If you have any questions or suggestions regarding the notes, please feel free to reach out to me on WhatsApp: **Sanjay Yadav Phone: 8310206130**

DSA: Tries - 8 - Aug 2023

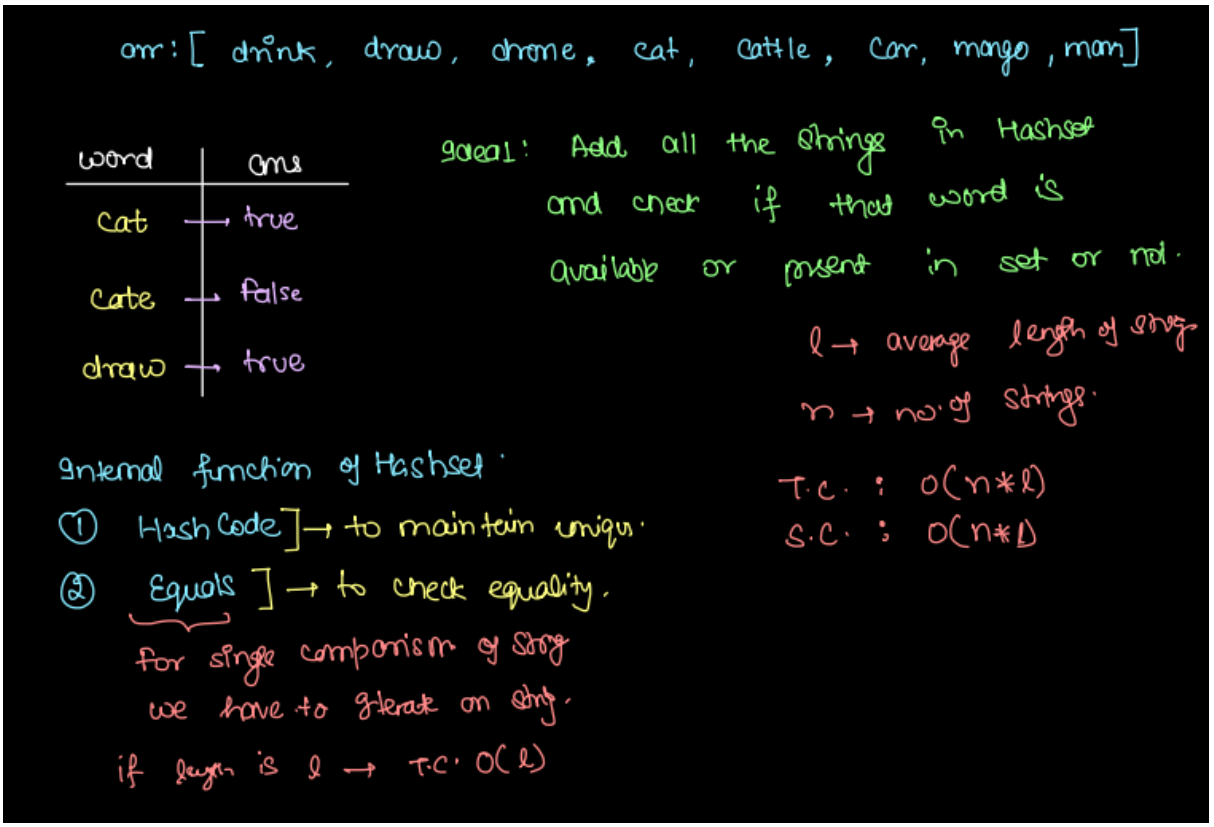
Search availability of word in array of words

Searching in TRIE

Count of string with given prefix

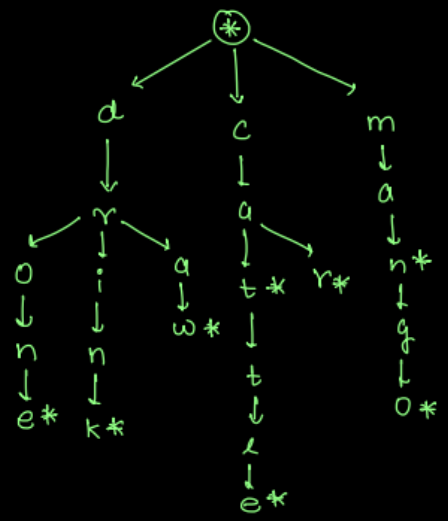
Problem1 : Given an Array of strings and a word check if that word is present in array or not.
arr[drink, draw, drone, cat, cattle, car, mango, man]

https://www.interviewbit.com/snippet/6b52473a91ef2c4dfae5/



```
class Node {
    boolean eow;
    HashMap<character, Node> map;
}
```

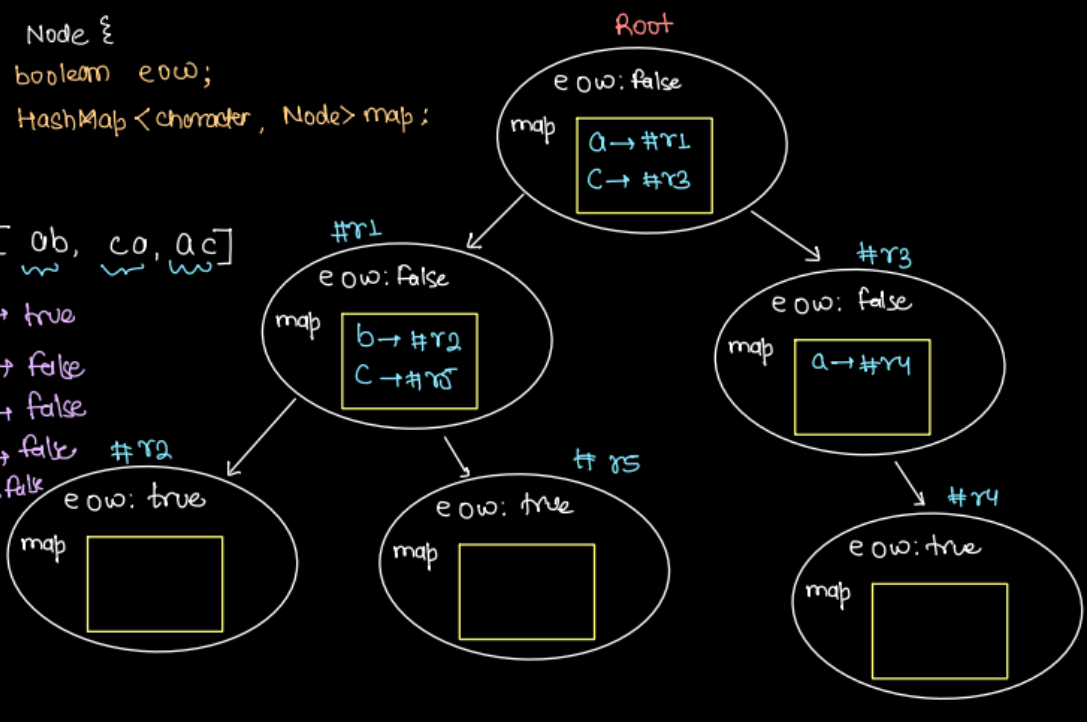
arr: [drink, draw, drone, cat, cattle, car, mango, man]



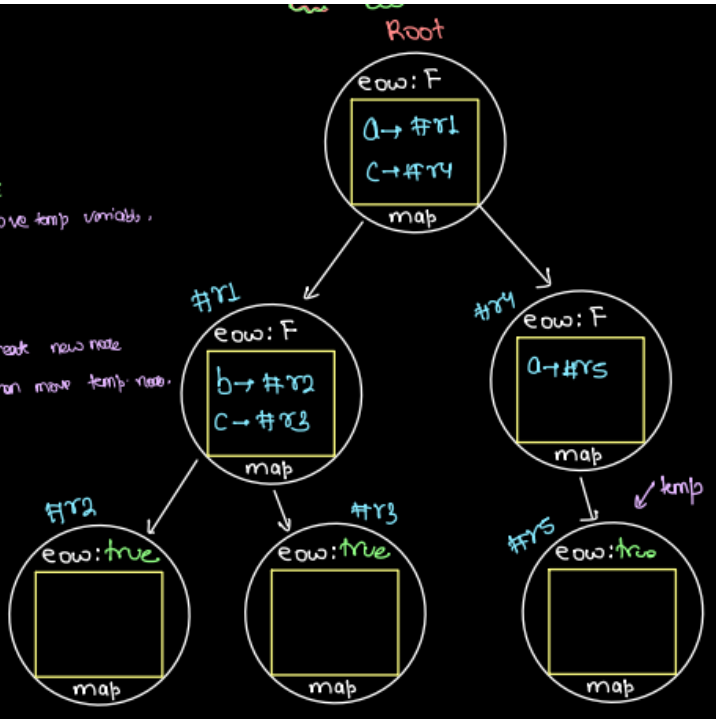

```
class Node {
    boolean eow;
    HashMap<character, Node> map;
}
```

arr[ab, co, ac]

ac → true
cb → false
a → false
cab → false
cable → false



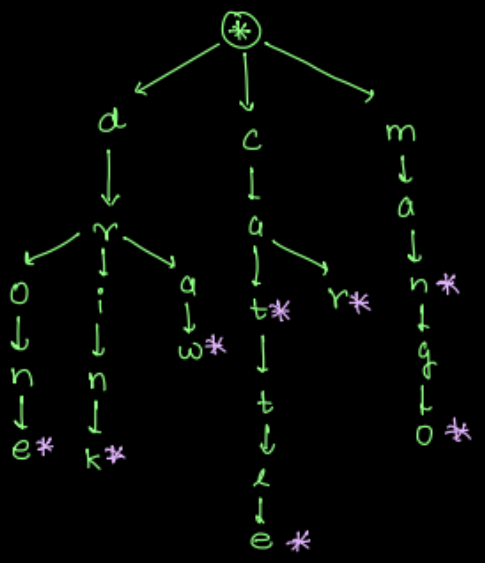

```
void insert( Node root, String word) {
    Node temp = root;
    for(int i=0; i< word.length(); i++){
        char ch = word.charAt(i);
        if( temp.map.containsKey(ch)) {
            // if character is available move temp variable.
            temp = temp.map.get(ch);
        }
        else {
            // if char is not available, create new node
            and store it in the map, then move temp node.
            Node nn = new Node();
            temp.map.put(ch, nn);
            temp = nn;
        }
    }
    temp.eow = true;
}
```



move toward searching

* Assume trie is prepared
* we are here for SEARCHING

words → mang → false
can → false
man → true
drift → false




```
boolean search(Node root, String word) {
    Node temp = root;
    for(int i = 0; i < word.length(); i++) {
        char ch = word.charAt(i);
        if (temp.map.containsKey(ch)) {
            // if that ch is available, move on that ch's node
            temp = temp.map.get(ch);
        }
        else {
            return false;
        }
    }
    return temp.eow;
}
```

Slightly better

T.C: O(n*l)

S.C: O(n*l)

```java
import java.util.*;

class Main {

    // Node class
    public static class Node {
        boolean eow;
        HashMap<Character, Node> map;

        Node() {
            this.eow = false;
            this.map = new HashMap<>();
        }
    }

    public static void insert(Node root, String word) {
        Node temp = root;
        for(int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            // search for ch in temp.map
            if(temp.map.containsKey(ch)) {
                // if available, move toward that node
                temp = temp.map.get(ch);
            } else {
                // if not available, create a node, them move
                Node nn = new Node();
                temp.map.put(ch, nn);
                temp = nn;
            }
        }
        temp.eow = true;
    }

    public static boolean search(Node root, String word) {
        Node temp = root;
        for(int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            // search for ch in temp.map
            if(temp.map.containsKey(ch)) {
                // if available, move toward that node
                temp = temp.map.get(ch);
            } else {
                // if not available, return false
                return false;
            }
        }
        return temp.eow;
    }

    public static void demo() {
        String[] arr = {"drink", "draw", "drone", "cat",
                        "cattle", "car", "mango", "man"};

        // preparation of TRIE
        Node root = new Node();
        for(String word : arr) {
            insert(root, word);
        }

        // Searching
        String str = "cattle";
        System.out.println(search(root, str));


    }
```

Count of Strings with given Prefix

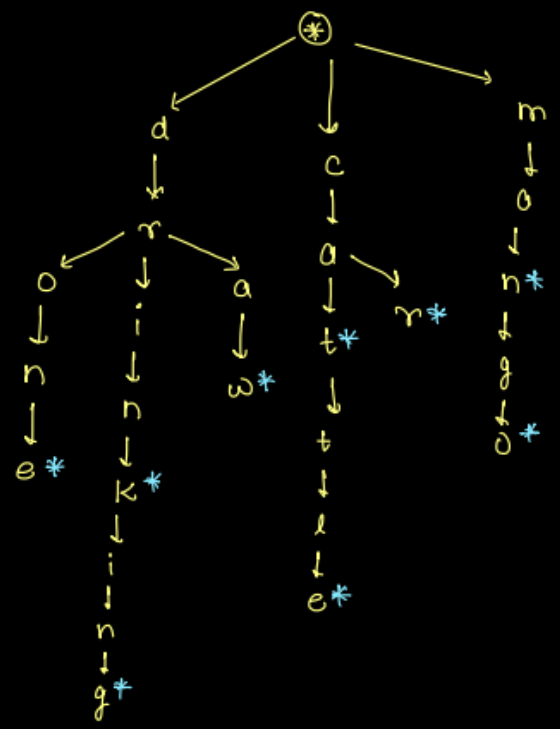
Given an array of Strings and a word, check how many strings have prefix equal to given word. arr:[drink,draw, drone, cat, cattle, car, mango, man,drinking]

https://www.interviewbit.com/snippet/1ff06b0a7c76fec81e72/

arr: [drink, draw, drone, cat, Cattle, Car, mango, man, drinking]

words	ans
dr	4
car	1
ca	3
r	0

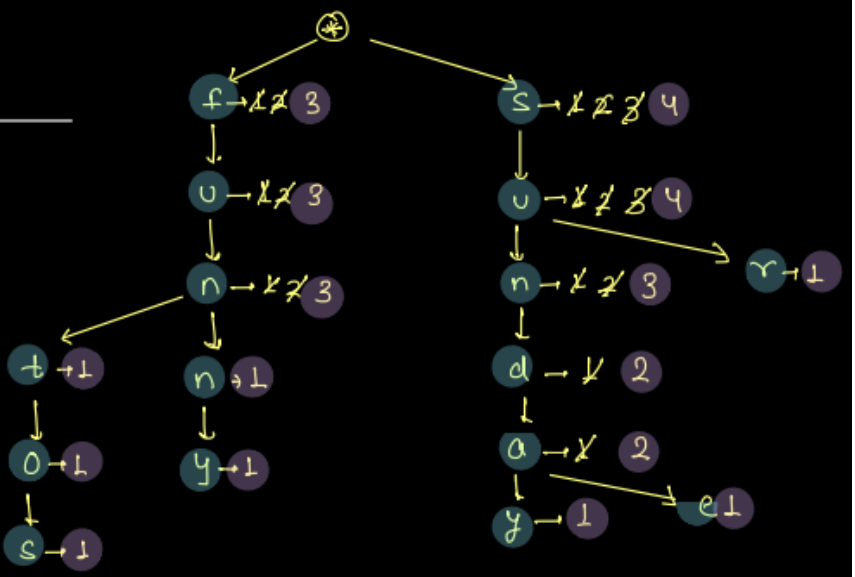
trie is prepared with given Array of String



NOTE: If we want to calculate prefix count in array, End of word (eow) will doesn't matter here. Instead we have to calculate no. of time we travel through any specific node while inserting words in TRIE.

[fun, funny, funtos, sun, sunday, Sur, sundae]

word	ans
sun	3
su	4
fun	3
sum	0
funday	0
sunny	0
shreesh	0



Real life Example:

→ Spelling Checker; [95 82] . . . 1 . . . 2 . . . 3

→ Phonebook →

→ Auto fill or Auto Suggestion.

Count of occurrence

KMP Algorithm.

```java
import java.util.*;

class Main {

    // Node class
    public static class Node {
        int count;
        HashMap<Character, Node> map;

        Node() {
            this.count = 0;
            this.map = new HashMap<>();
        }
    }

    public static void insert(Node root, String word) {
        Node temp = root;
        for(int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            // search for ch in temp.map
            if(temp.map.containsKey(ch)) {
                // if available, move toward that node
                temp = temp.map.get(ch);
                temp.count++;
            } else {
                // if not available, create a node, them move
                Node nn = new Node();
                 temp.map.put(ch, nn);
                 temp = nn;
                 temp.count++;
             }
        }

    }

    public static int searchPrefixCount(Node root, String word) {
        Node temp = root;
        for(int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            // search for ch in temp.map
            if(temp.map.containsKey(ch)) {
                // if available, move toward that node
                temp = temp.map.get(ch);
            } else {
                // if not available, return false
                return 0;
            }
        }
        return temp.count;
    }

    public static void demo() {
        String[] arr = {"drink", "draw", "drone", "cat",
                        "cattle", "car", "mango", "man", "drinking"};

        // preparation of TRIE
        Node root = new Node();
        for(String word : arr) {
            insert(root, word);
        }

        // Searching
        String str = "ca";
        System.out.println(searchPrefixCount(root, str));


    }

}
```

DSA: Heaps - 10 - Aug 2023

- Introduction of Priority Queue
- Kth Smallest Element
- Running Median in an array
- PQ with user defined class

Introduction of Priority Queue ->

There are two types of priority queue

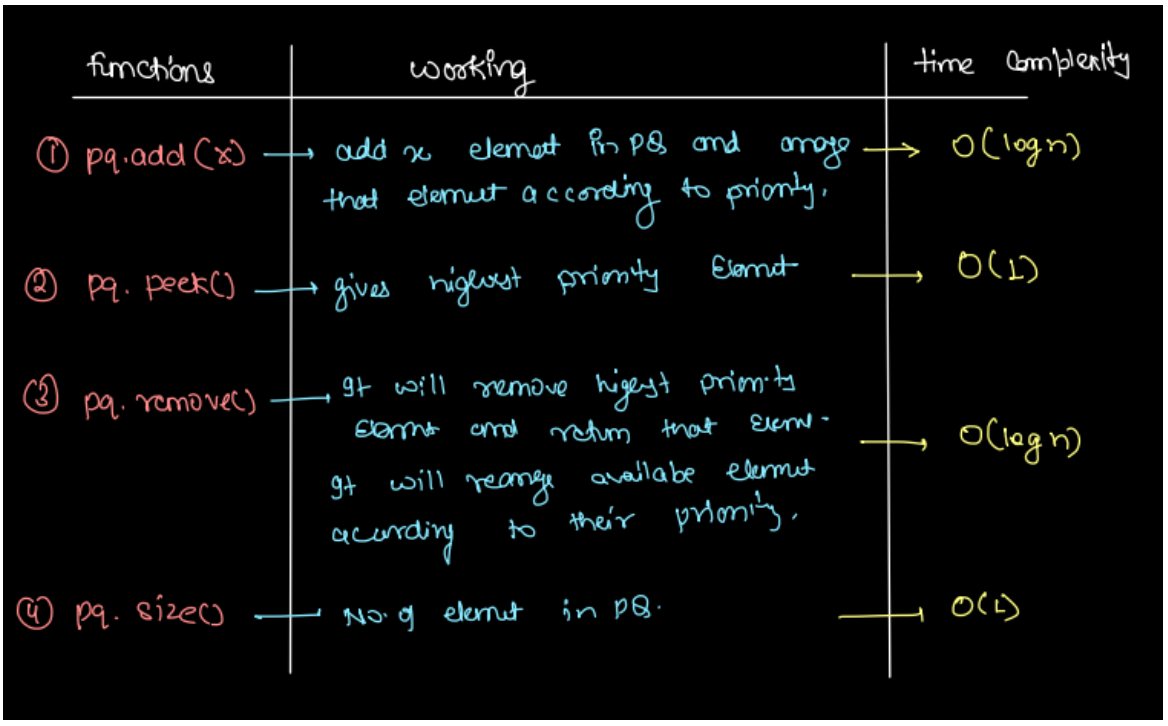
1. **Min Priority Queue** -> We will give min element more priority here - default in java
2. **Max Priority Queue** -> We will give max element more priority

Note -> By default Java have min Pq.

MinPq -> PriorityQueue<Integer> pg = new PriorityQueue<>();
MaxPq -> PriorityQueue<Integer> pg = new PriorityQueue<>(Collections.reverseOrder());

Note -> It is use for integer if there is any custom type then we have to pass customComparator

```
pq.add(19);
pq.add(10);
sopln(pq.peek());
pq.remove();
```



Problem1: Kth Smallest element -> Given an Array A[] and a value K. Find Kth smallest element from it.

A = [8, 4,10,5,11,9,7,6,14,1]

https://www.interviewbit.com/snippet/d2e9ea0bb7660eb4f7dc/

Main Logic ->

We will address this using a max priority queue. Initially, we will insert the first 'k' elements into the queue. In a subsequent loop, for each element from the (k+1)th position to the end of the array, we will compare the element with the maximum element in the queue (peek). If the peek element is greater than the current array element, we will replace the peek element with the current array element.

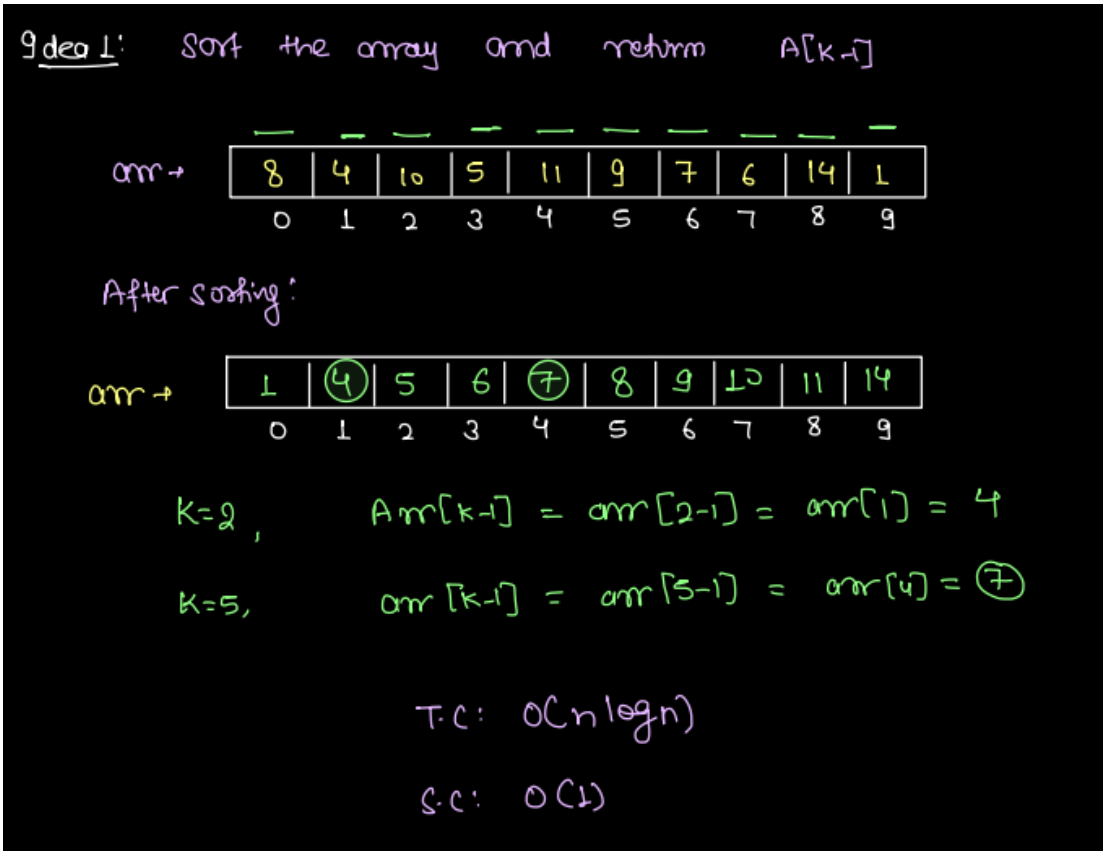
Here are the steps to achieve this:

Define a max priority queue using `Collections.reverseOrder`**.**
Iterate through the array from index 0 to < kth element and insert each array item into the queue.
Run another loop from the kth element to the length of the array:

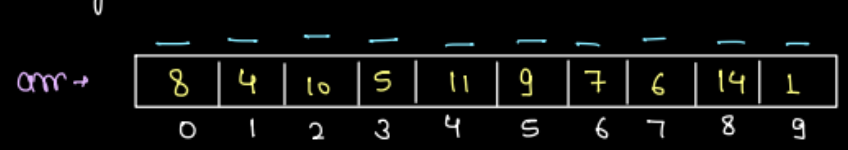
If the maximum element in the queue (peek) is greater than the current ith element, remove the peek element and insert the ith element into the queue.

