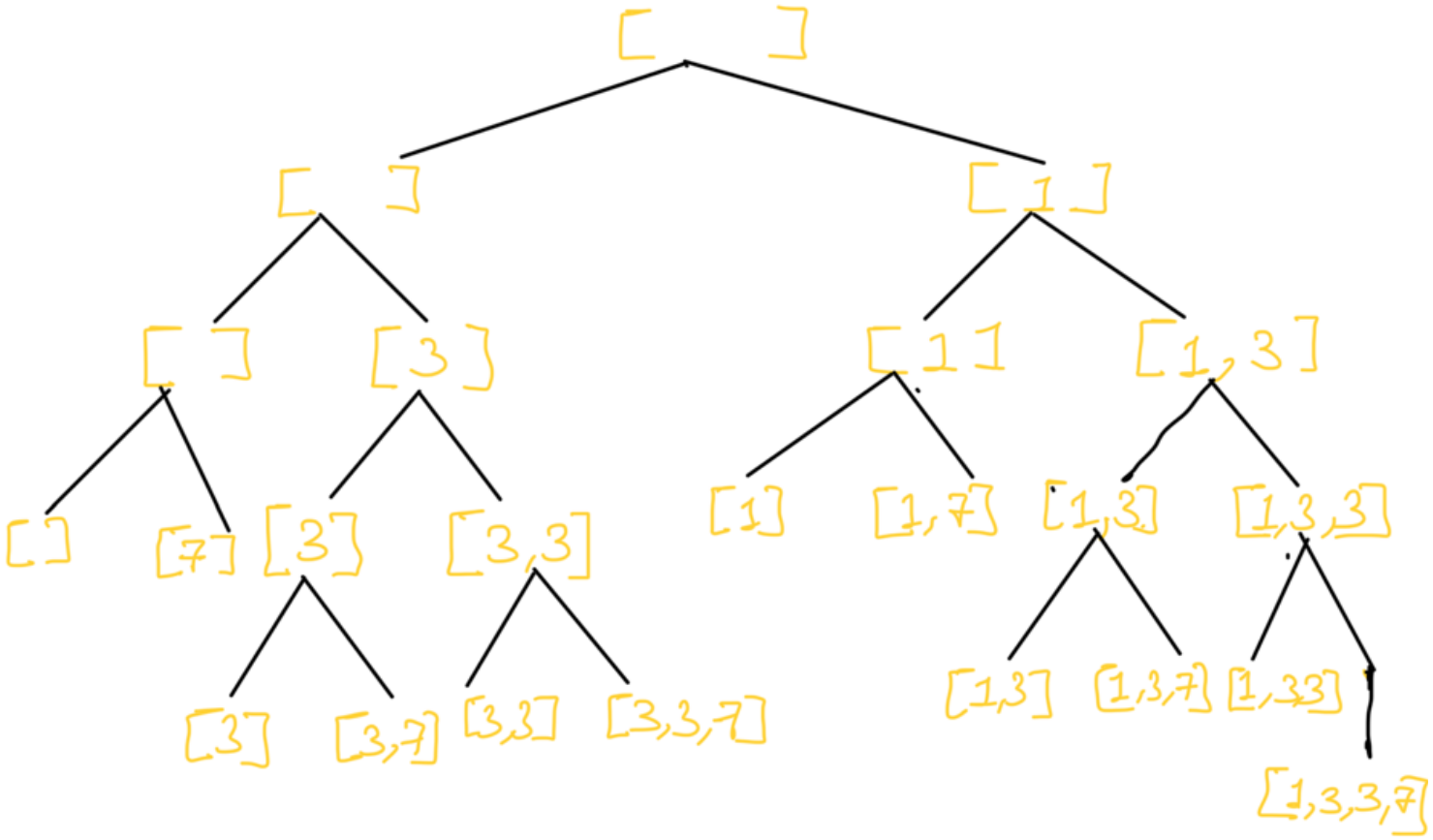
```
def subsets(nums):

    n = len(nums)
    res = []
    cur = []

    def backtrack(i):
        if i == nums:
            res.append(cur[:])
            return

        backtrack(i+1)
        cur.append(nums[i])
        backtrack(i+1)
        cur.pop()

    backtrack(0)
    return res
```

90. Subsets II

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

Different approach : In the exclude branch we will not consider any repeating number.