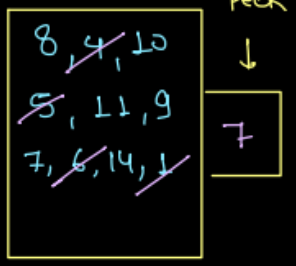
After the loop, return the maximum element (peek) of the queue as the answer.

In summary, this approach ensures that the queue contains the k largest elements from the initial portion of the array. As we iterate through the rest of the array, we maintain the k largest elements in the queue by replacing smaller elements. Finally, the maximum element (peek) of the queue will represent the kth largest element in the entire array.



Idea 2: Using PQ





Remove
1st → 1
2nd → 4
3rd → 5
4th → 6

* add all Elements in min PQ.
* Remove K-1 Elements.
* Peek element is Kth Smallest Elem.

min PQ → Default → 5th Smallest is peek of pq.

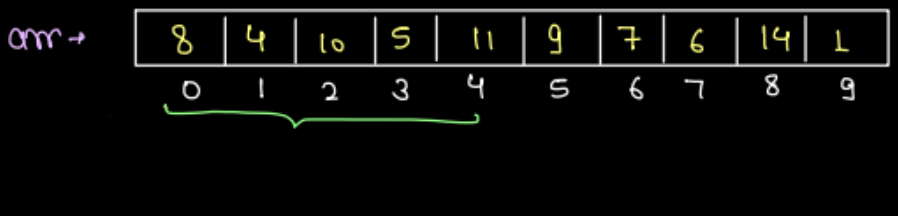
Time Complexity: nlogn + Klogn ≡ nlogn

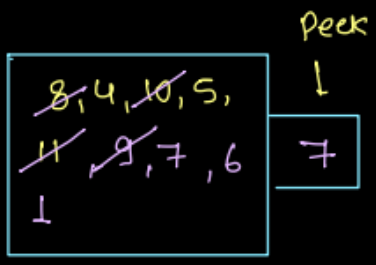
Space Complexity: O(n)

Improvised version:

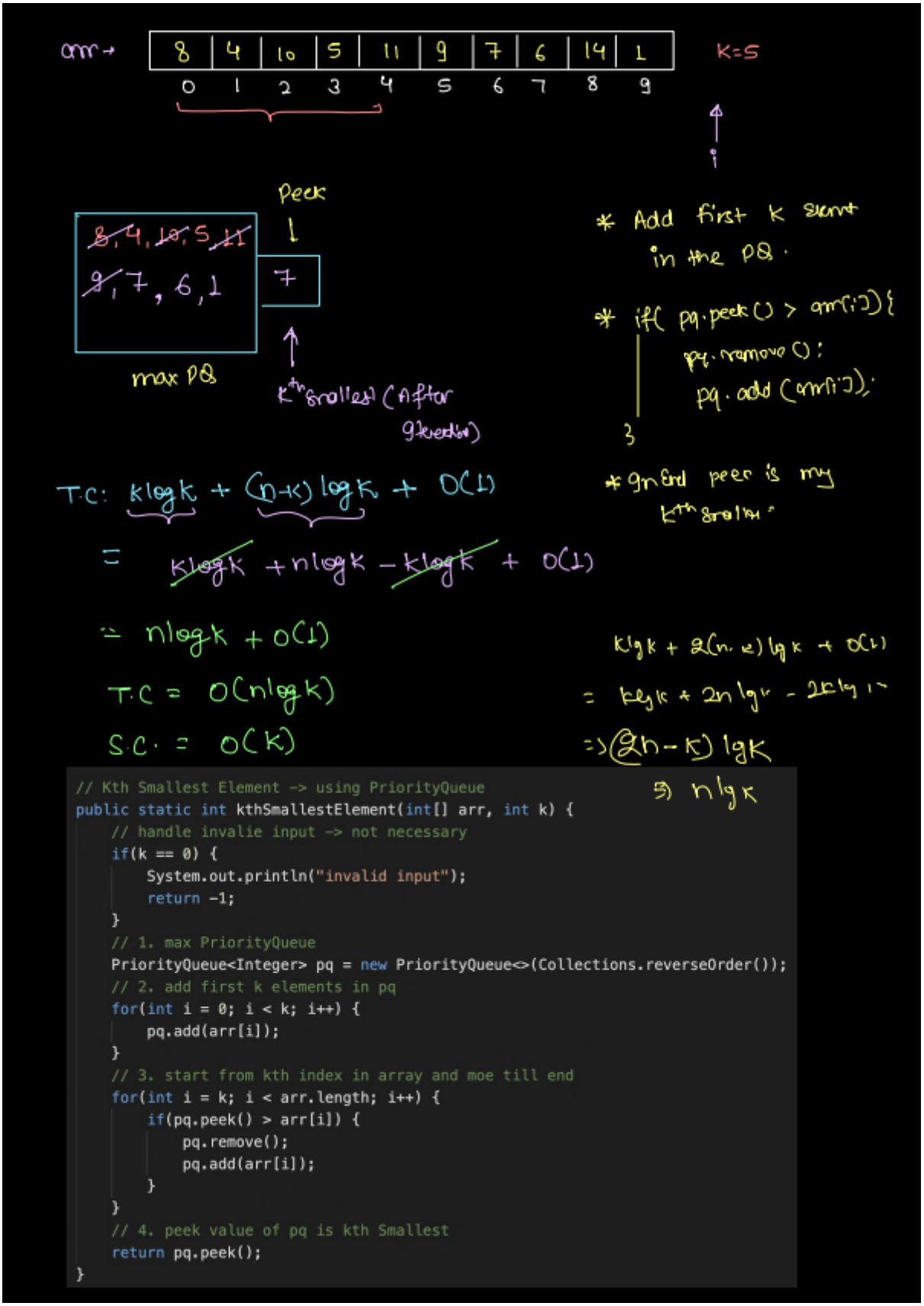
Idea 3: using max PQ [do not add more than K elems in PQ]

K=5

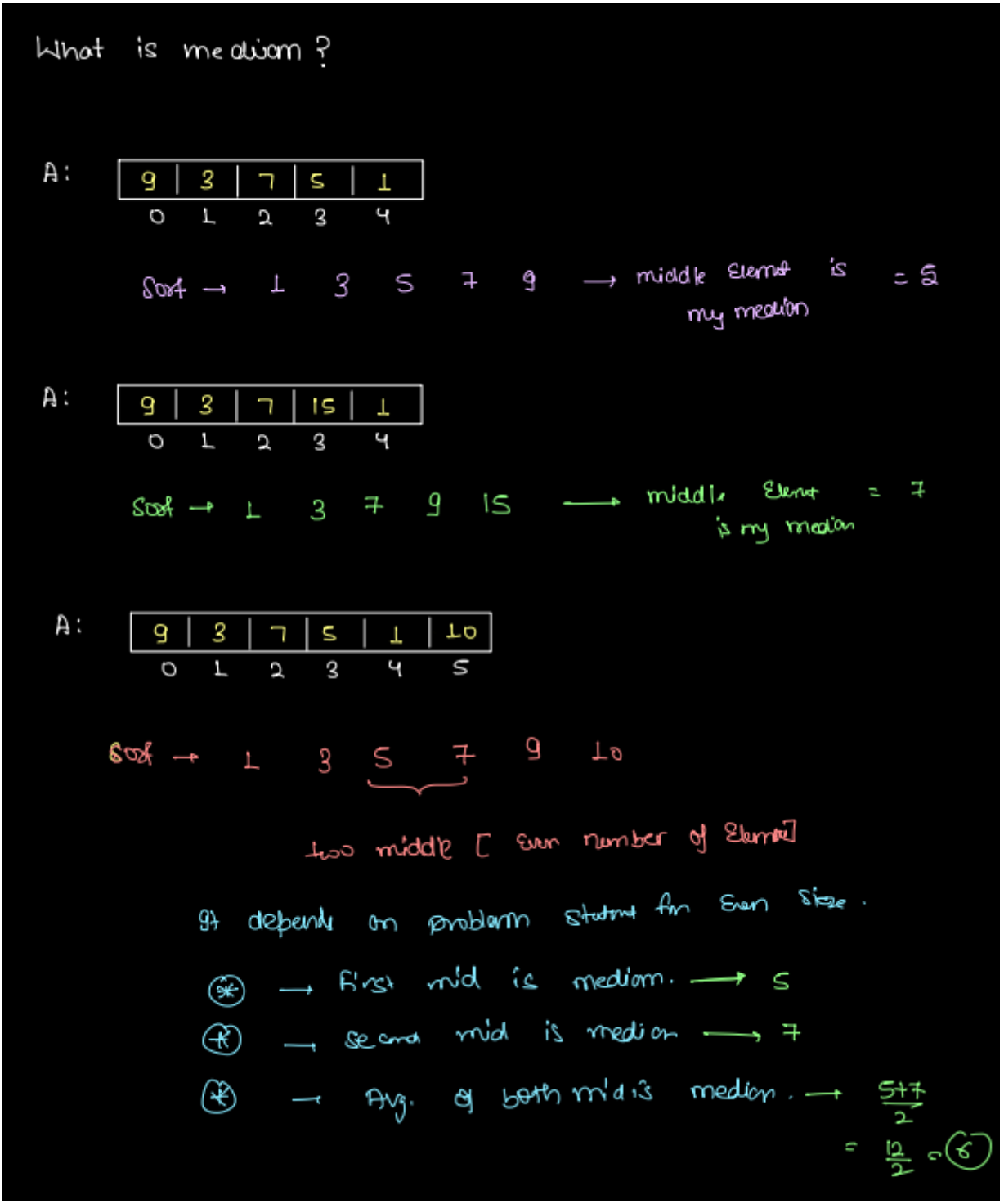




* Add first K elems in the PQ.
* if(pq.peek() > arr[i]){
  pq.remove();
  pq.add(arr[i]);
  }
* In End peek is my Kth smallest.



What is median -> Median is the middle value after sorting an array. If the array has an odd length, the median is the value at position length / 2. If the array has an even length, there are two possible medians: either the average of the two middle values (left and right), or we can choose one of the two middle values based on the problem's context.



Problem 2 -> Running Median - Given an array A[]. at every index find out the median till now.
A = [5,3,23,11,20,25,17,7]

https://www.interviewbit.com/snippet/2bf70309d1aedf5826b1/

Main logic ->

Summary: We'll utilize two priority queues: a max queue as the left queue and a min queue as the right queue. By iterating through the array, we'll maintain the median of the elements seen so far. If the sizes of the queues are equal, we'll add elements to the right queue and maintain the median using the left queue's maximum value. If the sizes differ, we'll add elements to the left queue, balance the queues, and calculate the median using the average of the left and right queue's maximum values.

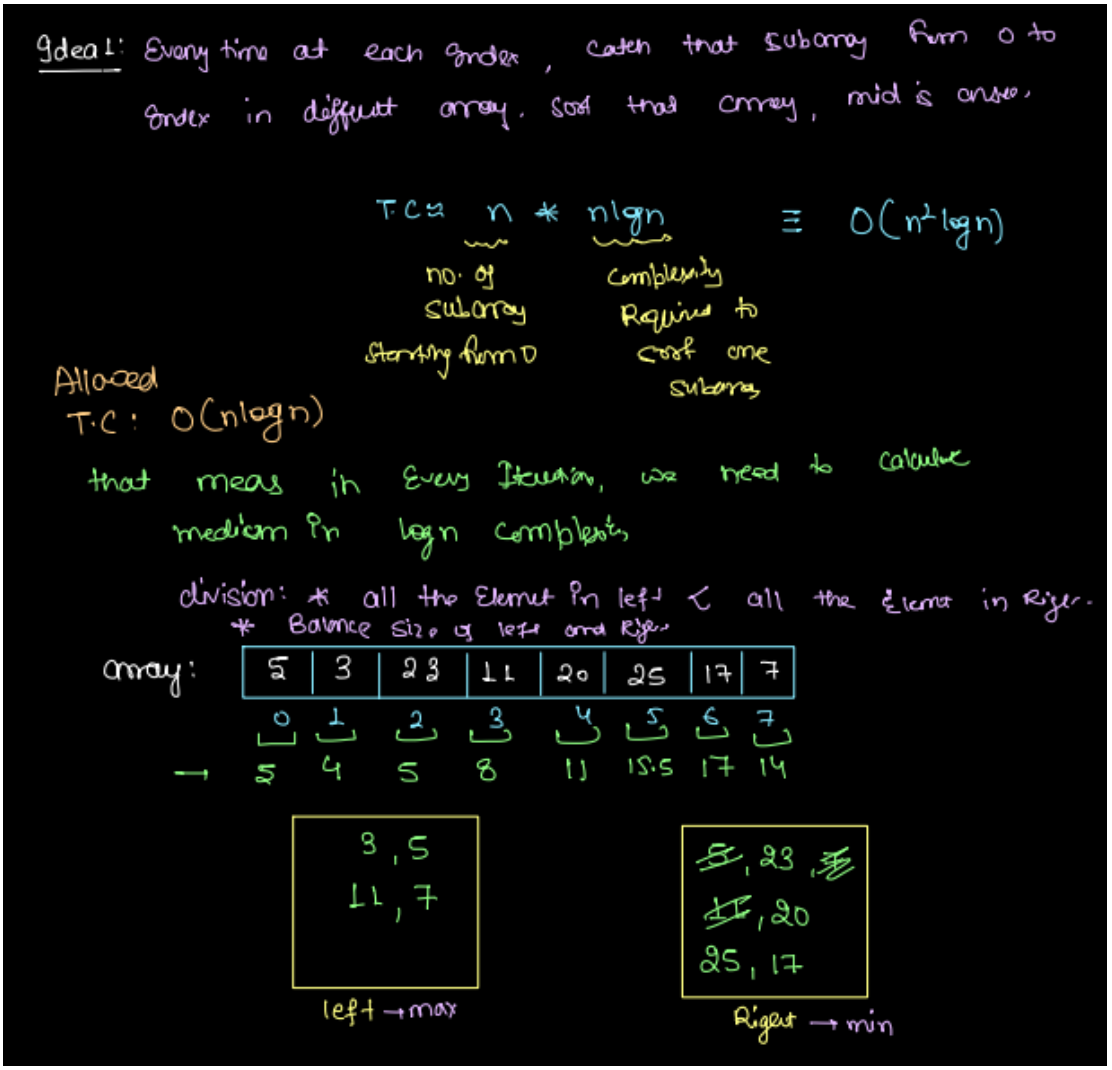
Steps:

Create two priority queues: a max priority queue for the left queue and a min priority queue for the right queue. Initialize an array `ans` of the same length as the input array.
Iterate over the array from index 0 to less than the length of the array.

- **If the sizes of the left and right queues are equal:**
- **Add the current array element to the right queue.**
- **Remove an element from the right queue and add it to the left queue.**
- **Assign the maximum element (peek) of the left queue to the corresponding index in the `ans` array.**
- **If the sizes of the left and right queues are not equal:**
- **Add the current array element to the left queue.**
- **Remove an element from the left queue and add it to the right queue.**
- **Calculate the median by taking the average of the maximum elements (peeks) from both the left and right queues, and assign it to the corresponding index in the `ans` array.**

Return the `ans` array as the final answer.

By following these steps, we can efficiently compute and maintain the median of the elements encountered while iterating through the array.



Scenario: Even number of Element →

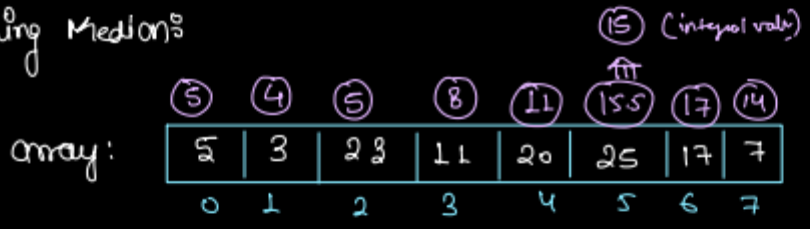
$$\text{median} = \frac{\text{Select one from left (max)} + \text{Select one from right (min)}}{2}$$

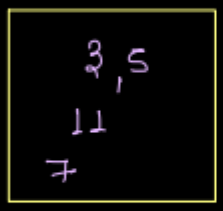
Odd number of Element -

→ We will add Extra Element in left PQ.

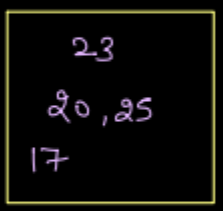
→ That means Extra Element of left PQ is median of array till now.

Running Median:





left → max

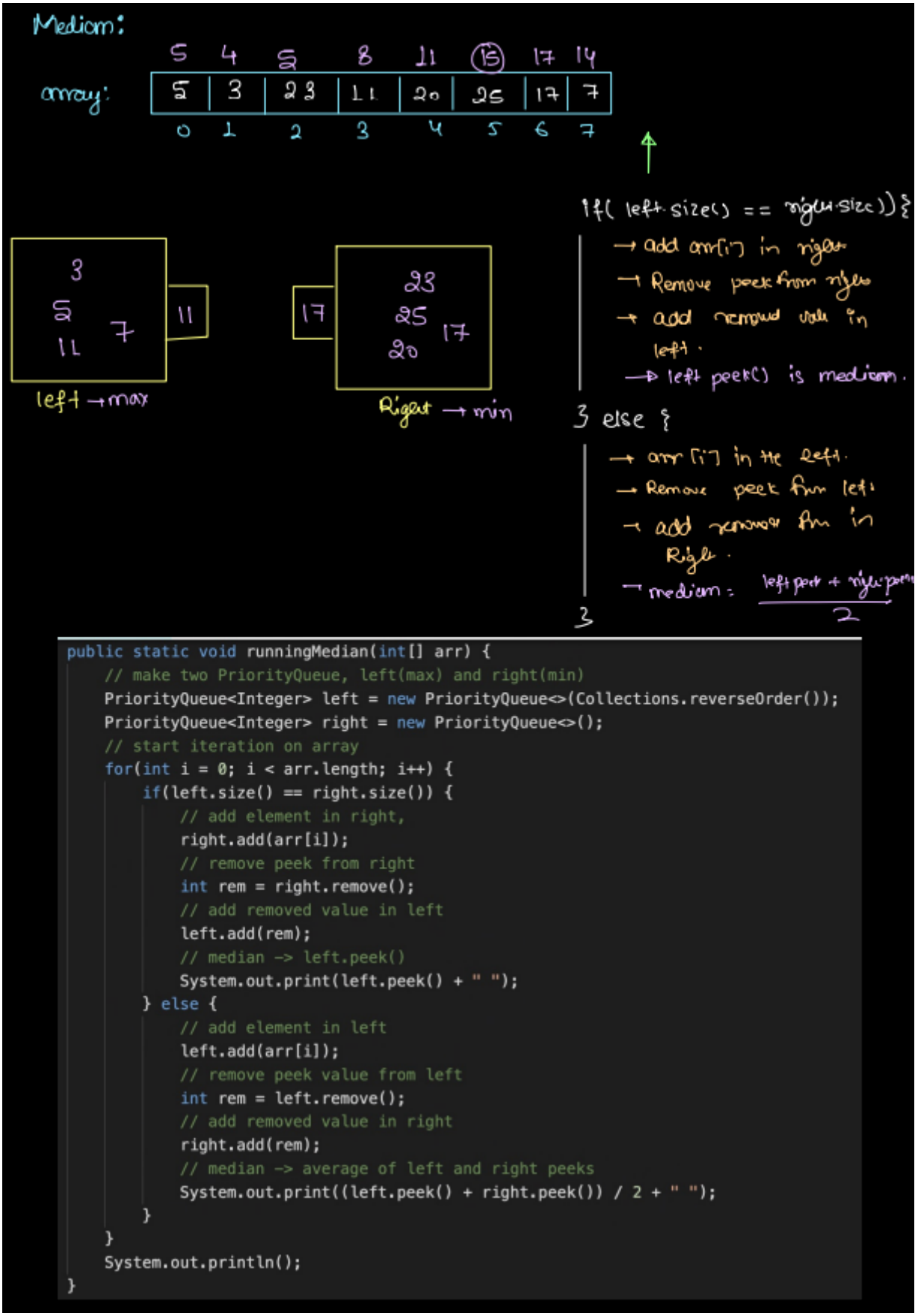


Right → min

if(left.size() == right.size)
→ add element in right
→ remove peek from right add it in left.

else {
add arr[i] in left
remove peek from left and add it in right.
}

3

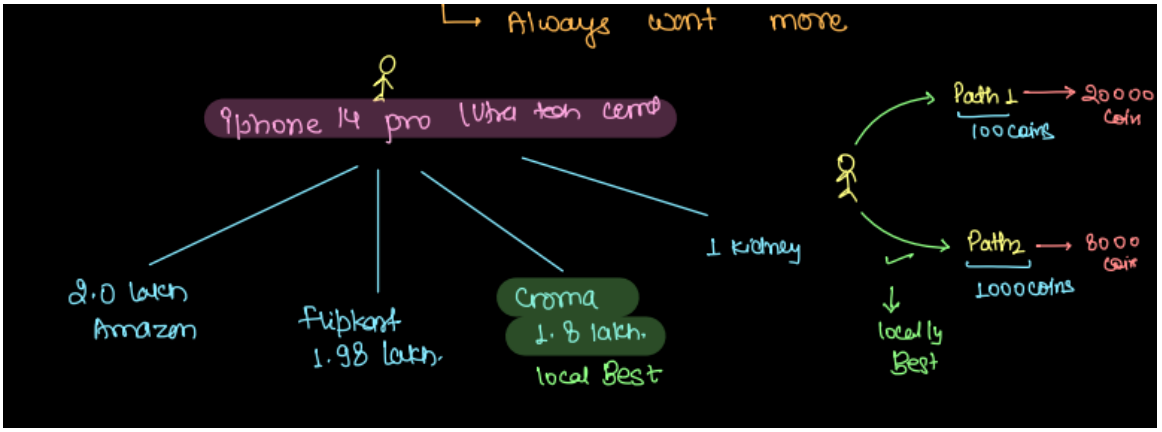



```java
public static void runningMedian(int[] arr) {
    // make two PriorityQueue, left(max) and right(min)
    PriorityQueue<Integer> left = new PriorityQueue<>(Collections.reverseOrder());
    PriorityQueue<Integer> right = new PriorityQueue<>();
    // start iteration on array
    for(int i = 0; i < arr.length; i++) {
        if(left.size() == right.size()) {
            // add element in right,
            right.add(arr[i]);
            // remove peek from right
            int rem = right.remove();
            // add removed value in left
            left.add(rem);
            // median -> left.peek()
            System.out.print(left.peek() + " ");
        } else {
            // add element in left
            left.add(arr[i]);
            // remove peek value from left
            int rem = left.remove();
            // add removed value in right
            right.add(rem);
            // median -> average of left and right peeks
            System.out.print((left.peek() + right.peek()) / 2 + " ");
        }
    }
    System.out.println();
}
```

DSA: Greedy Algorithms - 12 - Aug 2023

- Introduction to greedy algorithm
- Fractional Knapsack
- Activity Selection
- Job Scheduling

Introduction to greedy algorithm -> Always want more. here we will consider locally best means whatever second is big we will compair with that but choosing local best is always not a good options. This is not a algorithm it is technique.



Problem1: Fractional knapsack - We can consume K kg of food item. Find max protein we can get.

Note: Eating any integral amount of an item is also allowed.

https://www.interviewbit.com/snippet/fcb54f2f8b1c2b042318/

Main logic ->

Steps

- Define a class called "PairArray" comprising three integer variables: wt, protein, and ppk. Create a constructor for this class, assign values to these variables, and calculate ppk by dividing protein by kg.
- Initialize an integer variable "n" to represent the length of the wt array. Also, create another integer variable "totalProtein" and set it to 0.
- Create an array named "pairArray" of type "Pair" and iterate through a loop until it reaches "n". Inside the loop, create a Pair object by passing two values: the current index's wt and protein.
- Utilize a custom Comparator to sort the "pairArray" based on ppk in descending order.
- Iterate through a loop until "n" and check a condition: if wt is less than or equal to a user-defined limit, add the protein value to the "totalProtein" variable and subtract "wt" from the limit. Otherwise, multiply ppk by the limit, add the result to "totalProtein," and set the limit to 0. Break the loop after this step.
- Return the final value of "totalProtein" as the result.

Note Some time because of constraint we have to change custom comparator

```
// sort pair array use comparator


Arrays.sort(pairArr, new Comparator<Pair>(){
@Override
public int compare(Pair ppk1, Pair ppk2){
return Double.compare(ppk2.ppk, ppk1.ppk);
}

Pair(int val, int wt){
this.val = val;
this.wt = wt;
this.ppk = (double) val / wt;
}
}
});
```

Problem Constraints

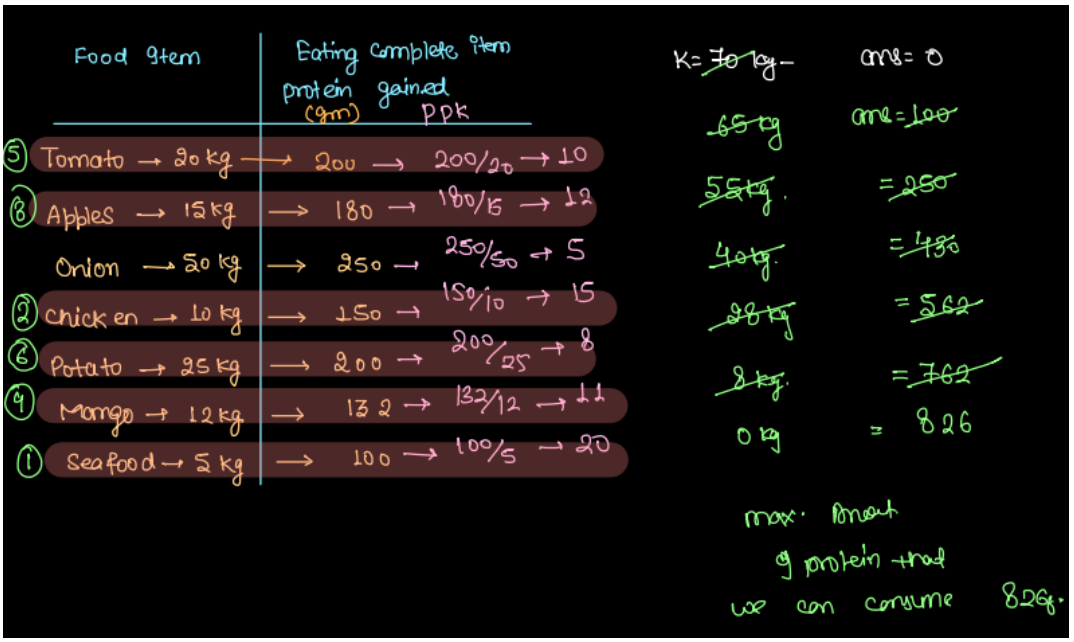
$1 <= N <= 10^5$

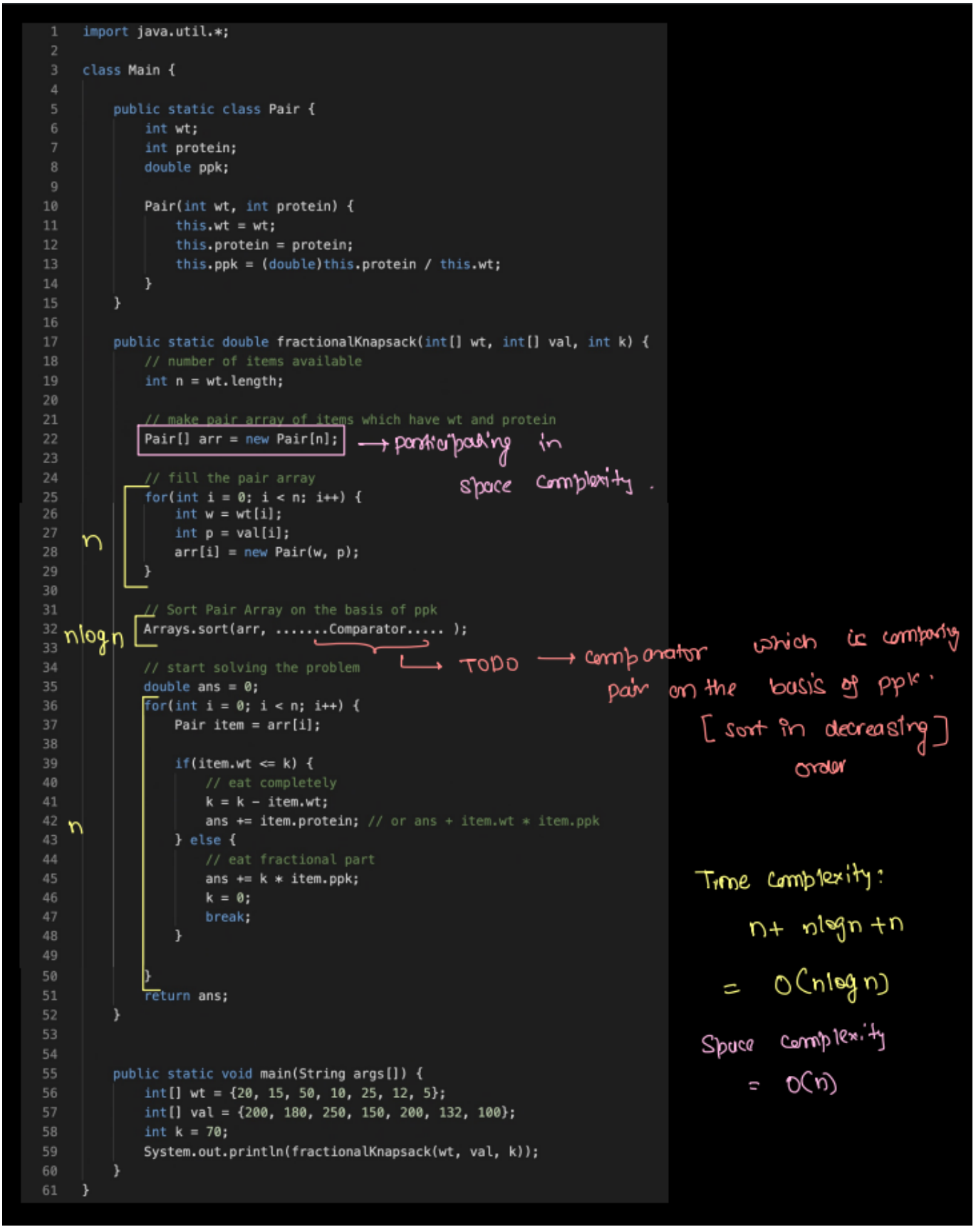
$1 <= A[i], B[i] <= 10^3$

$1 <= C <= 10^3$

```
class Pair{
int val;
int wt;
double ppk;

Pair(int val, int wt){
this.val = val;
this.wt = wt;
this.ppk = (double) val / wt;
}
}

public class Solution {
public static void printPairArray(Pair[] pairArr) {

for (Pair item : pairArr) {
System.out.println("Weight: " + item.wt + ", Protein: " + item.val + ", PPK: " + item.ppk);
}
}
public int getFractional(int[] values, int[] weights, int capacity){
int n = values.length;
Pair[] pairArr = new Pair[n];
double maxTotal = 0;
double remainCap = capacity;
// create pair array
for(int i = 0; i < n; i++){
pairArr[i] = new Pair(values[i], weights[i]);
}
```

```
// sort pair array use comparator


Arrays.sort(pairArr, new Comparator<Pair>(){
@Override
public int compare(Pair ppk1, Pair ppk2){
return Double.compare(ppk2.ppk, ppk1.ppk);
}
});

// printPairArray(pairArr);
// Itrate pair array and fill knapsack
for(int i = 0; i < n; i++){
Pair item = pairArr[i];
if(item.wt <= remainCap){
maxTotal += item.val;
remainCap -= item.wt;
}else{
maxTotal += (item.ppk * remainCap);
remainCap = 0;
break;
}
}

return (int)(maxTotal * 100);
}

public int solve(int[] A, int[] B, int C) {

return getFractional(A, B, C);

}
}
```





Problem 2: Activity Selection -> Find maximum number of task we can do:

Note: 1. On Start of a task, we need to complete it first.
2. At any point of time we can do single task.

https://www.interviewbit.com/snippet/aefaf2c1c9b586b67036/

Main logic ->
Steps

- Define a class named "TimePair" with two integer variables: sTime and eTime. Initialize these variables within the constructor.

- Create an array named "timePairArr" and an integer variable "n" to represent the length of the sTime array.

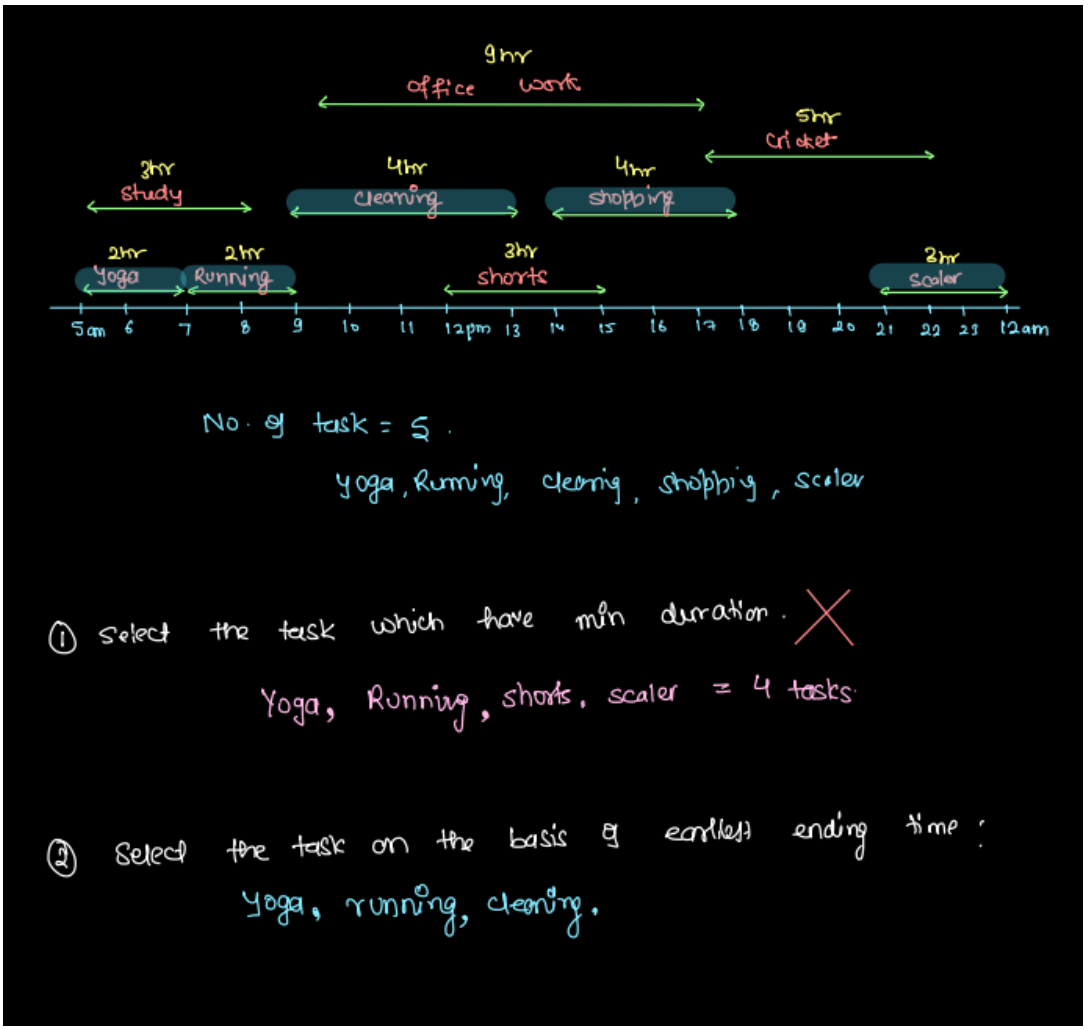
- Utilize a for loop to iterate through the range of "n." Inside the loop, create an object of the TimePair class using the values eTime[i] and sTime[i], and add it to the "timePairArr."

- Initialize two variables: "activityCount" with a value of 1, and "activityEndTime" with the value of the first element in the "timePairArray."

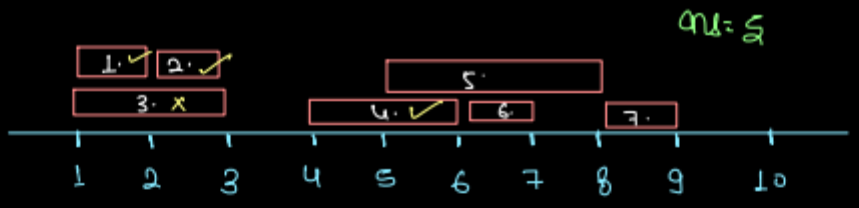
- Sort the "timePairArr" using a custom Comparator based on the eTime values in ascending order.

- Run a loop until "n" and create a variable named "item" to represent the current element in the "timePairArray." Then, use a conditional statement: if item.sTime is greater than or equal to "activityEndTime," increment the "activityCount" variable.

- Finally, return the value of "activityCount" as the result.



Selection of Activity on the basis of ending point:



Selected Activity:

⇒ 1 → 1-2

⇒ 2 → 2-3

⇒ 4 → 4-6

⇒ 6 → 6-7

⇒ 7 → 8-9

task →	0	1	2	3	4
Start →	4	5	3	9	2
Ending →	7	7	5	10	8

make a pair [start, end] array, then sort on the basis of end pt.

Arrays of pairs:

st = 4, e = 7	st = 5, e = 7	st = 3, e = 5	st = 9, e = 10	st = 2, e = 8

Sort the array on the basis of ending point [sort in increasing order]

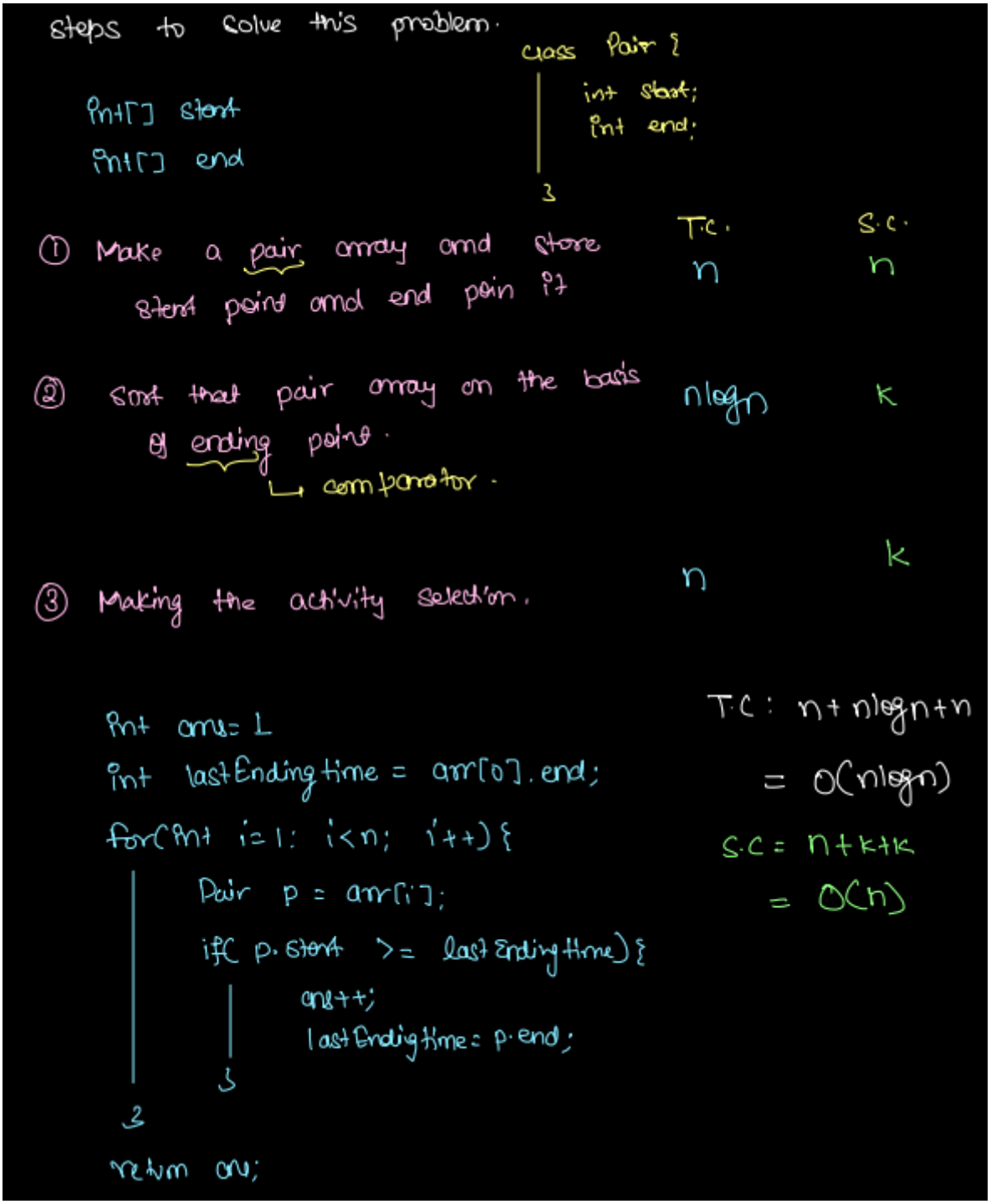
st = 3, e = 5	st = 5, e = 7	st = 4, e = 7	st = 2, e = 8	st = 9, e = 10

last Selected activity = ~~3-5~~ ~~5-7~~ 9-10

curr = 9-10

count = ~~1~~ & 2

2 is ans.



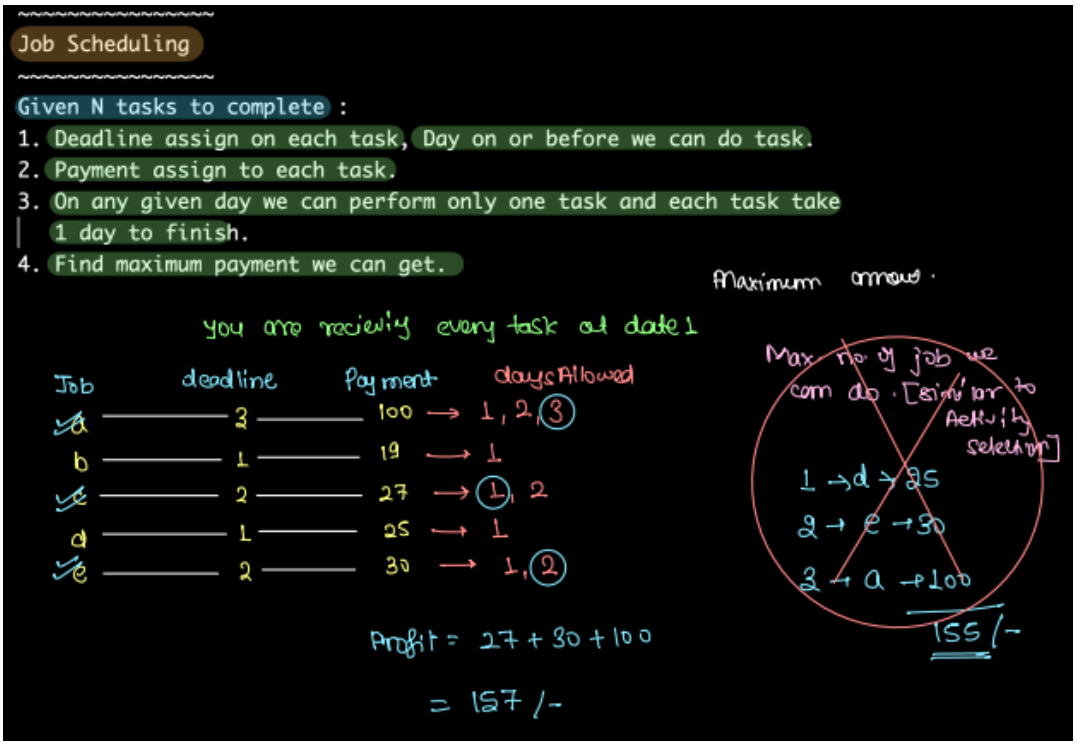
Problem 3 -> Job Scheduling - Given N tasks to complete

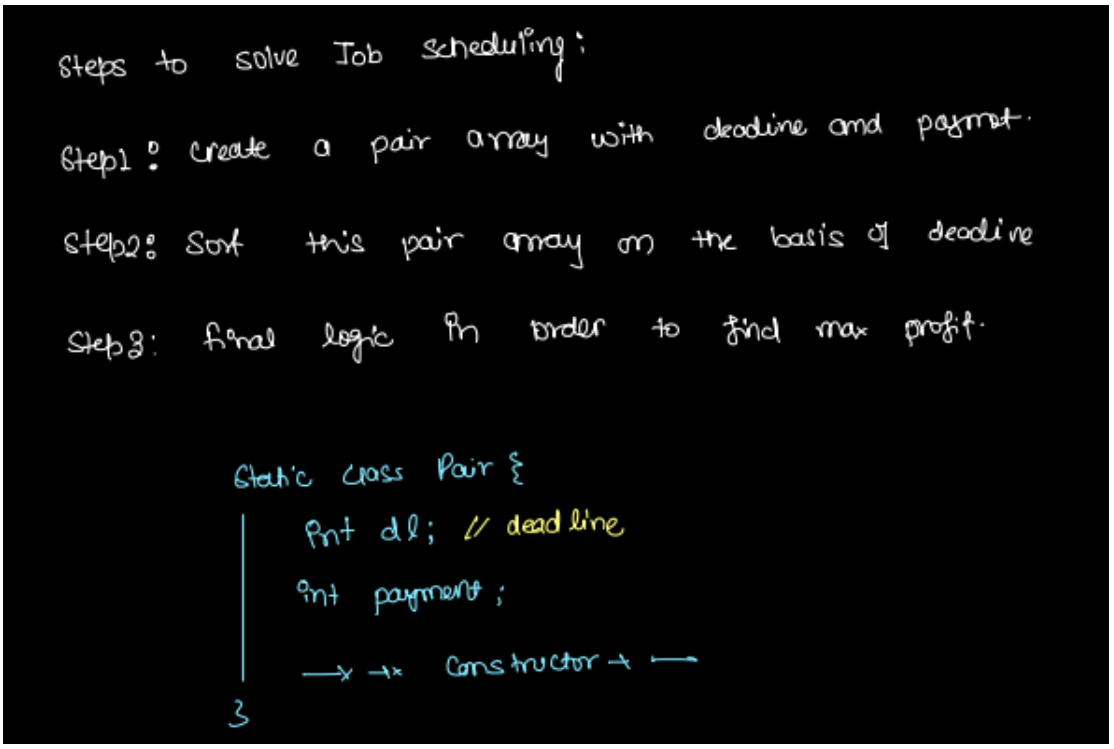
https://www.interviewbit.com/snippet/5e913edc651a3c8c0dc9/

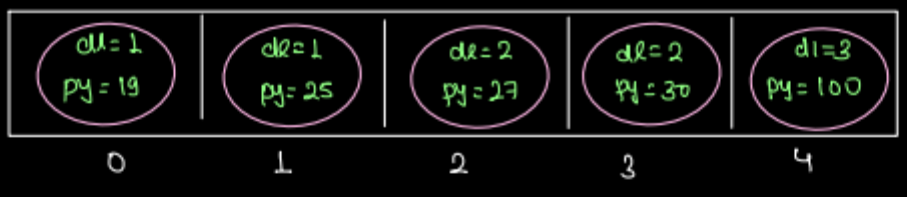
Main logic ->
Steps

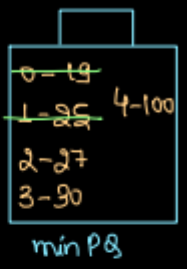
- Create a class named "Pair" containing two integer variables: "deadline" and "profit." Initialize these variables within the constructor.
- Implement a function called "calculateProfit."
- Declare an array named "pairArr," and define an integer variable "n" to represent the length of the "deadline" array.
- Employ a for loop to iterate over the range of "n." Inside the loop, instantiate an object of the Pair class using the values "deadline[i]" and "profit[i]," and then add it to the "pairArr."