Example : [1,3,3,7]



CODE :

```
def subsetswithdup(nums):
    nums.sort()
    n = len(num)
```

```
res = []
cur = []

def backtrack(i):

    if i == n:
        res.append(cur[:])
        return

    # include
        cur.append(nums[i])
        backtrack(i+1)
        cur.pop()

    # exclude:
    while i<n and nums[i] == nums[i+1]:
            i += 1
    backtrack(i+1)

backtrack(0)
return res
```

Key points:

(i) Don't forget to sort the nums.
(ii) #include part before #exclude part.

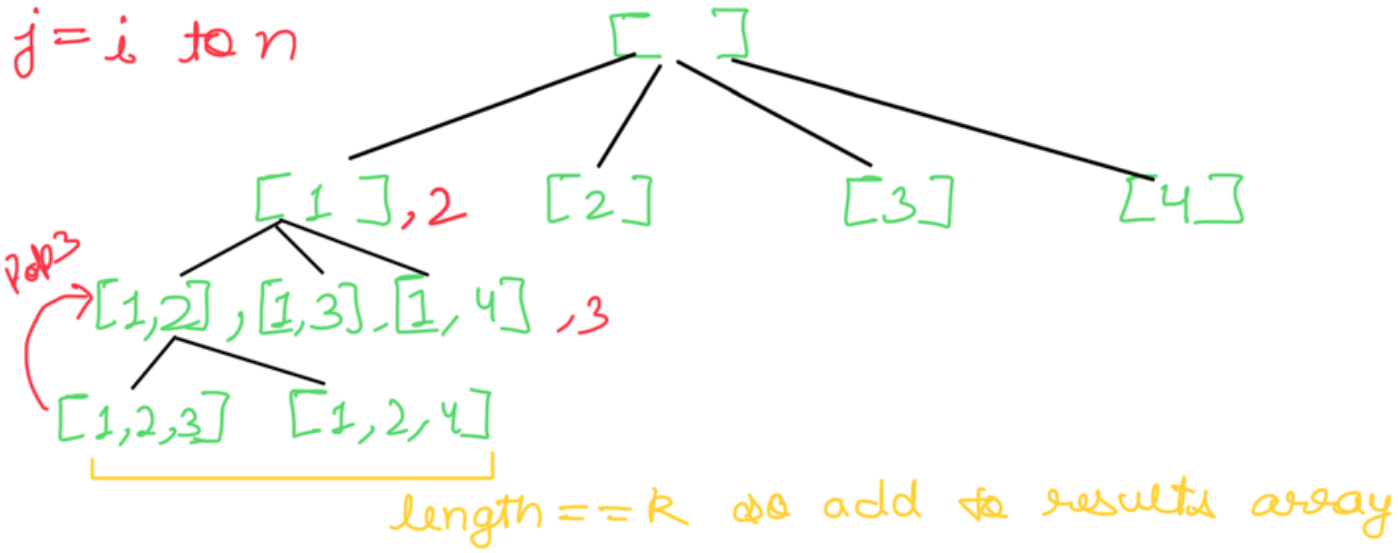
77 Combinations:

Input : n=4 R=2

Output : [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]

There are 4 choose 2 = total 6 combinations

Let's take example of n = 4 and k = 3



CODE:

```
def combine(n, k):
    res, cur = [], []

    def backtrack(start):

        if len(cur) == k:
            res.append(cur[:])
            return


        for j in range(start, n+1):

            cur.append(j)
            backtrack(j+1)
            cur.pop()

    helper(1)
    return res
```

For optimization purpose we can introduce a

"need" variable like need = k - len(cur)
then the loop goes from start to
"n - (need-1) + 1"

39. Combination Sum

Input : candidates = [2,3,5] , target = 8
output : [[2,2,2,2],[2,3,3],[3,5]]

CODE :

```
def combinationSum(candidates, target):

    cur, res = [], []
    n = len(candinates)

    def backtrack(start, cursum):

        if cursum == target:
            res.append(cur[:])
            return

        if cursum > target:
            return

        for j in range(start, n):

            cur.append(candidates[j])
            backtrack(j, cursum + candidates[j])
            cur.pop

    backtrack(0, 0)
    return res
```