- Initialize a priority queue.
- Sort the "pairArr" using a custom Comparator that is based on the "deadline" values in ascending order.
- Execute a loop until "n" and create a variable named "item" to represent the current element in the "pairArray." Next, use a conditional statement: if the size of the queue is less than the "ith deadline," then add "item.profit" to the queue. Otherwise, if the peek value of the queue is less than the current profit, remove the peek element and add the current profit.
- Initialize a variable named "totalProfit" and set it to 0.
- Iterate through the queue until its size, and for each element, remove it from the queue and add its value to the "totalProfit" variable.
- Finally, return the value of "totalProfit" as the result.










```
for(int i=0; i< n; i++){
    Pair p = arr[i];
    if( p.dl > pq.size()) {
        pq.add (p.payment);
    }
    else {
        // Replace the previous job if
        current is better.
        if( pq.peek() < p.payment) {
            pq.remove();
            pq.add (p.payment);
        }
    }
}
```

3

Sum of all number of entries in priority Que. is max profit

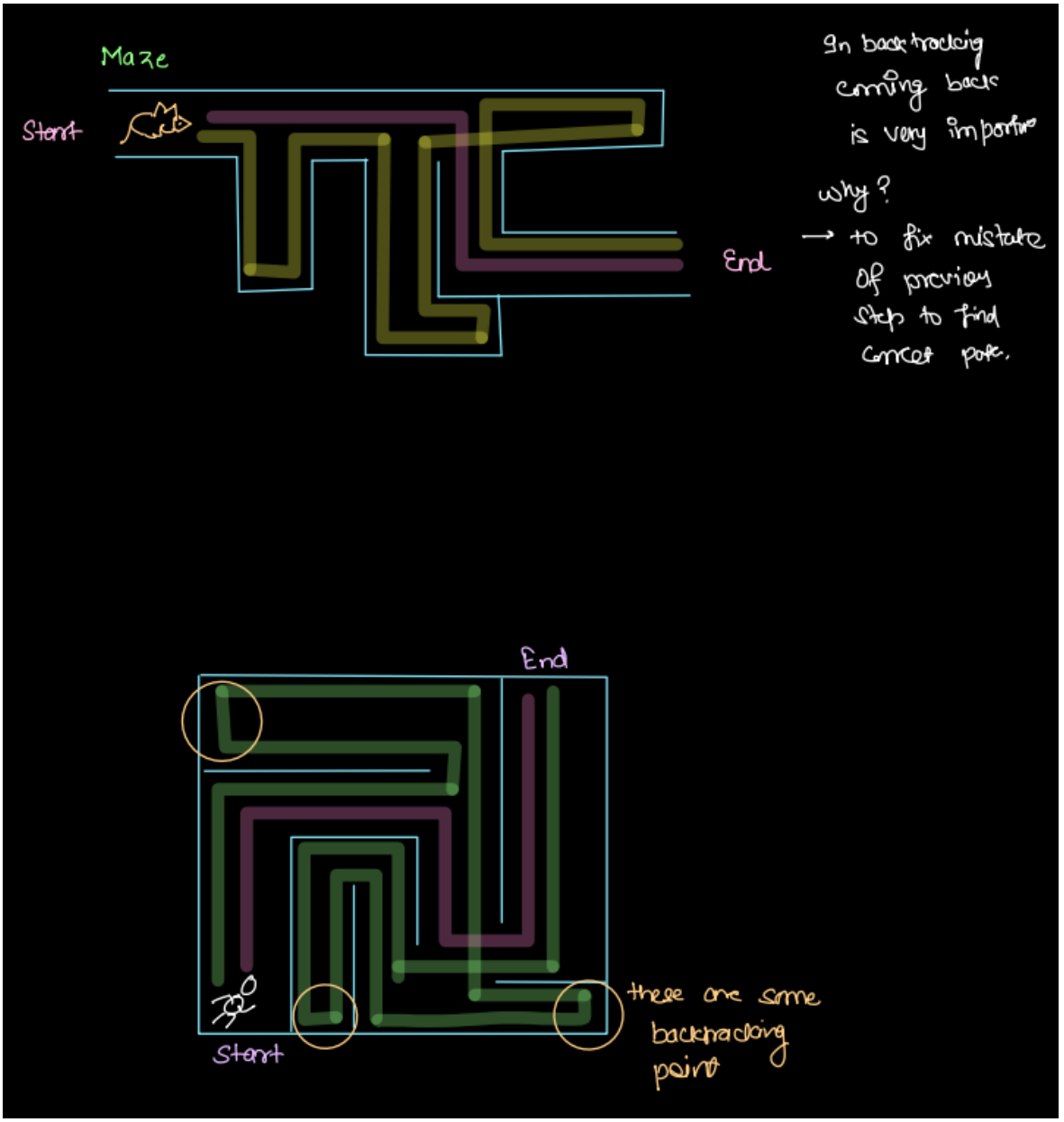
```
int sum=0;
while( pq.size()>0) {
    sum += pq.remove();
}
return sum.
```

DSA: Backtracking - 18 - Aug 2023

- Introduction of Backtracking
- All numbers using 1 and 2
- SubSet Sum
- N Queens

Introduction of Backtracking ->

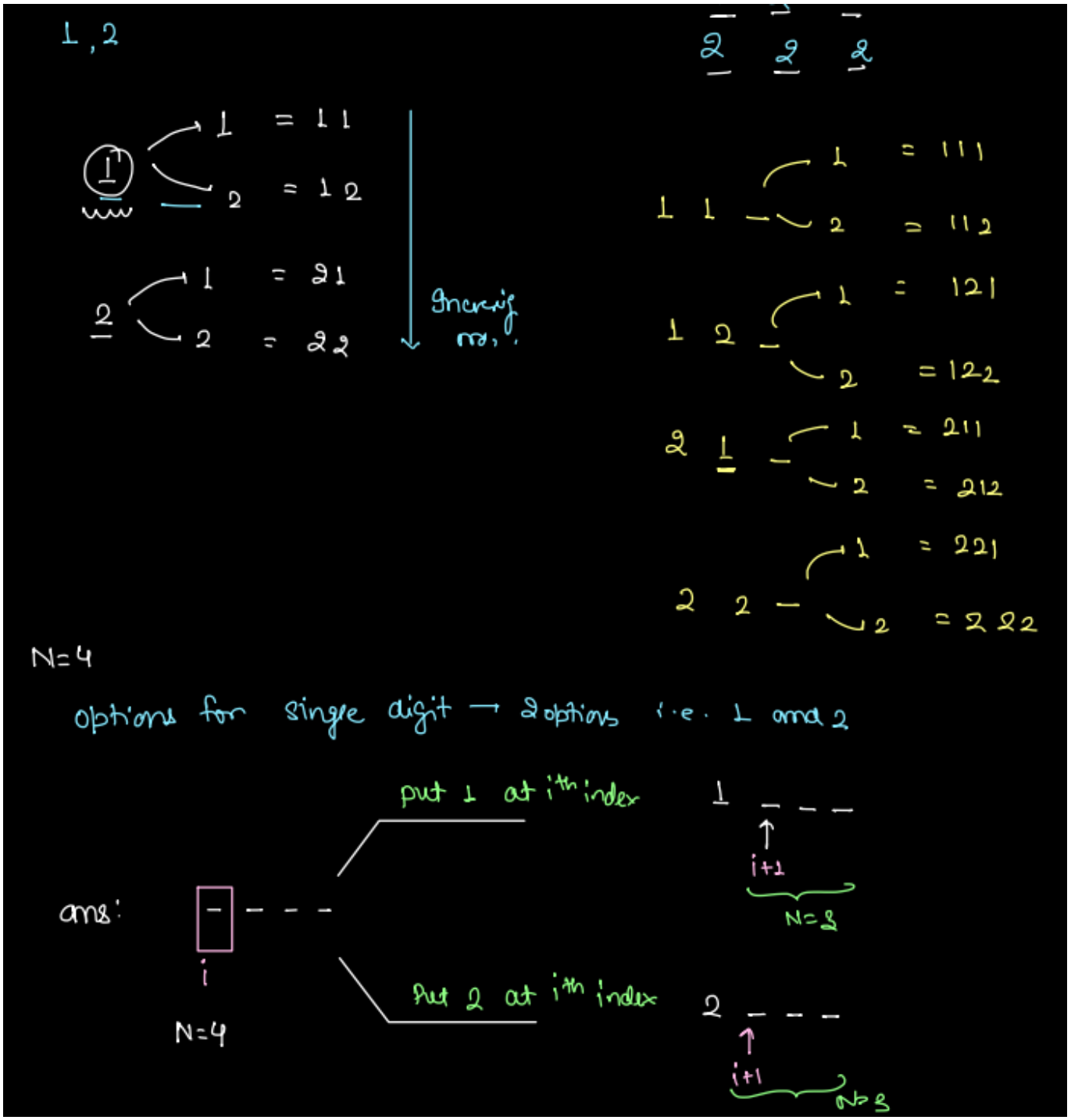
We will employ a backtracking approach to solve the problem. When we encounter a dead end, we will come back on same path to rectify our errors and then proceed with seeking the solution. We will repeat this iterative process until we arrive at the correct solution. To achieve this, we will adopt a recursive methodology.



All Numbers(Using 1 and 2)

Problem1: Given N. Print all N digit numbers formed by 1 and 2 in increasing order of numbers
N = 2, N = 3;

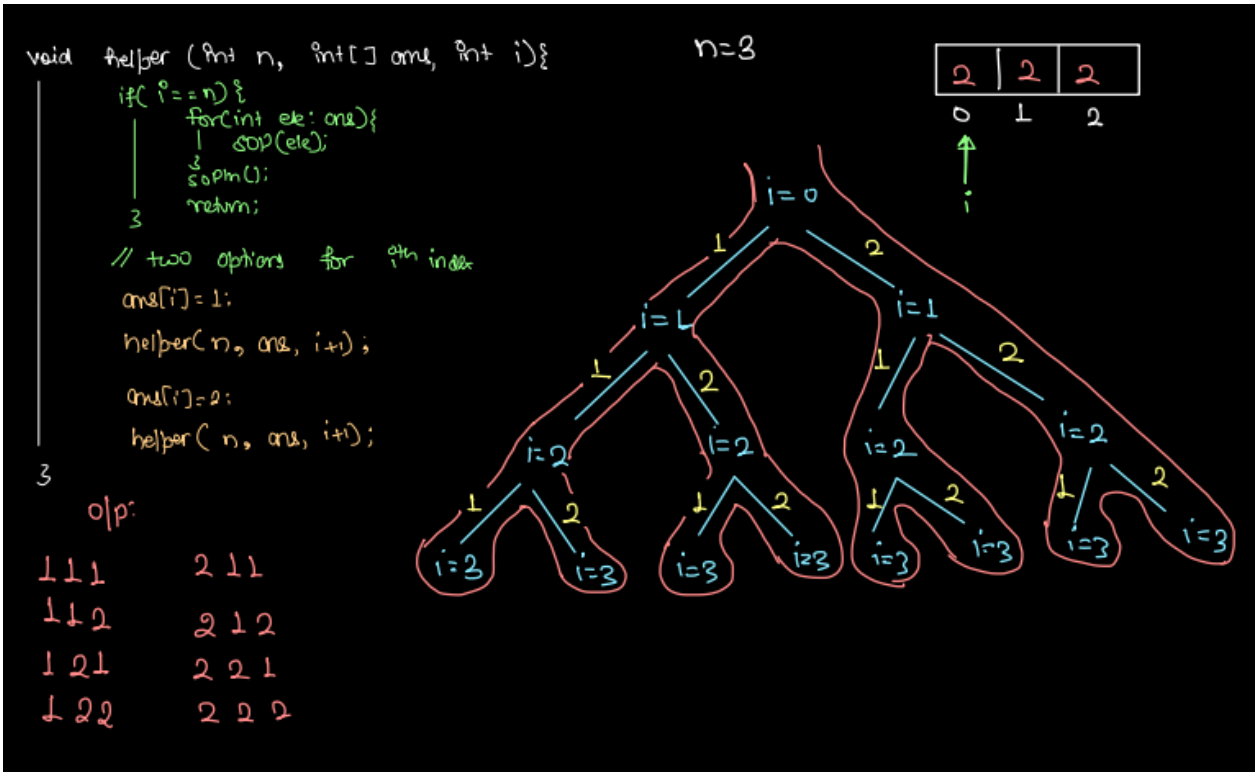
https://www.interviewbit.com/snippet/569823ae5494438cde69/



```
void solve(int n) {
    int[] ans = new int[n];
    helper(n, ans, 0);
}

void helper(int n, int[] ans, int i){
    if(i==n){
        for(int ele: ans){
            SOP(ele);
        }
        SOPln();
        return;
    }
    // two options for ith index
    ans[i] = 1;
    helper(n, ans, i+1);

    ans[i]=2;
    helper(n, ans, i+1);
}
```

TC: $O(2^n * n)$

S.C: $O(n)$ ↓ Recursive Space + answer array.



What is subset -> subset is non-contiguous part of an array in which order of index does not matter. How many sub aray we can create for any array it 2^n it means like array has 4 length the it is 2 X 2 X 2 X 2 = 16 sub array possible.

Subset Sum

Probelm2: Given an array. Find count of subsets with sum = k
arr = [5,7,2]

https://www.interviewbit.com/snippet/ad17567efb9f10cccfe1/

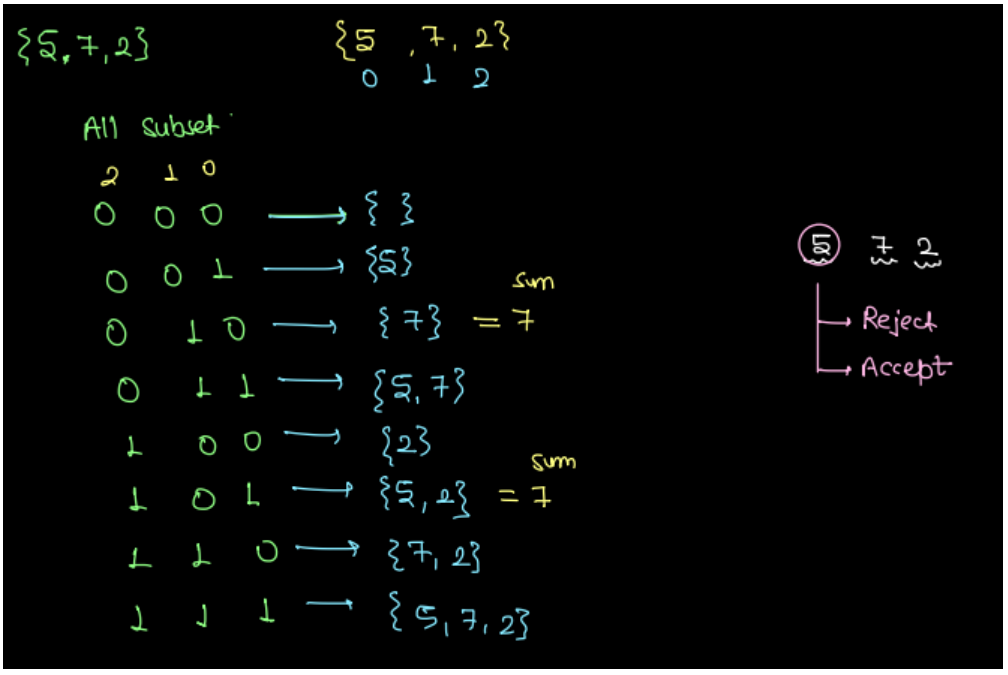
Main Logic ->

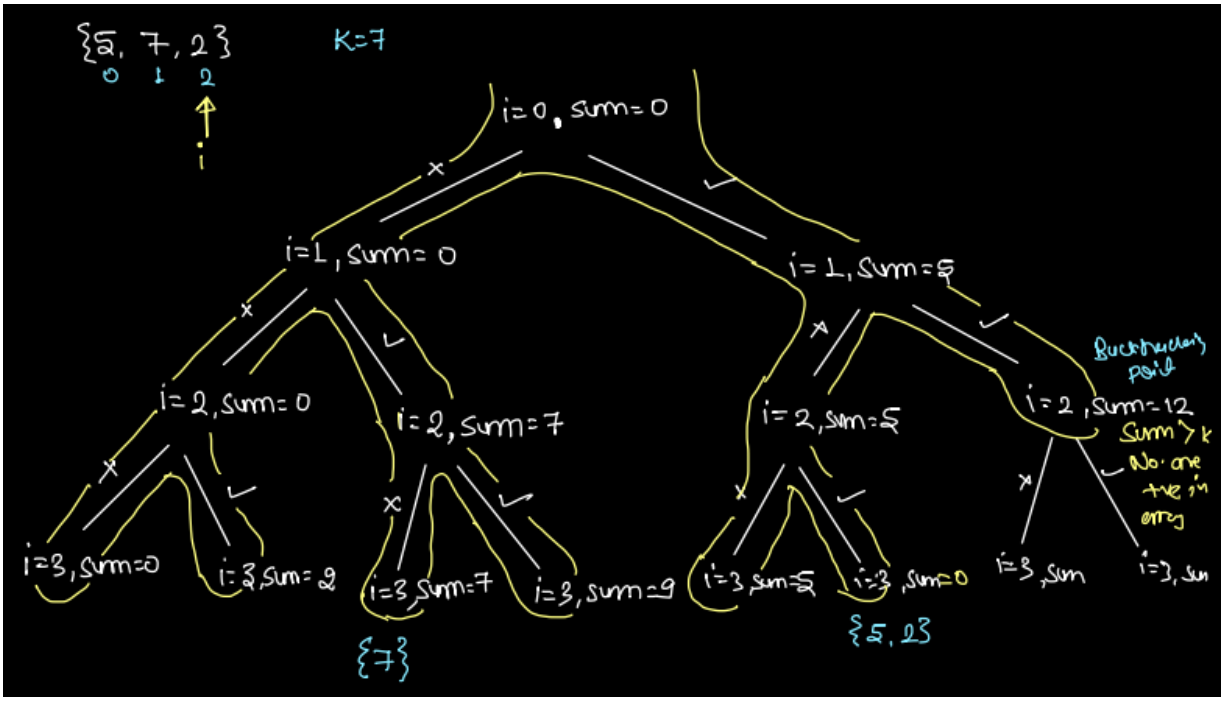
Process Overview: The underlying idea involves using binary choices for generating subsets. The binary value 0 signifies rejection, while 1 stands for selection. Initially, we reject, then in the next step, we add.

Implementation:

1. Begin by establishing a global variable named "count" within the main function and initializing it as 0.
2. Additionally, define another variable named "sum" within the main function, and pass it as a parameter to the recursive function.
3. Create a function named "countSubsetK" that accepts four parameters: an integer array "arr," an integer "index," an integer "k," and an integer "sum."
4. Invoke the "countSubsetK" function in the main function, passing "arr," 0 as the initial "index," "k," and 0 as the initial "sum."
5. Inside the "countSubsetK" function, establish a base case by checking if "index" equals the length of the array "arr." Within this base condition, check if "sum" equals "k," and if so, increment the "count" variable. Then, return.
6. Make the first recursive call to "countSubsetK," passing the parameters "arr," "index + 1," "k," and "sum."
7. Proceed with the second recursive call to "countSubsetK," passing the parameters "arr," "index + 1," "k," and "sum + arr[index]."
8. Finally, in the main function, return the value of the "count" variable.

In summary, this approach uses binary choices to generate subsets, wherein 0 indicates rejection and 1 represents selection. The algorithm counts the subsets that satisfy a specified sum criterion by recursively exploring all possible combinations of elements. The "count" variable tallies the subsets meeting the criteria, and this value is returned from the main function.






```
public static int count;

public static int countOfSubset(int[] arr, int k) {
    count = 0;
    helperforCount(arr, k, 0, 0);
    return count;
}

public static void helperforCount(int[] arr, int k, int i, int sum) {
    // Base case
    if (i == arr.length) {
        if(sum == k){
            count++;
        }
        return;
    }
    //two possibility
    helperForCount(arr, k, i+1, sum);          Reject call
    helperForCount(arr, k, i+1, sum+arr[i]);   Accept call
}
```

8:30 - 8:40
Break

if it is given that arr[i] is +ve ⇒ if(arr[i] + sum <= k){ make ye call. }
make a check before you call

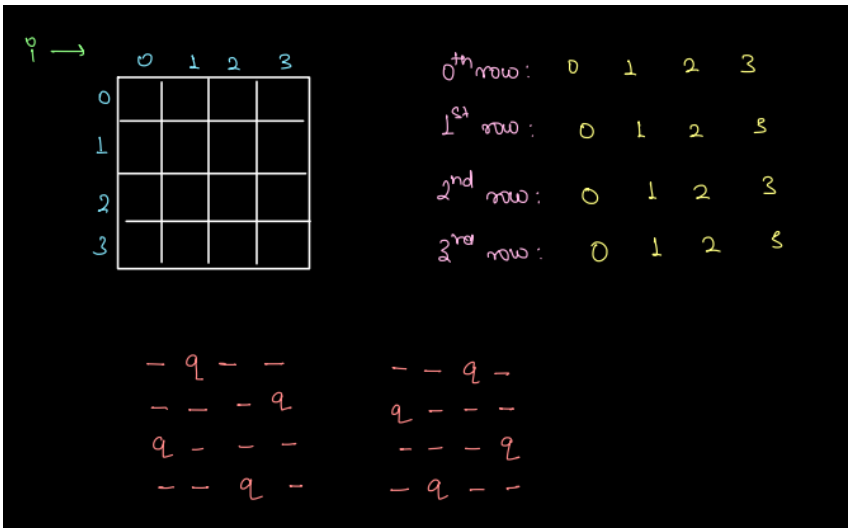
Problem 3:
Given N. Print valid placements of N queens on a NXN board such that no two queens kill each others.
Note: If 2 queens are present in same row/column/diagonal they will kill each other.

Main Logic ->

Steps to Follow ->

1. Start by defining a global 2D array named "board" of type int[][] within the main function. Initialize it using "new int[][]."
2. Create a function called "placeNQueens" and invoke it from the main function, passing 0 as the initial "rowIndex."
3. Inside the "placeNQueens" function, establish a base case: If "rowIndex" is greater than or equal to the length of "board," call the "printBoard" function and return.
4. Implement a conditional check using the "isSafePlace" function. Pass the "rowIndex" and "j" to it. If it returns true, iterate through the range from 0 to < length of "board[0]". Assign 1 to "board[rowIndex][j]."
5. Call the "placeNQueens" function and pass "rowIndex + 1" as the parameter.
6. After the function call, set "board[rowIndex][j]" back to 0. This is essential to unset the queens and explore further possibilities. Close the conditional check and loop.
7. Develop another function named "isSafePlace" with a boolean return type. It should accept two parameters: "rowIndex" and "columnIndex," both of type int.
8. To determine a safe place for new queens, focus on the upper rows since the lower rows are empty and can be discarded. For this, create three loops:
    a. First, check the same column upwards until reaching the top row. Iterate using a "for" loop, initializing two variables, "r" and "c." Set "r = rowIndex" and "c = columnIndex." Implement a condition: "r <= board.length" and decrease "r" while keeping "c" constant. If "board[r][c] = 1," return false.
    b. Second, check the top left diagonal. Use a "for" loop to iterate, creating variables "r" and "c." Initialize "r = rowIndex" and "c = columnIndex." Implement a condition: "r >= 0 && c >= 0." While iterating, decrease both "r" and "c" (r--, c--). If "board[r][c] = 1," return false.
    c. Third, check the top right diagonal. Similar to before, iterate using a "for" loop with variables "r" and "c." Initialize "r = rowIndex" and "c = columnIndex." Implement a condition: "r >= 0 && c < board[0].length." While iterating, decrease "r" and increase "c" (r--, c++). If "board[r][c] = 1," return false.
9. Finally, return true at the end of the "isSafePlace" function.
10. Conclude by implementing the "printBoard" function to display the contents of the 2D array.

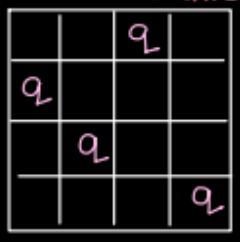
This strategy revolves around placing queens on a chessboard while ensuring their safety in accordance with the "isSafePlace" conditions. The process is governed by recursion, exploring different queen placements and ultimately printing the resulting 2D board configuration.



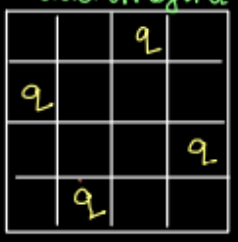
kill each other

N=4

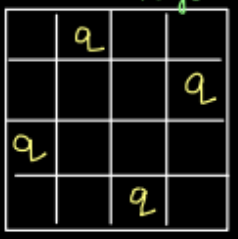
Not a valid arrangement



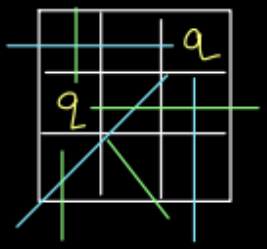
valid arrangement



valid arrangement



N=3

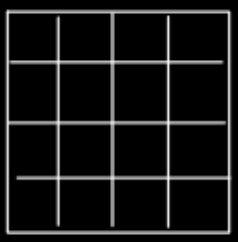


N=3, 3x3 board

talk → place 3 queen

for N <= 3, we can't place n queens in nxn board.

Major talk: Place N queens in N*N board. place queen in such a way that no two queens are killing each other.



Observation:

① Each row will contain exactly one queen.

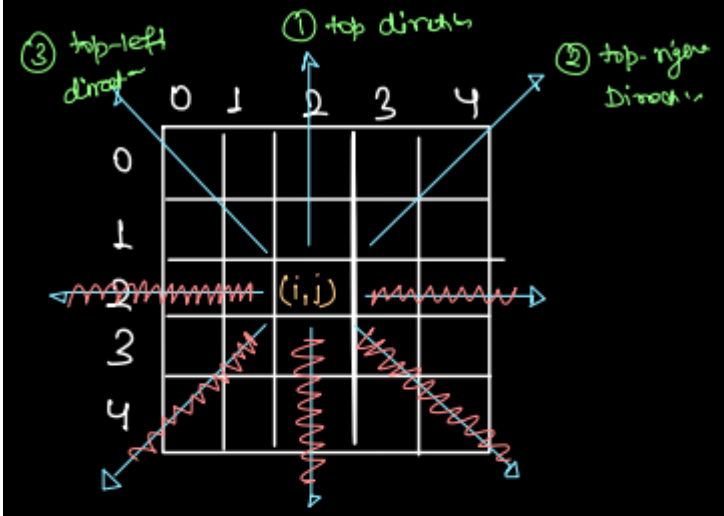
② for queen q_i, how many option option available in single row? → N option.

```
void nqueen(int n){
    int[][] board = new int[n][n];
    helperNQueen(board, 0);
}
void helperNQueen(int[][] board, int rowIndx){
    if(rowIndx == board.length){
        print(board); // print 2D array.
        return;
    }
    // for Every queen, N option available in row
    for(int j=0; j< board.length; j++){
        if(isSafeToPlace(board, rowIndx, j)){
            board[rowIndx][j] = 1; // place queen
            helperNQueen(board, rowIndx+1);
            board[rowIndx][j] = 0; // unplace queen.
        }
    }
}
```

Backtracking.

check function:

isSafeToPlace(int[][] board, int i, int j)




```
// top check
for(int r=i-1, c=j; r>=0; r--){
    if( board[r][c]==1){
        return false;
    }
}
// top-left check → diagonal
for(int r=i-1, c=j-1; r>=0 && c>=0; r--, c--){
    if( board[r][c]==1){
        return false;
    }
}
// top-right check → diagonal
for(int r=i-1, c=j+1; r>=0 && c<board.length; r--, c++){
    if( board[r][c]==1){
        return false;
    }
}
return true;
```

DSA: DP-1 - 19 - Aug 2023

- Introduction of DP
- Repeating Problems in fib.
- N stairs
- Minimum Number of Squares

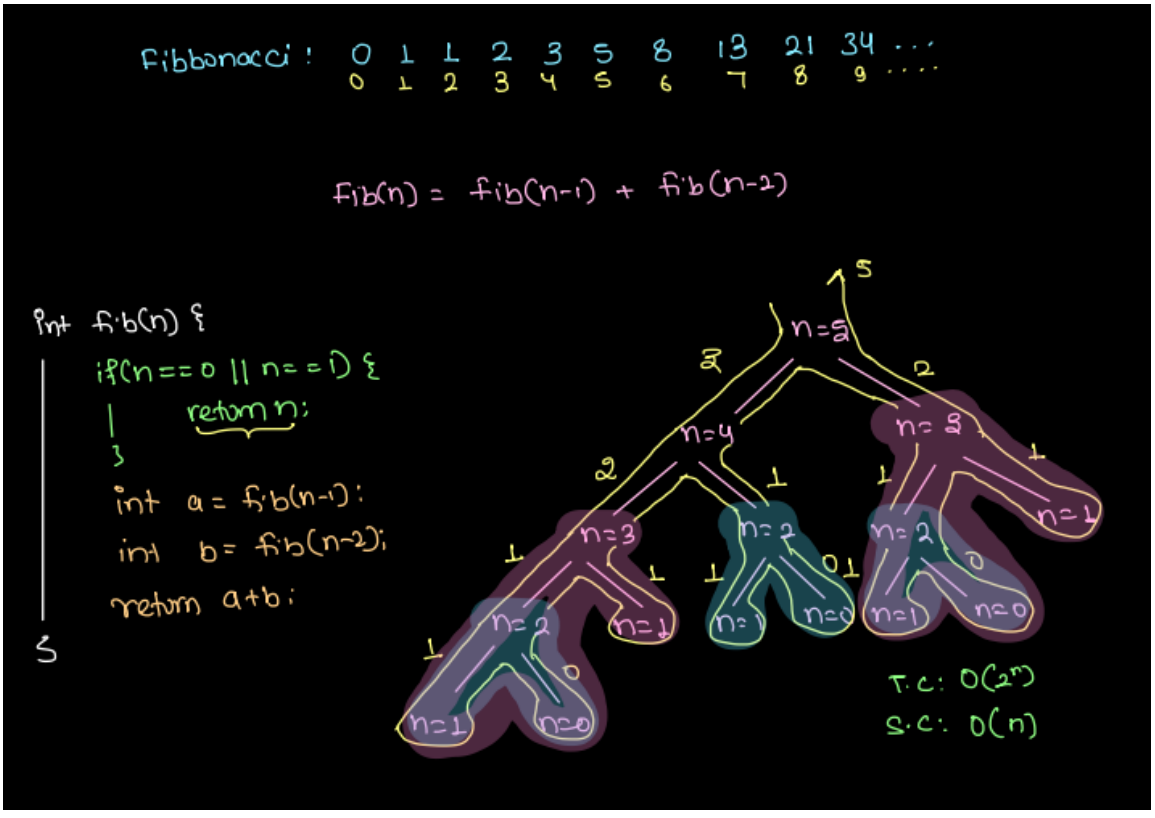
Problem 1: Introduction to DP (Using Fibbonacci)
Given the value N, find Nth Fibbonacci number:
https://www.interviewbit.com/snippet/cda908b27178d5afced5/

Main Logic ->

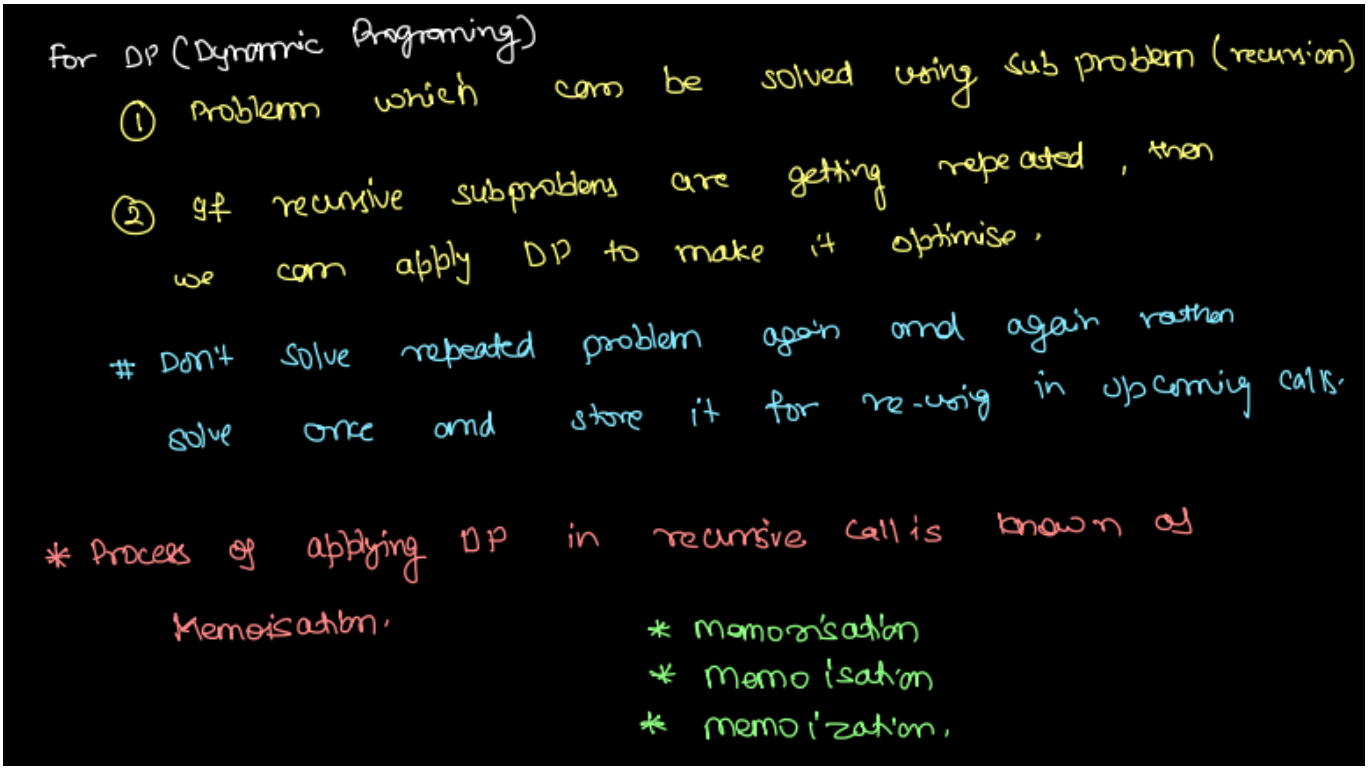
What is the Fibonacci series? The Fibonacci series is generated by the iterative addition of the preceding two numbers. In essence, this sequence relies on the initial two numbers for its construction, often commencing with 0 and 1, though alternative starting values are possible.
To compute the Fibonacci series, a recursive approach is commonly employed, following these steps:

1. Define a function named "fib" and int return type.
2. Establish a base case: If the input "n" equals 0 or 1, return "n" (these are the first two values in the series).
3. Introduce two integer variables, "num1" and "num2". Assign "num1" with the result of the "fib" function, supplied with the argument "n-1". Assign "num2" with the result of the "fib" function, using "n-2" as the argument.
4. Combine "num1" and "num2" through addition, and subsequently, return the sum.
   In the main part of the code, invoke the "fib" function and provide the desired input number.

This recursive process generates the Fibonacci series by successively adding the two preceding terms, with the initial two values acting as the foundation of the sequence.



Introduction of DP



Main Logic
Implement DP (Dynamic programming)

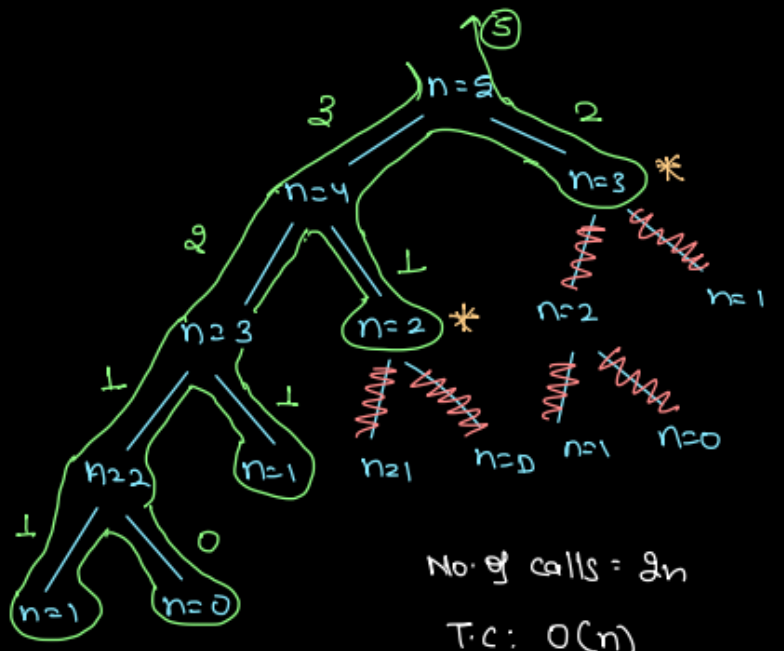
To efficiently compute the Fibonacci series using Dynamic Programming (DP), the results are stored in an array to be reused whenever needed, thus avoiding redundant calculations. The process follows these guidelines:

1. Begin by defining a global variable named "dp" of type int[], or within the main function if preferred. If defined within the main function, pass it as an argument to the recursive function. If global, it can be used directly.
2. Initialize the "dp" array in the main function with a new int array of size "n". Fill the "dp" array with -1 using the method Array.fill(dp, -1).
3. Create a function named "fib" with an integer return type.
4. Establish a base case: If the input "n" is equal to 0 or 1, return dp[n] = n. These values represent the first two elements of the series.
5. Add a condition to check if the value for index "n" already exists in the "dp" array. If dp[n] is not equal to -1, return dp[n].
6. Introduce two integer variables, "num1" and "num2". Assign "num1" the result of the "fib" function with the argument "n-1". Assign "num2" the result of the "fib" function using "n-2" as the argument.
7. Combine "num1" and "num2" through addition, and then return dp[n] = sum.
8. In the main portion of the code, call the "fib" function with the desired input number.

This approach employs a recursive procedure that generates the Fibonacci series while making optimal use of previously computed values. The initial two values in the series serve as the foundational elements for this iterative process.

```
int[] strg = new int[n+1];
Arrays.fill(strg, -1);

int fib(n) {
    if(n==0 || n==1) {
        strg[n]=n;
        return n;
    }
    if(strg[n] != -1) {
        return strg[n];
    }
    int a = fib(n-1);
    int b = fib(n-2);
    strg[n] = a+b;
    return a+b;
}
```

0	1	1	2	3	5
0	1	2	3	4	5



No. of calls = $2n$

T.C: $O(n)$

S.C: $O(n)$

Comparison of T.C →

Without DP:

T.C: $O(2^n)$

$n=100$, Itr = 2^{100}

$2^{100} \equiv 10^{27}$

Itr: 10^{27}

With DP (memoisation)

T.C: $O(n)$

$n=100$

Itr = 100

Massive saving of Itr with help of DP.


```java
import java.util.*;

class Main {

    static int[] strg;

    public static int fib_rec(int n) {
        if(n == 0 || n == 1) {
            strg[n] = n;
            return n;
        }

        // if answer is available, no need to calculate it again
        // return from storage
        if(strg[n] != -1) {
            return strg[n];
        }

        int a = fib_rec(n - 1);
        int b = fib_rec(n - 2);

        // if answer is calculated now, store the answer in storage
        strg[n] = a + b;
        return strg[n];
    }

    public static int fib(int n) {
        // initialise global storage with n+1 size
        strg = new int[n + 1];
        // fill -1 in storage
        Arrays.fill(strg, -1);
        // make recusive call
        return fib_rec(n);
    }

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

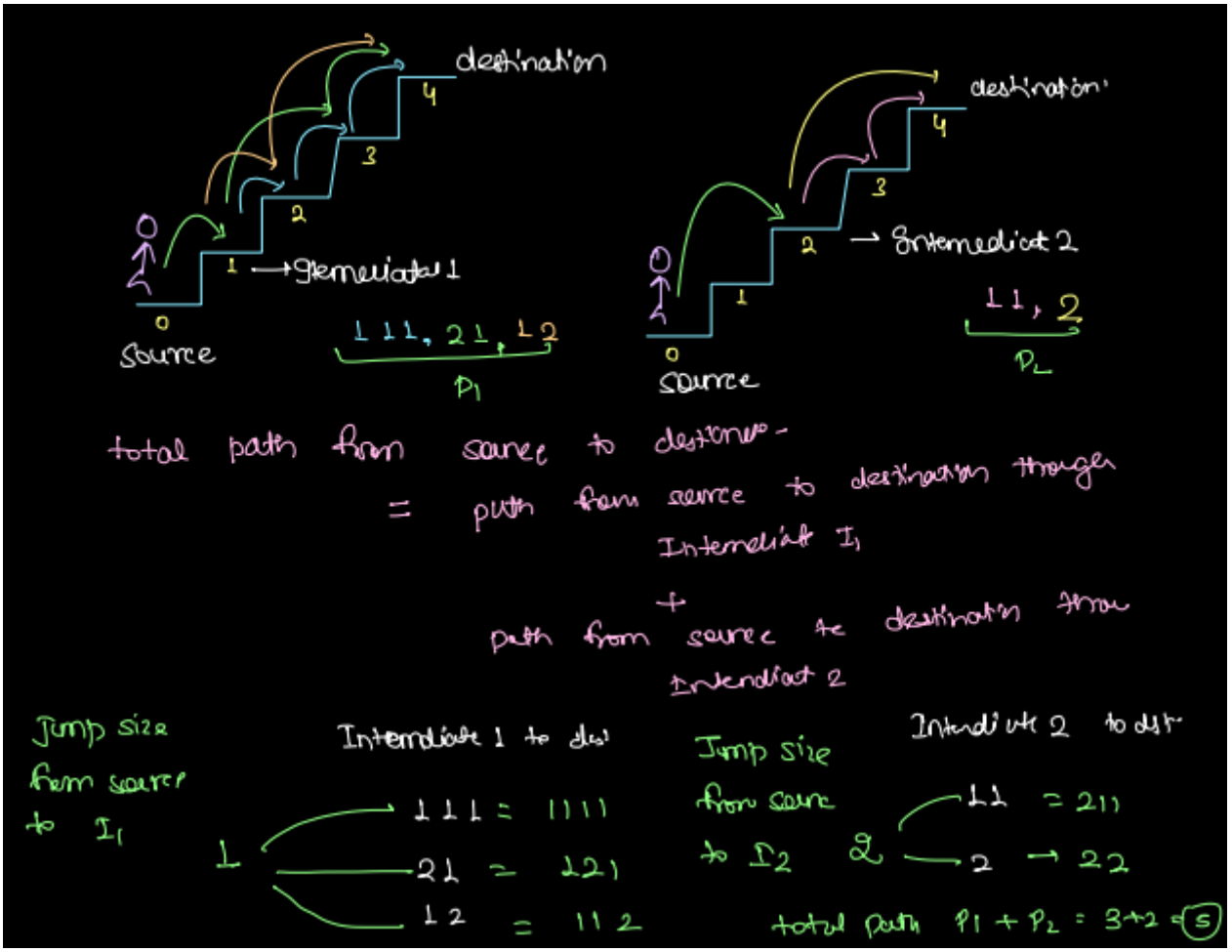
Problem 2 - N Stairs
Given N, find total number of ways to go from 0th stair to Nth stair.

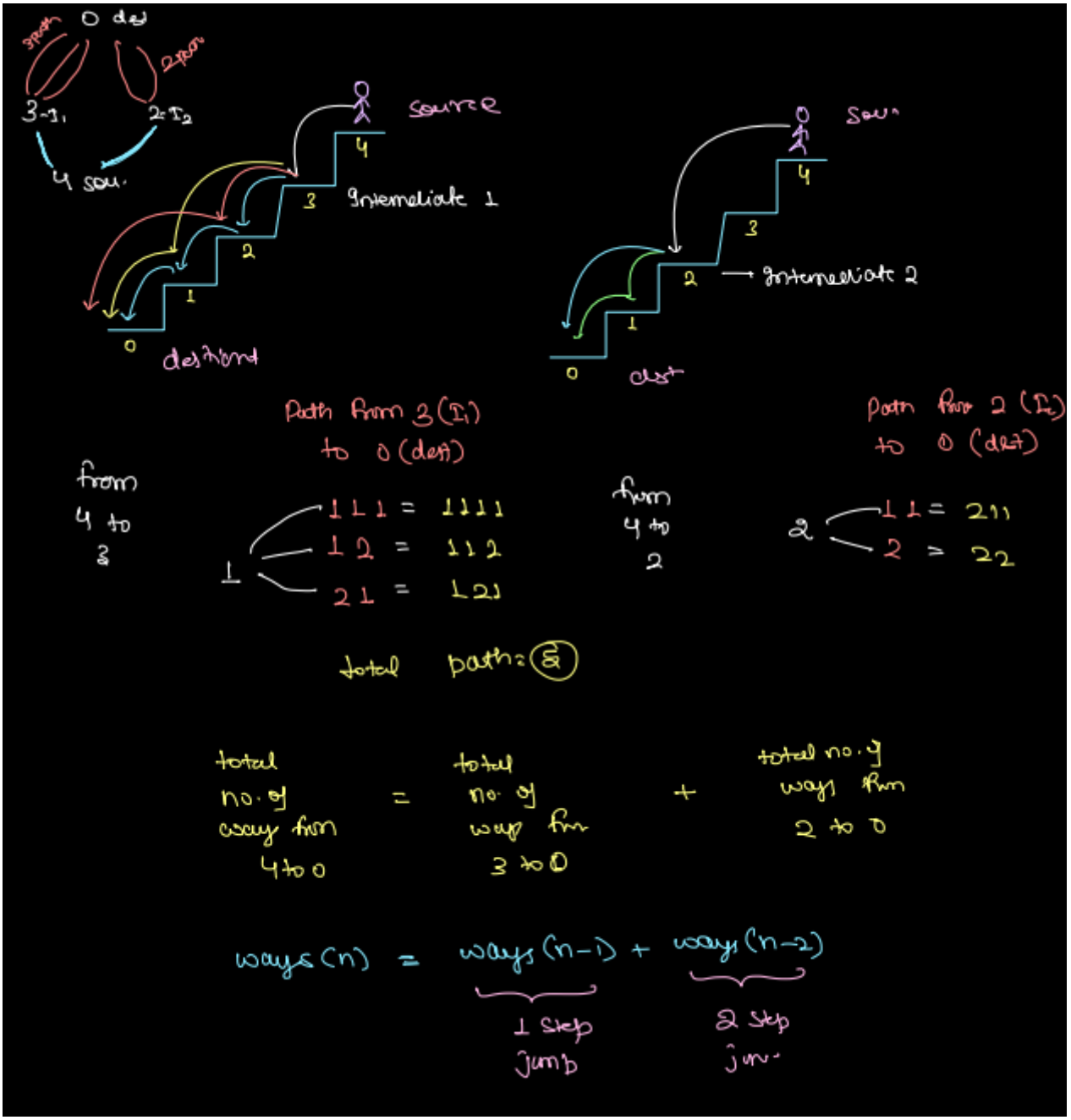
Note: you can take step of length 1 and 2.
https://www.interviewbit.com/snippet/15e6316ce24df0de0f86/

Main logic

When figuring out how to reach a destination, we can use two approaches: starting from the first step and going to the top, or starting from the top and going to the first step. Surprisingly, both methods give the same result. We'll use the top-to-bottom approach here, similar to how we handle the Fibonacci series. Just remember that instead of using 0 and 1 as the starting points, we'll use the given steps to reach the destination.

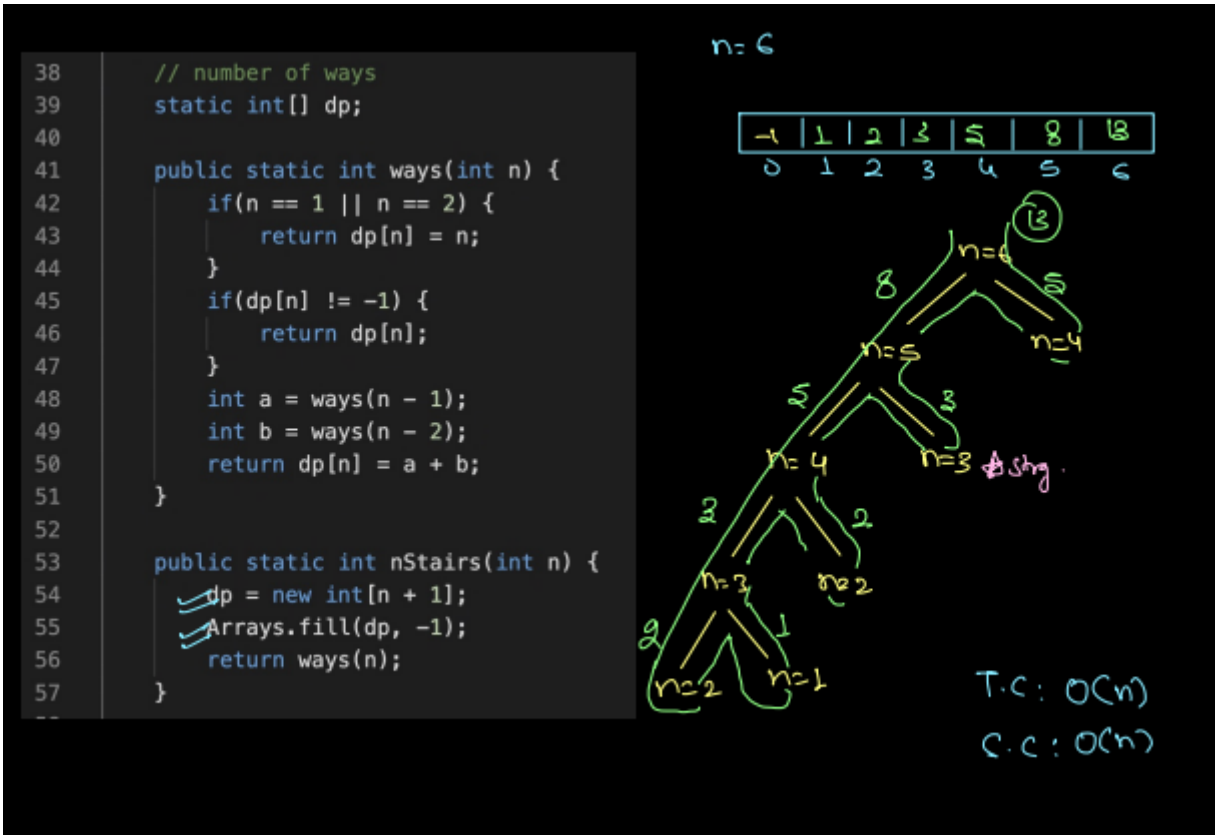
1. Begin by defining a global variable named "dp" of type int[], or within the main function if preferred. If defined within the main function, pass it as an argument to the recursive function. If global, it can be used directly.
2. Initialize the "dp" array in the main function with a new int array of size "n". Fill the "dp" array with -1 using the method Array.fill(dp, -1).
3. Create a function named "fib" with an integer return type.
4. Establish a base case: If the input "n" is equal to 1 or 2 or given jump numbers, return dp[n] = n. These values represent the first two elements of the series.
5. Add a condition to check if the value for index "n" already exists in the "dp" array. If dp[n] is not equal to -1, return dp[n].
6. Introduce two integer variables, "num1" and "num2". Assign "num1" the result of the "fib" function with the argument "n-1". Assign "num2" the result of the "fib" function using "n-2" as the argument.
7. Combine "num1" and "num2" through addition, and then return dp[n] = sum.
8. In the main portion of the code, call the "fib" function with the desired input number.





Reminder $fib(n) = fib(n-1) + fib(n-2)$