40 Combination Sum II

Input: candidates = [2,5,2,1,2], target = 5

Output: [[1,2,2],[5]]

CODE:

```
def combinationSum2(candidates, target):
    candidates.sort()
    cur, res = [], []
    n = len(candidates)

    def backtrack(start, cursum):

        if cursum == target:
            res.append(cur[:])
            return

        if cursum > target:
            return

        if start > n-1:
            return

        hash = {}

        for j in range(start, n):
            if candidates[j] not in hash:
                hash[candidates[j]] = True
                cur.append(candidates[j])
                backtrack(j+1, cursum+candidates[j])
                cur.pop()

    backtrack(0,0)
    return res
```

Differences from Combination Sum 1 :

(1) Use of sorting to avoid duplication

(2) Use of hash map to store the used values

(3) Use "j+1" instead of 'j' as duplicates are not allowed.

216 Combination Sum III

Input : k = 3 n = 9

Output = [[1,2,6],[1,3,5],[2,3,4]]

```
cur, res = [], []

def backtrack(start, cursum):
    if cursum == n and len(cur) == k:
        res.append(cur[:])
        return
    if cursum > n:
        return
    for j in range(start, 10):
        cur.append(j)
        backtrack
```

QUESTION :

37. Soduko Solver

```
def solveSudoku(board):
```

```python
#The function modifies the board in place to present the solution. Hence there is no need to return the board

def isValid(board, row, col, num):
    # Check if a number 'num' can be placed at board[row][col].
    for x in range(9):
        # Check row and column: The number must not already exist in the same row and column.
        if board[x][col] == num or board[row][x] == num:
            return False

        # Calculate the start row and column index for the 3x3 sub-box.
        r = 3 * (row // 3) + x // 3
        c = 3 * (col // 3) + x % 3

        # Check 3x3 sub-box: The number must not already exist in the same 3x3 sub-box.
        if board[r][c] == num:
            return False

    # If the number 'num' is not found in the same row, column, and 3x3 sub-box, it is valid.
    return True

def helper(board):
    # Iterate through each cell in the board.
    for row in range(9):
        for col in range(9):
            # If the cell is empty ('.').
            if board[row][col] == '.':
                # Try placing each number from 1 to 9 in the empty cell.
                for num in '123456789':
                    # Check if the number is valid in the current position.
                    if isValid(board, row, col, num):
                        # Place the number in the cell.
                        board[row][col] = num

                        # Recursively attempt to solve the rest of the board with this number placed.
                        if helper(board):
                            return True

                        # If placing the number does not lead to a solution, reset the cell and try the next number.
                        board[row][col] = '.'

                # If no number from 1 to 9 can be placed in this cell, backtrack.
                return False

    # If the entire board is filled without conflicts, the puzzle is solved.
    return True

# Start the solving process.
helper(board)
```

Steps followed while writing the code:

(i) We need two function one function to fill the board and one function to check if the entry is valid or not

(ii) check for the empty cell

(iii) fill the empty cell with a valid number using isvalid function

(iv) Implement the isValid function.

51 N-Queens

Input = N # 4X4 chess board
output =

[[".Q..","...Q","Q...","..Q."],
["..Q.","Q...","...Q",".Q.."]]

2 distinct solutions for 4X4 N-Queen problem

CODE:

```
def solveNQueens(n):
    res = []
    board = [ ["."]*n for _ in range(n)]

    def convertBoard(board):
        return ["".join for row in board ]

    def isValid(row, col, board)
        for x in range(row]:
            if board[x][col] == 'Q':
                return False

        for r, c in zip(range(row,-1,-1),
                        range(col,-1,-1)):
            if board[r][c] == 'Q':
                return False

        for r, c in zip(range(row,-1 -1),
                        range(col, n )):
            if board[r][c] == 'Q'.
                return False
```

```
        return True

    def positionNextQueen(board, row):
        if row == n:
            res.append(convertboard(board))

        for col in range(n):
            if isValid(row, col, board):
                board[row][col] = "Q"
                positionNextQueen(board, row+1)
                board[row][col] = '.'

positionNextQueen(board, 0)
return res
```