```
int ways(int n){
    if(n==1 || n==2){
        return n;
    }
    int a= ways(n-1);
    int b= ways(n-2);
    return a+b;
}
```



Probelm 3: Minimum number of squares
Find minimum count of numbers, sum of whose squares is N. N > 0;
https://www.interviewbit.com/snippet/492e4ad3559f800ffa91/

Main logic ->

The idea here is to calculate the squares of numbers from 1 up to a number called "num," and then subtract these squares from "num." This result is passed to a recursive function. This function counts the remaining steps by repeatedly subtracting the squares of numbers and then returns the minimum count of steps.

Here are the steps to achieve this:

1. Create a global array called "dp" to store intermediate results.
2. In the main function, create an array "dp" of size n+1 and fill it with -1 using "Arrays.fill(dp, -1)."
3. Define a function called "minSqr."
4. Establish a base case: If the input "n" is 0 or 1, return dp[n] = n.
5. Create an integer variable "min" and set it to a high value like Integer.MAX_VALUE.
6. Check if the value for index "n" already exists in the "dp" array. If dp[n] is not -1, return dp[n].
7. Run a loop up to "n," using "i" as the loop variable. Inside the loop, calculate "count" by recursively calling the function with the parameter "n - i*i." Store the recursive result in the "count" variable.
8. Compare "min" and "count" using "Math.min" and assign the smaller value to "min."
9. Return dp[n] = min + 1. (Here, 1 represents the square value, and "count" represents the remaining steps.)

In a nutshell, this process involves calculating squared numbers, subtracting them from the given value, and then recursively finding the minimum steps required to reach zero using these squared values. The intermediate results are stored in the "dp" array to avoid redundant calculations.

N=6

$1^2+1^2+1^2+1^2+1^2+1^2 = 6$ (6)

$1^2+1^2+2^2 = 6$ (3)

→ min = (3) Ans

N=10

$1^2+1^2+1^2+1^2+\ldots\ldots+1^2 = 10$ [10]

$1^2+1^2+\ldots+1^2+2^2 = 10$ [7]

$1^2+1^2+2^2+2^2 = 10$ [4]

$1^2+3^2 = 10$ [2]

min = (2)

N=9 ⇒ $3^2 = 9$ [1]] min = (1)

N=12

$1^2+1^2+1^2+\ldots\ldots+1^2 = 12$ [12]

↳ $9+1+1+1$

$= 3^2+1^2+1^2+1^2$

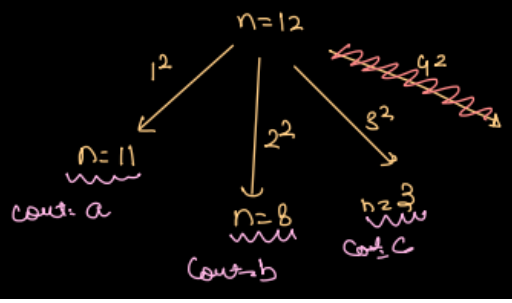
$1^2+1^2+1^2+3^2 = 12$ [4]

$2^2+2^2+2^2 = 12$ [3]

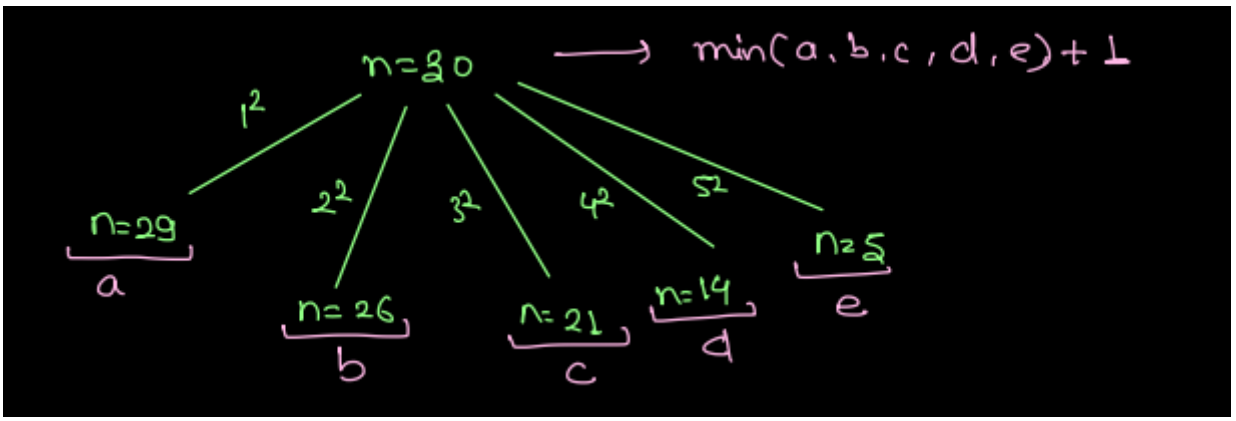
why logic

we can't apply logic of nearest perfect square why?

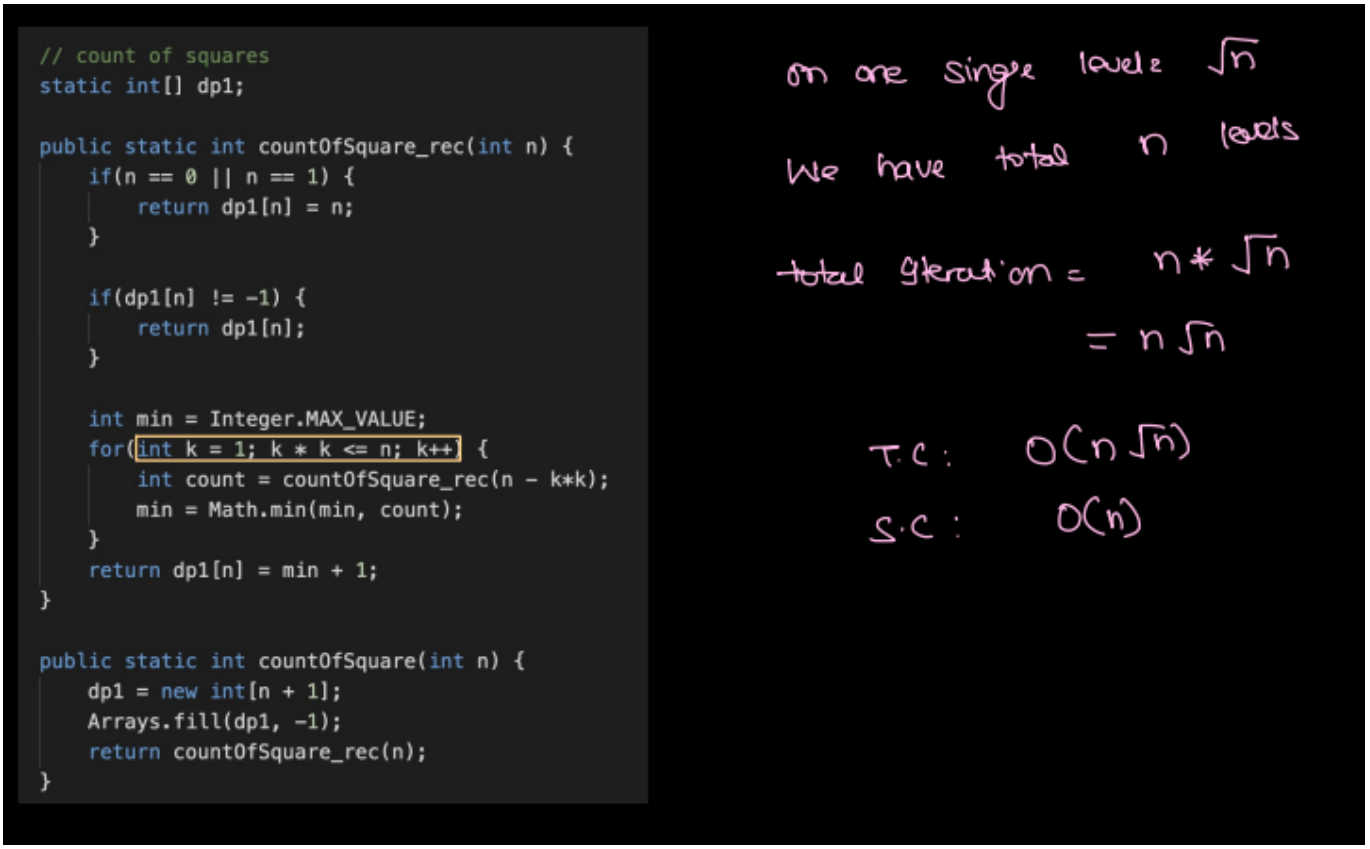
→ greedy will not work



min(a, b, c) + 1



→ min(a, b, c, d, e) + 1



DSA: DP-2 - 22 - Aug 2023

- Max Subseq Sum without adjacent element
- Count of Path In Grid
- Count of Path in Grid with blocker
- Min Cost Path

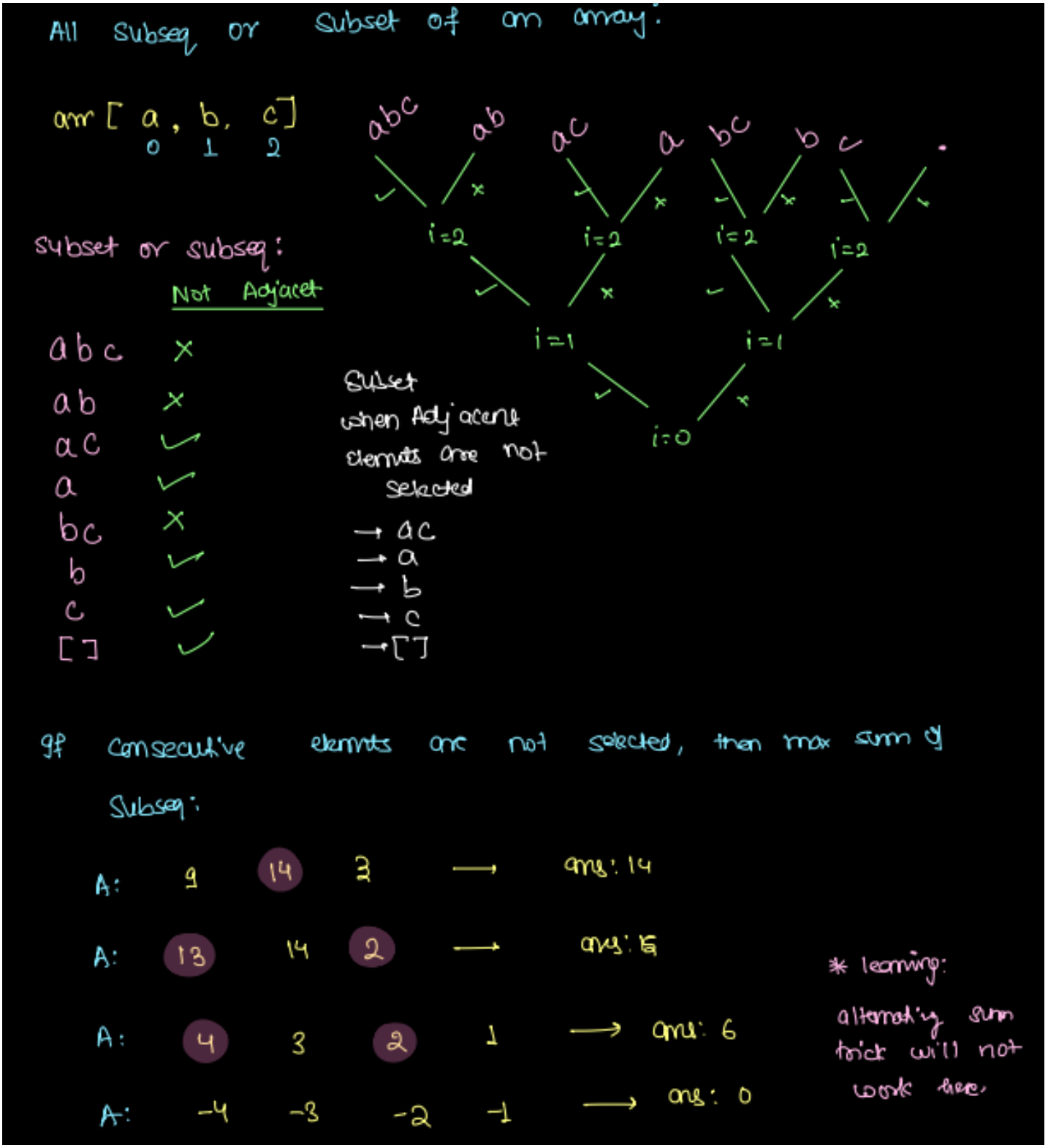
Problem1: Maximum Subsequence sum without adjacent elements
Given an array A[], Find maximum subsequence sum such that no two consecutive elements are picked.
https://www.interviewbit.com/snippet/eac302f78f5888999d64/

Main logic ->

We will approach this problem by traversing the array in reverse for the sake of simplicity in handling base conditions. The basic logic is to avoid selecting adjacent elements. Elements are considered adjacent if they are side by side (e.g., "ab" is adjacent, but "ac" is not). If we select the first element, we'll skip the second element and directly select the third. Conversely, if we skip the first element, we'll directly select the second.
Here are the steps to achieve this:

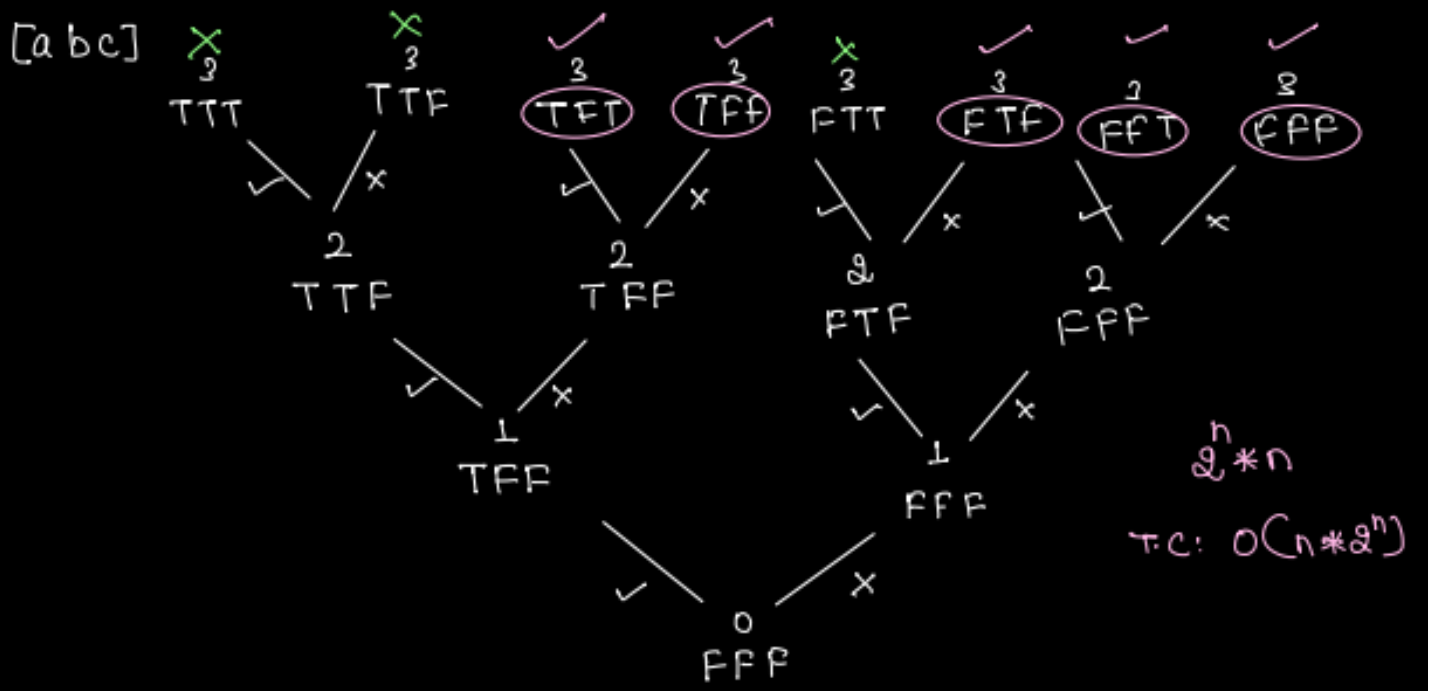
1. Define a function maximumSubsequenceSum with a return type of int and two parameters: an integer array arr and an integer index.
2. Implement a base case to handle the recursion. Since we're traversing from the back and reducing the index by 1 and 2 in each recursive call, we'll have a condition that if index < 0, we'll return 0.

3. Create two integer variables a and b. Assign the result of the recursive function call to a, passing arr and index - 2. Similarly, assign the result of the recursive function call to b, passing arr and index - 1.
4. Since the first call involves selecting the first element and the third element in the next call, we'll consider it a selected call. We'll calculate a + arr[i] and use the max function to choose the greater value between a + arr[i] and b. Return this maximum value.
5. In the main function, call maximumSubsequenceSum, passing arr and arr.length - 1, and return its result.

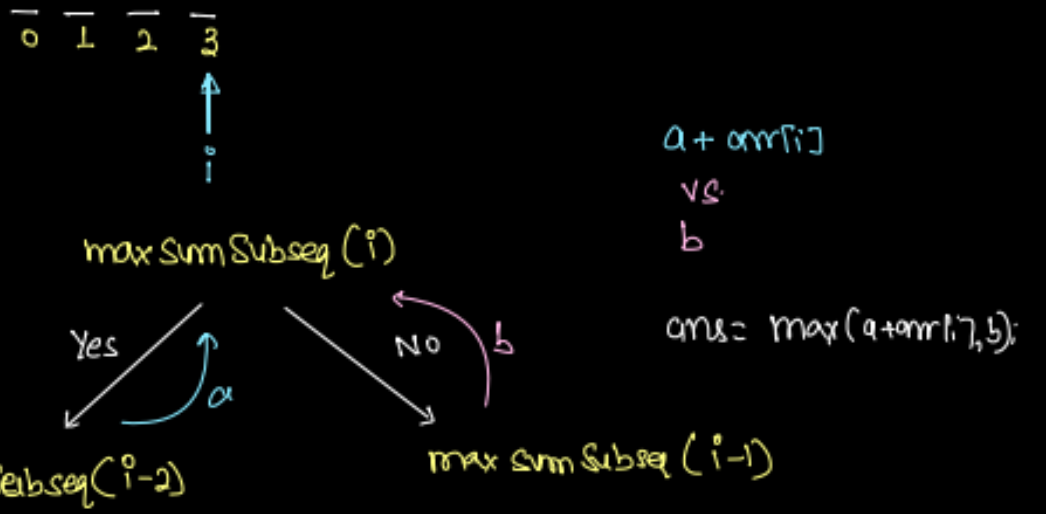


bruteforce Approach:

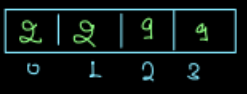
→ create all the subseq. on reaching at base case check if the seq. have adjacent elemt are selected or not, if adjacent or consecutive elemt are not selected maximise sum.

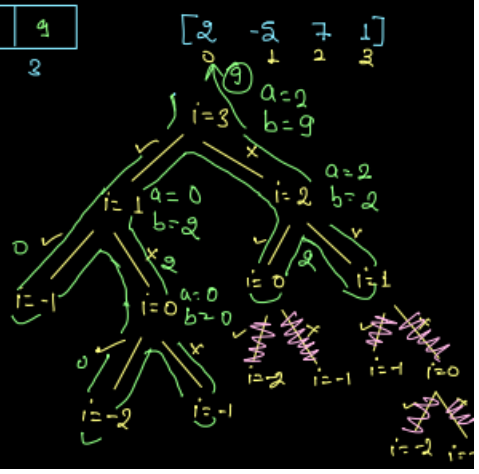


optimisation :



```
int solve (int[] arr){
    int n= arr.length;
    dp = new int[n];
    Arrays.fill ( dp, -1);
    return helper (A, n-1);
}

int[] dp;

int helper (int[] arr, int i){
    if( i < 0){
    |   return 0;
    }
    if(dp[i] != -1) {
    |   return dp[i];
    }
    int a = helper(A, i-2); // Selecting ith index
    int b = helper (A, i-1); // Rejecting ith index

    int ans= Math.max (a+arr[i], b);

    return dp[i]=ans;
}
```

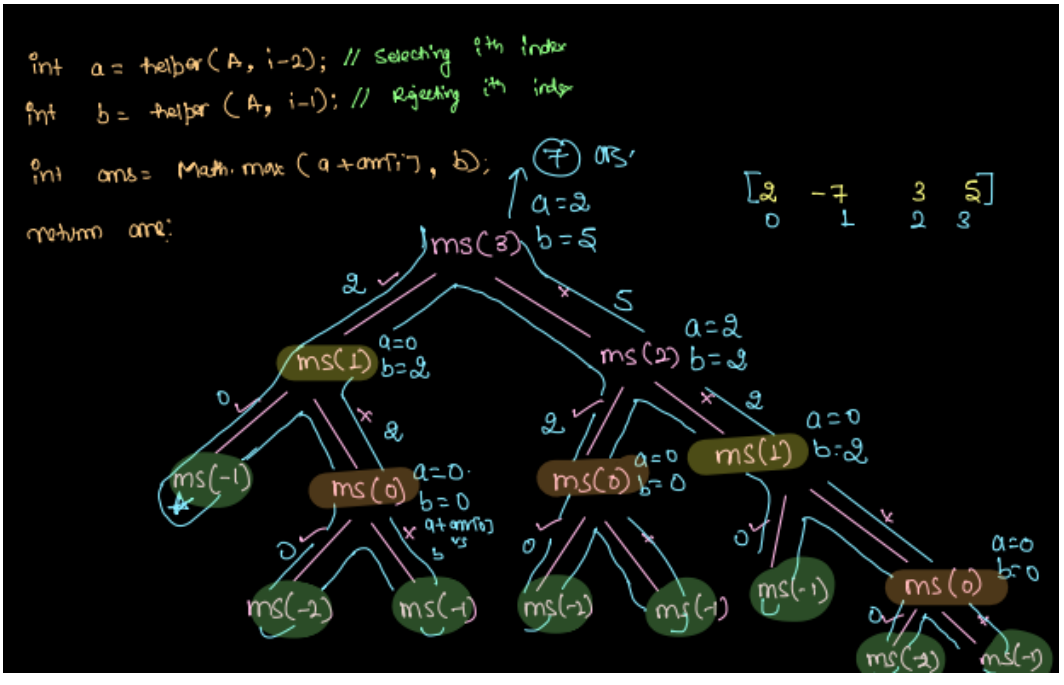




Tc: O(n)

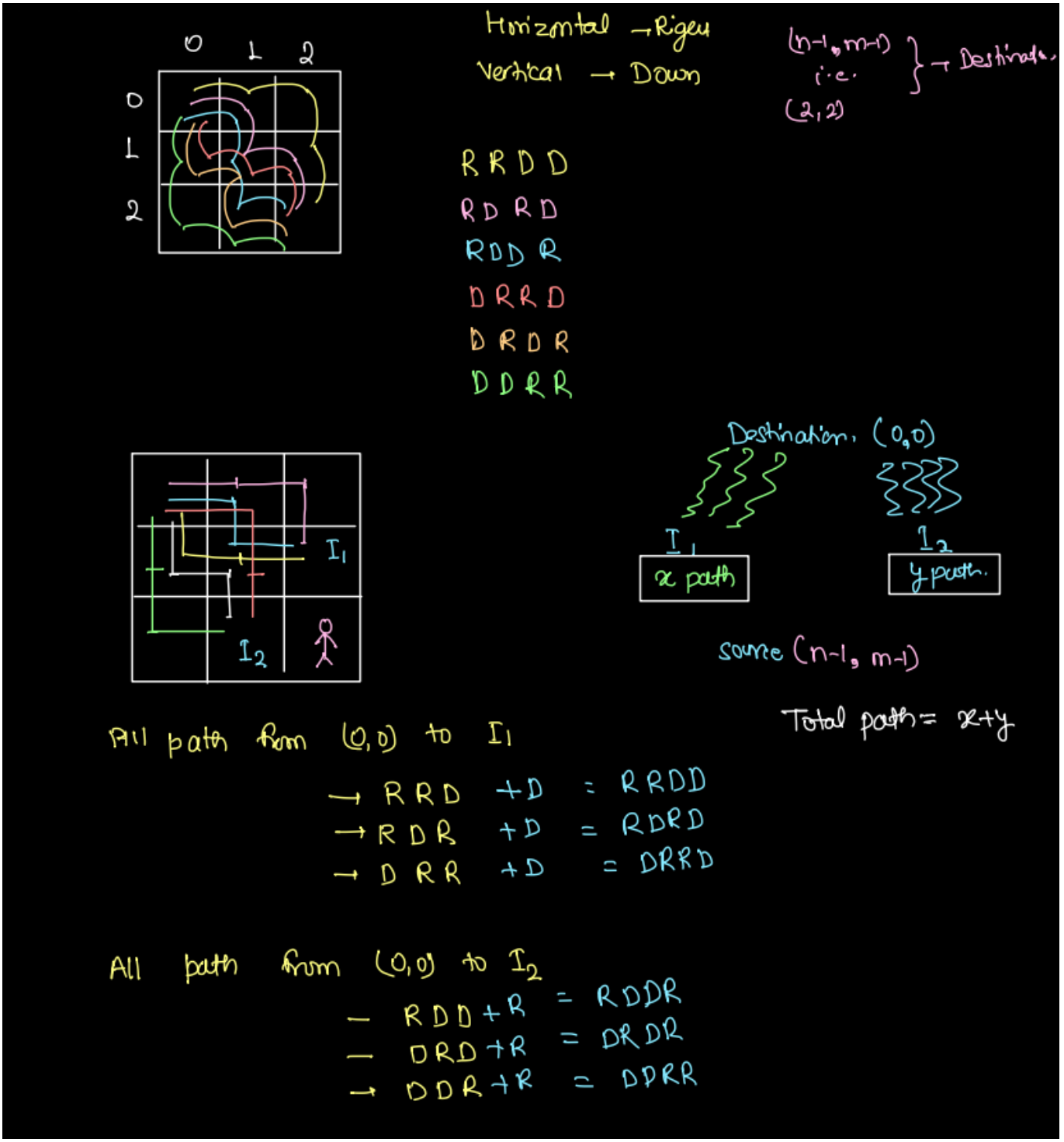
S.C. O(n)

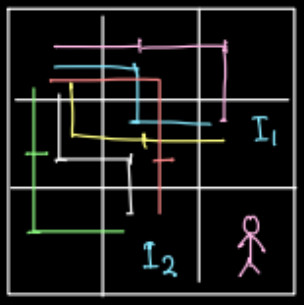
```
int a = helper(A, i-2); // Selecting ith index
int b = helper (A, i-1); // Rejecting ith index

int ans= Math.max (a+arr[i], b);

return ans;
```



Problem2:Count of paths in grid
Count total number of ways to go from (0,0) to (n-1, m-1).

Allowed movement is:
i. Right movement by one step
ii. Down movement by one step



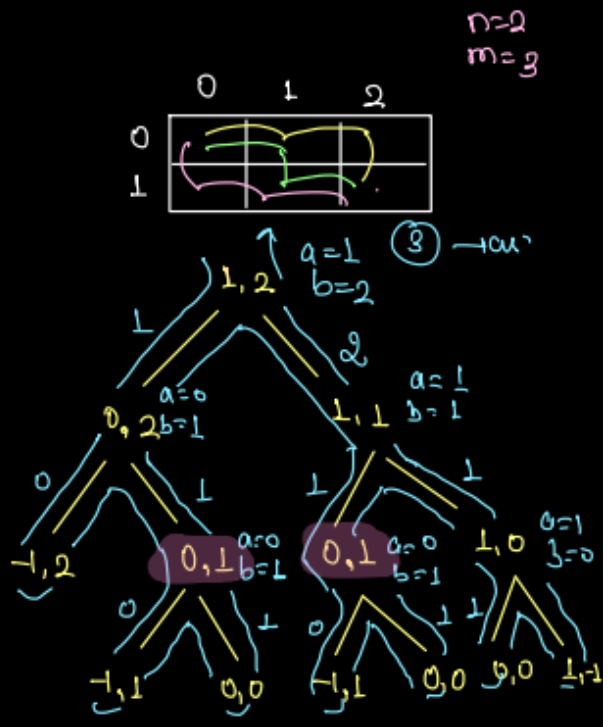


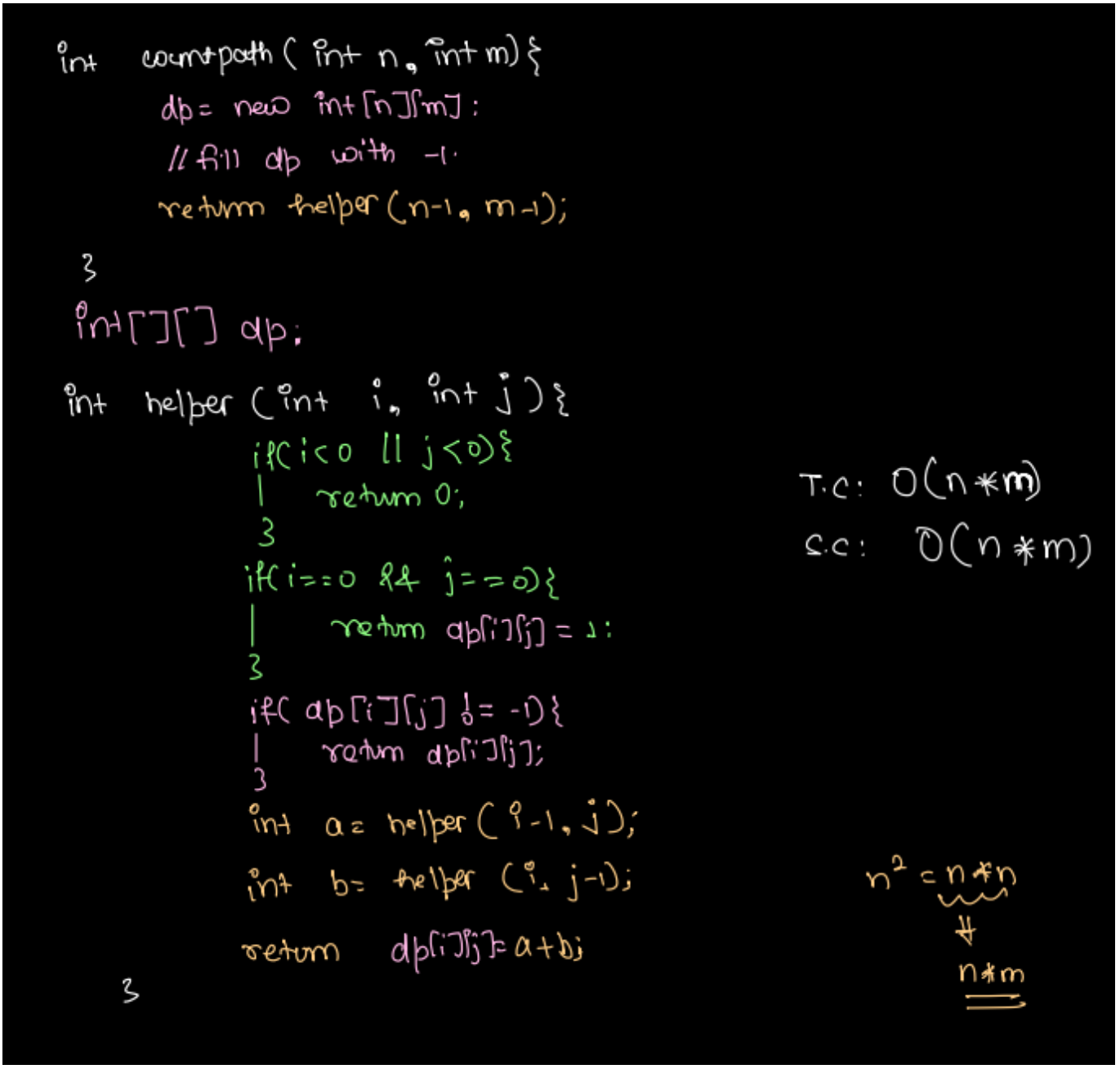
ways(i,j) = ways(i-1, j) + ways(i, j-1);

path from (i,j) to (0,0) — path from (i-1, j) to (0,0) — path from (i, j-1) to (0,0)

```
int countpath(int n, int m){
    return helper(n-1, m-1);
}

int helper(int i, int j){
    if(i<0 || j<0){
        return 0;
    }
    if(i==0 && j==0){
        return 1;
    }
    int a = helper(i-1, j);
    int b = helper(i, j-1);
    return a+b;
}
```

RRD
RDR
DRR



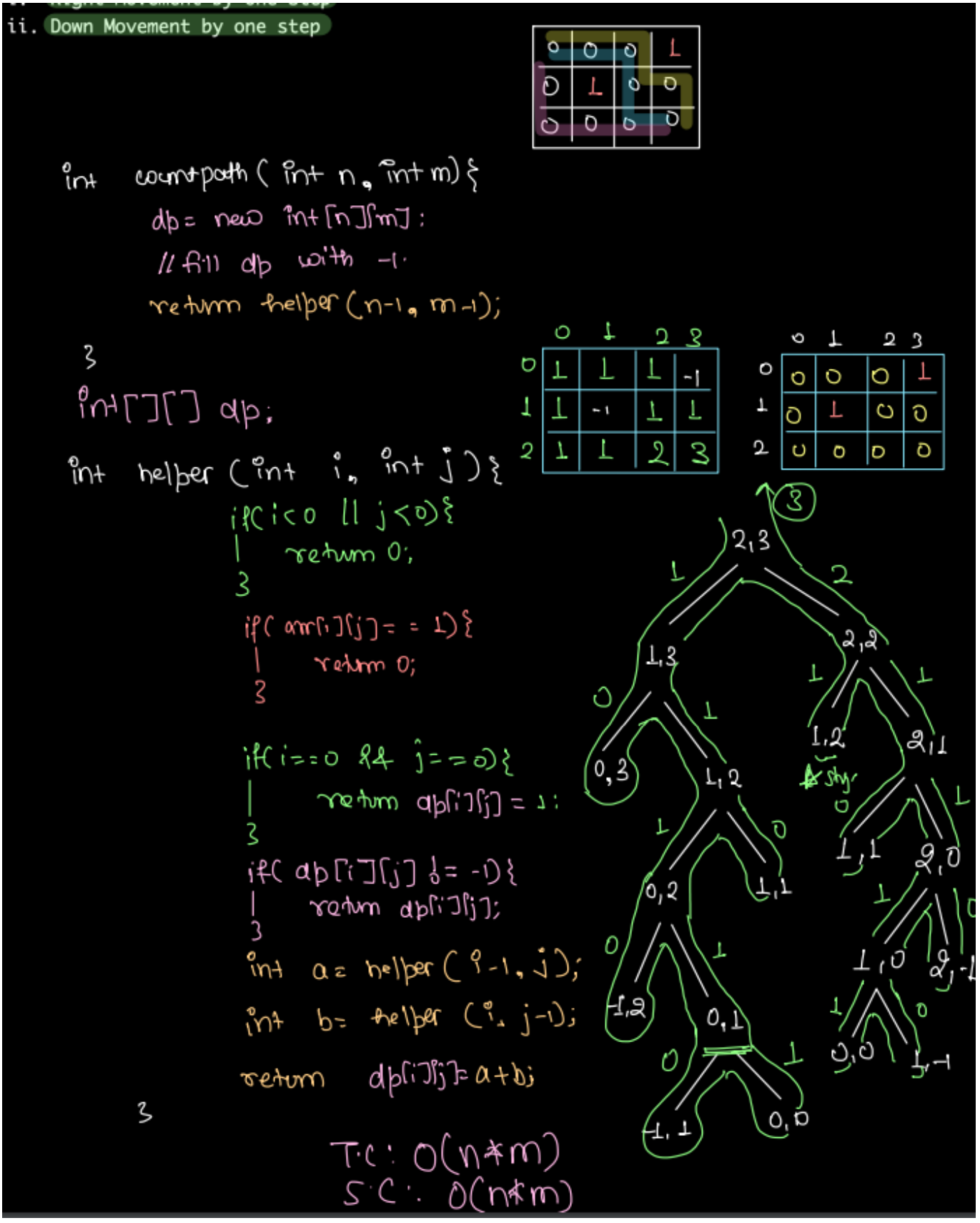


Probelm3: Count of paths in grid with blocked cells
Count total number of ways to go from (0,0) to (n-1, m-1)

Note: if grid[i][j] = 1, that means(i,j) cell is blocked. otherwise unblocked

Allowed movement is:

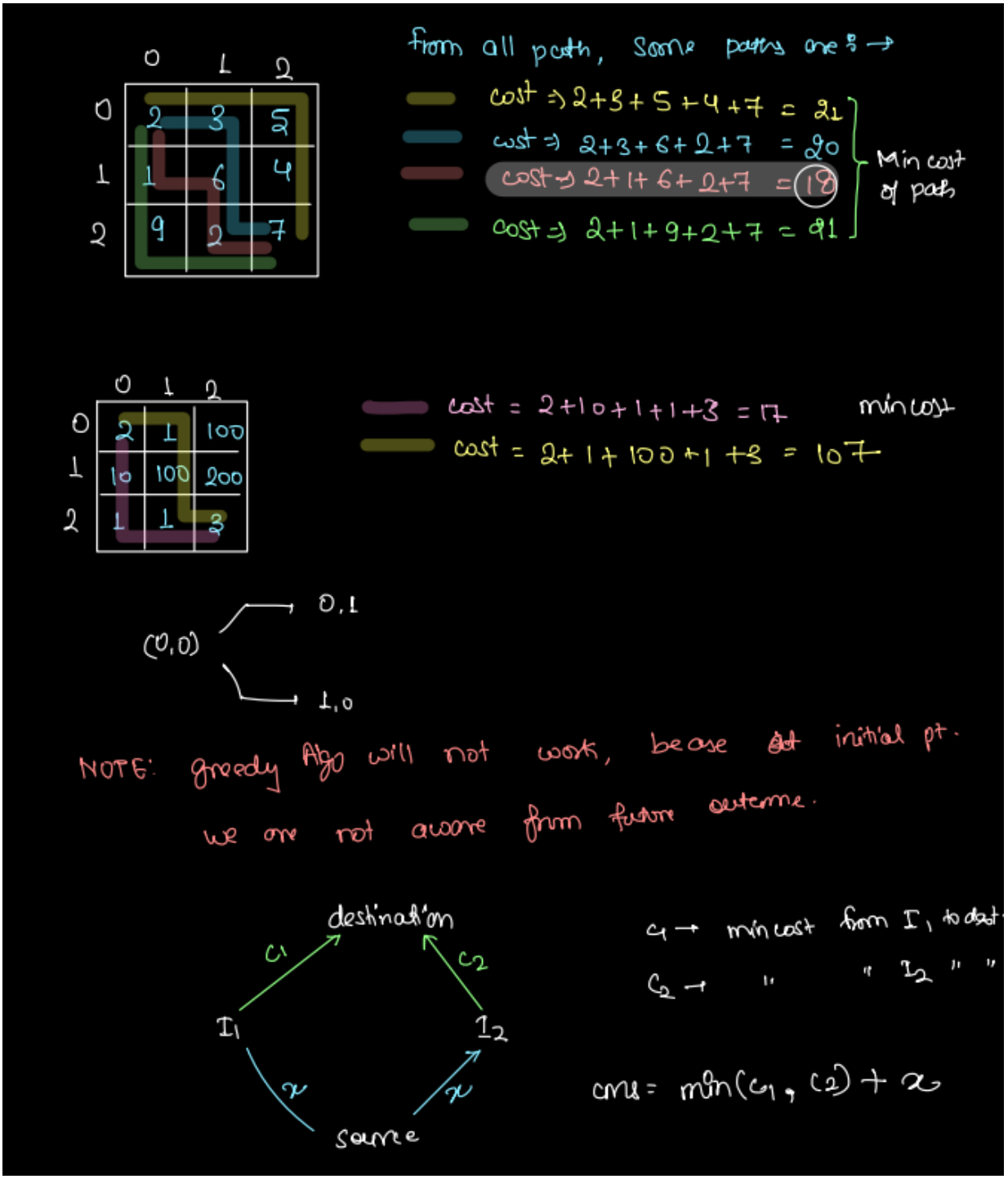
i. Right Movement by one step
ii. Down movement by one step.

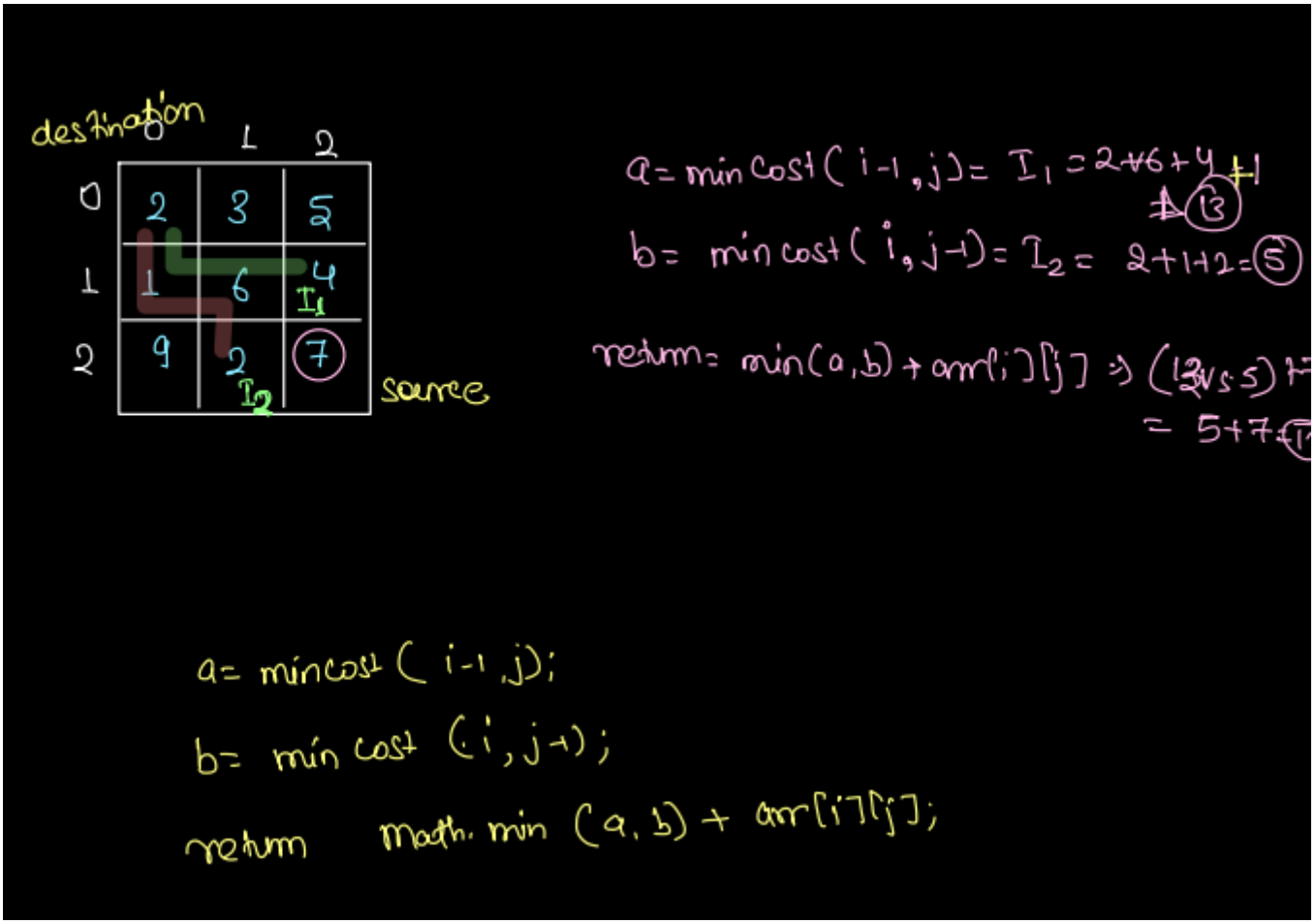


Problem4: Minimum sum path
Given a 2D matrix mat[][], filled with non-negative numbers, find the minimum cost path from (0,0) to (n-,m-1)

Allowed movement is:
i. Right movement by one step.

ii. Down movement by one step





DSA: Contest 7 discussion - 24 - Aug 2023

- **Greater Sum Tree**
- **Winner Stone**
- **Print Maze Path**

Problem1 : Greater Sum tree

Given the root of a binary search tree of size of size N with distinct values, transform it into a "greater sum tree" such that each node in the new tree has a value equal to the original tree node value plus the sum of all values greater than the original node value in the tree.
return the new binary search tree.
https://www.interviewbit.com/snippet/fcfbff66f2b47a364911/

Main logic ->

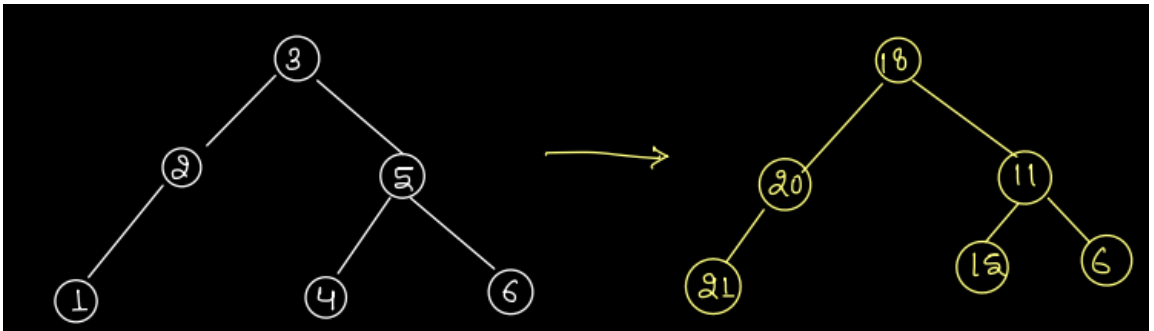
To find the greater sum of a binary tree, we'll traverse the tree using an inorder approach, but this time starting from the right instead of the left. In an inorder traversal, we obtain a sorted array in ascending order.

Here are the steps to achieve this:

1. Declare a static variable called 'sum' and create a function named 'greaterSum' with a void return type. This function will accept a 'node' parameter.

2. In the main function, initialize the 'sum' variable to 0. Then, call the 'greaterSum' function, passing the root node as an argument.
3. Within the 'greaterSum' function, establish a base case: if the 'node' is null, return.
4. Recursively call 'greaterSum' on 'node.right'. We traverse to the right child since we're aiming to traverse the tree in reverse.
5. Calculate the new sum as follows: sum = sum + node.val. After that, update the 'node.val' with the new sum value.
6. Finally, return the 'node' from the 'greaterSum' function.

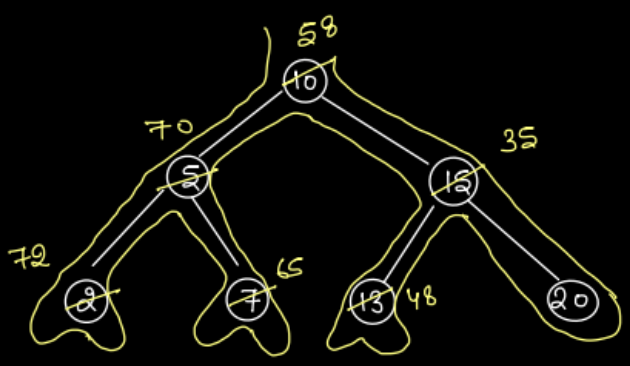
This approach will effectively modify the original tree with the updated values while ensuring the greater sum property.



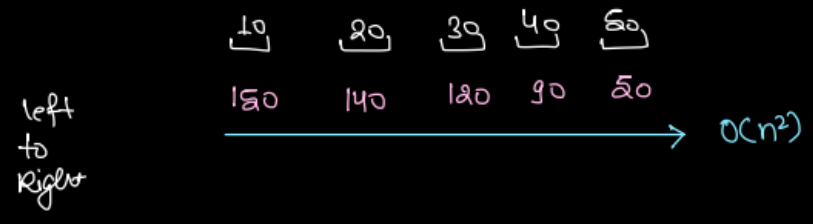
BST:

* Node value is greater than all values in the left side and smaller than all value in right side.
* In order of BST is Sorted.

* If inorder of BST is sorted
* Reverse inorder —



Sum = ~~0~~
~~20~~
~~25~~
~~48~~
~~58~~
~~65~~
~~70~~
72



```java
public class Solution {

    public static int sum;

    public static void helper(TreeNode A) {
        if(A == null) {
            return;
        }

        // move toward right side
        helper(A.right);
        // InArea
        sum += A.val;
        A.val = sum;
        // move toward left side
        helper(A.left);
    }

    public TreeNode solve(TreeNode A) {
        sum = 0;
        helper(A);
        return A;
    }
}
```

Problem: Winner Stone

You are given an array of integers A of size N, where each element represents the weight of a stone. we are playing a game with the stone. On each turn, we choose the two heaviest stones and smash them together. Suppose the stones have weight x and y with x <= y.

The result of this smash is:

1. If x== y, both stones are totally destroyed.
2. If x!=y, the stone of weight x is totally destroyed, and the stone of weight y has new weight y-x.

At the end, three is at most one stone left. Return the weight of this stone (or 0 if there are no stones left)

https://www.interviewbit.com/snippet/8b240e1506bb26d90236/

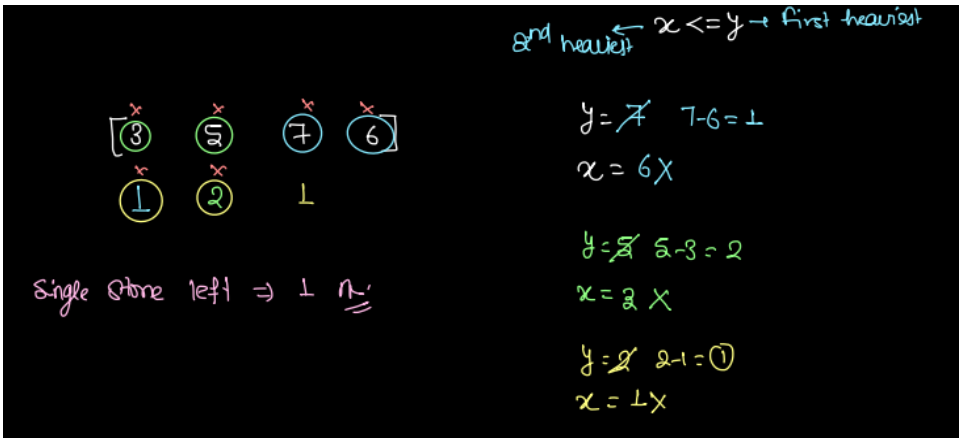
Main logic ->

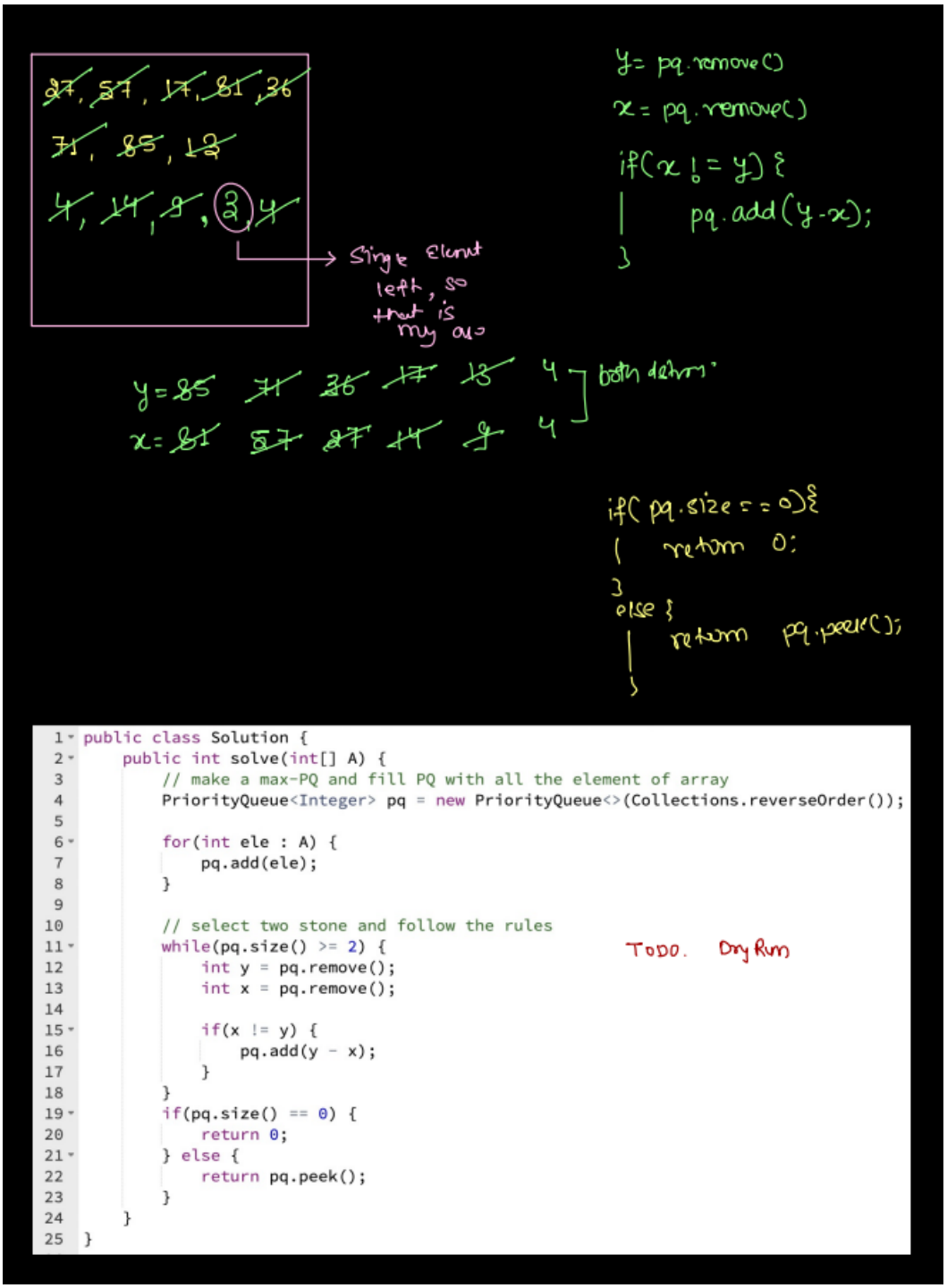
In order to find the winner stone, we will utilize a max priority queue. Here's a breakdown of the steps:

1. Create a function named getLargestTwo that takes an array of integers as input and returns an integer.
2. Initialize a max priority queue named pq for integers using Collections.reverseOrder() to ensure it functions as a max heap.
3. Use a loop to add each element from the input array to the priority queue (pq).
4. Iterate over the priority queue while its size is greater than or equal to 2.
    a. Create two variables, y and x, both integers. Assign y to the value of the first element removed from the priority queue and x to the value of the second element removed.
    b. Check if x is not equal to y, and if true, calculate the difference (y - x) and add it back to the priority queue (pq).

5. After the loop, check if the size of the priority queue pq is 0. If so, return 0. Otherwise, return the highest element in the priority queue using pq.peek().

In the main part of the code, call the getLargestTwo function and return its result.






```java
public class Solution {
    public int solve(int[] A) {
        // make a max-PQ and fill PQ with all the element of array
        PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

        for(int ele : A) {
            pq.add(ele);
        }

        // select two stone and follow the rules
        while(pq.size() >= 2) {
            int y = pq.remove();
            int x = pq.remove();

            if(x != y) {
                pq.add(y - x);
            }
        }
        if(pq.size() == 0) {
            return 0;
        } else {
            return pq.peek();
        }
    }
}
```

Problem: Print Maze Path

You are given the dimensions of a rectangular board AxB. You need to print all the possible paths from top-left corner to bottom-right corner of the board. you can only move down (denoted by 'v') or right (denoted by 'h') at any point in time.

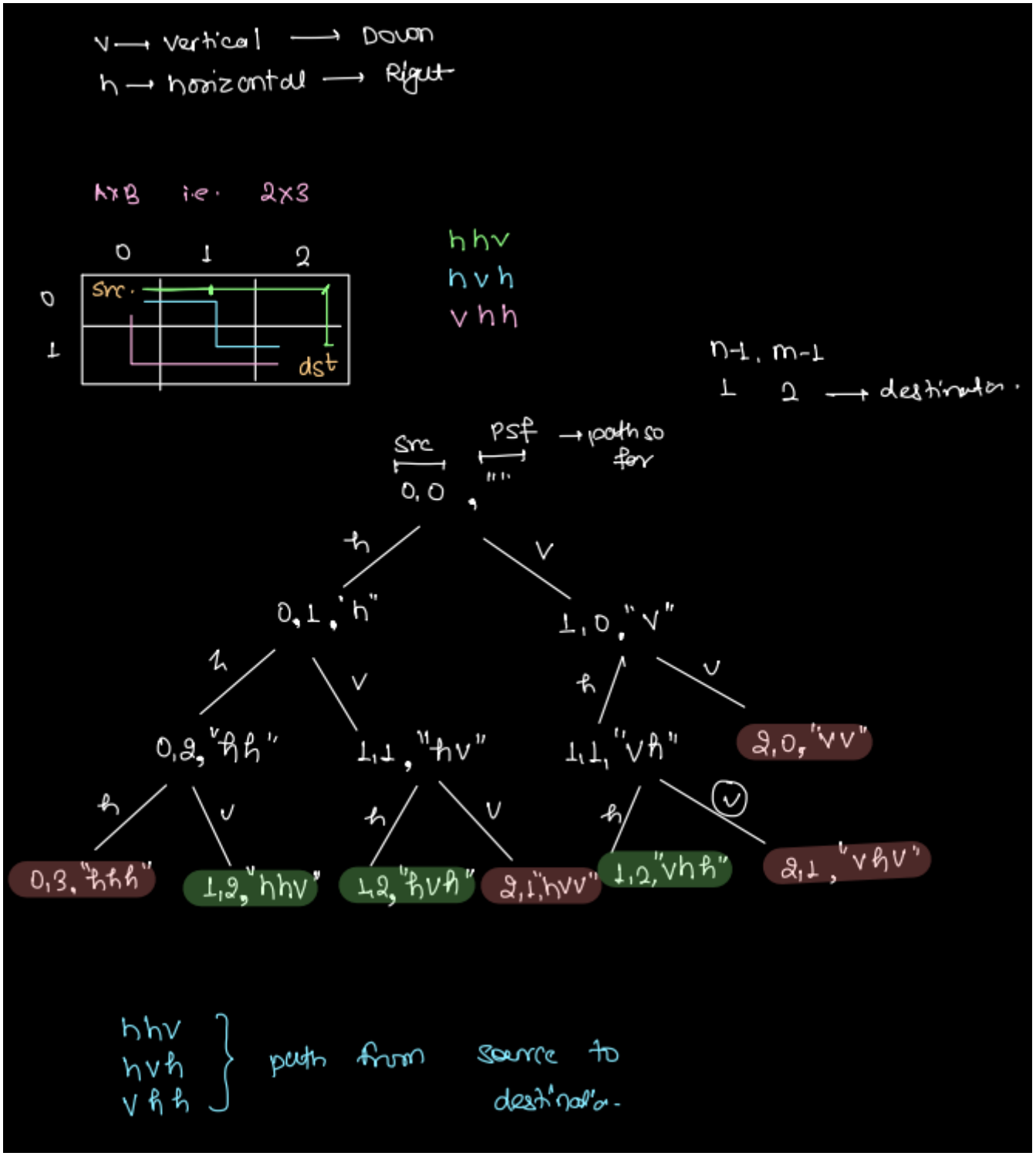
https://www.interviewbit.com/snippet/fa9809b4d580b6cb473c/

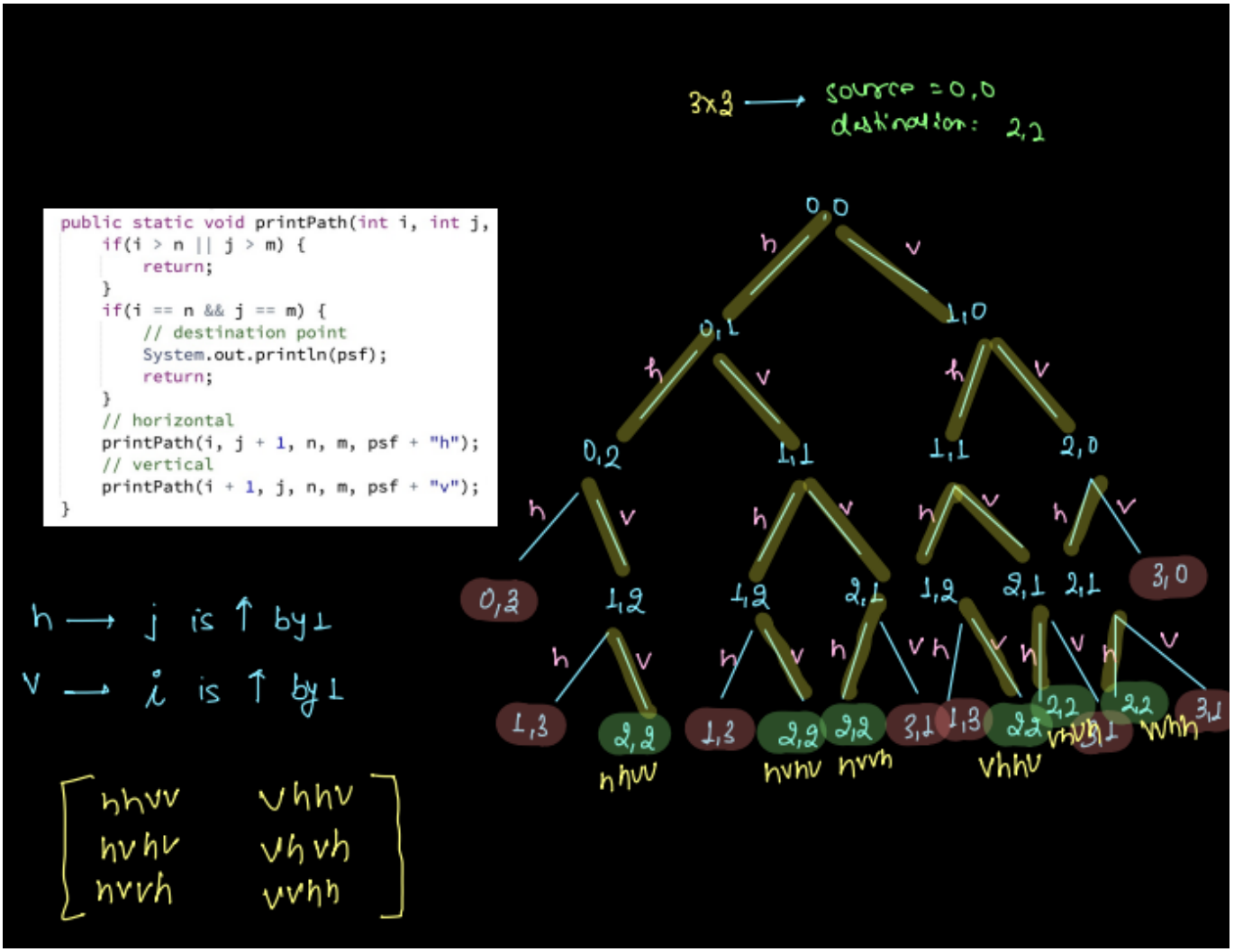
Main Logic ->

Step

1. Create a function named printPath that takes the following parameters: i, j, n, m (all integers), and path (a string). The function has a return type of void.
2. For the base condition, check if i is greater than n or if j is greater than m. If either of these conditions is met, return from the function.
3. For the destination print condition, if i is equal to n and j is equal to m, print the path on a new line using System.out.println and then return from the function.
4. Following the instructions, begin by moving in the horizontal path first, as per the question's requirements. Call the printPath function recursively and pass the arguments i, j + 1, n, m, and path + "h".
5. Next, make another recursive call to the printPath function, passing the arguments i + 1, j, n, m, and path + "v".
6. In the main function, utilize a Scanner to obtain two integer values, a and b. Then, invoke the printPath function, passing the initial values i and j as 0, n and m as a and b respectively, and an empty string as path.

```java
import java.lang.*;
import java.util.*;

public class Main {
    public static void printPath(int i, int j, int n, int m, String psf) {
        if(i > n || j > m) {
            return;
        }
        if(i == n && j == m) {
            // destination point
            System.out.println(psf);
            return;
        }
        // horizontal
        printPath(i, j + 1, n, m, psf + "h");
        // vertical
        printPath(i + 1, j, n, m, psf + "v");
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int A = scn.nextInt();
        int B = scn.nextInt();
        // make a call to print all possible path in A*B rectangular board
        printPath(0,0, A-1, B-1, "");
    }
}
```


v → vertical → Down
h → horizontal → Right
AxB i.e. 2x3
0 1 2
0
1
src
dst
hhv
hvh
vhh
n-1, m-1
1 2 → destination.
Src
0,0
psf → path so far
h
v
0,1, "h"
1,0, "v"
0,2, "hh"
1,1, "hv"
1,1, "vh"
2,0, "vv"
0,3, "hhh"
1,2, "hhv"
1,2, "hvh"
2,1, "hvv"
1,2, "vhh"
2,1, "vhv"
hhv
hvh
vhh
path from source to destination.



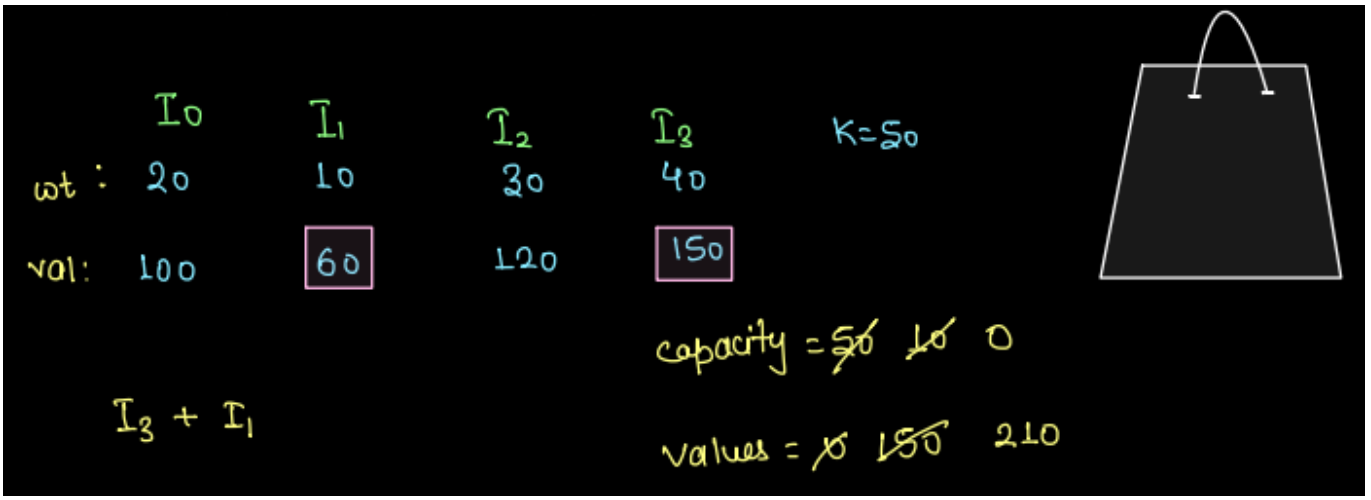
DSA: DP-3 - 30 - Aug 2023

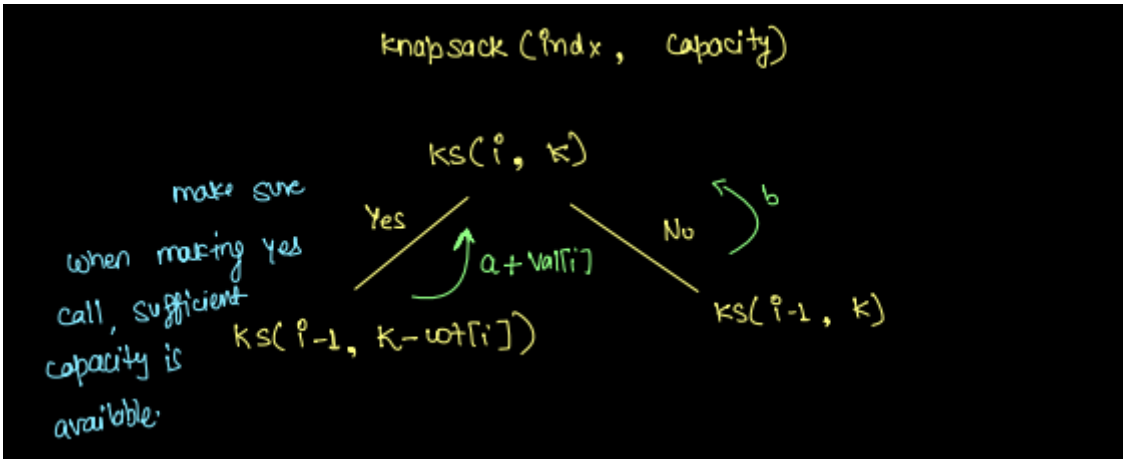
- 01 Knapsack
- unbounded knapsack
- Tabulation DP
- Fibonacci in Tabulation
- count of path in tabulation
- Min Cost Path Tabulation
- Knapsack Tabulation

0-1 Knapsack
Given N items and K capacity, each with a weight and value, find max value which can be optained by picking items such that total weight of picket items <= K.
Note:
i. Every item can be picked at max one time.
ii. We can't take part OR fraction of item.

wt = [20, 10, 30, 40]
value = [100, 60, 120, 150]
k = 50

https://www.interviewbit.com/snippet/69bc8334dbc923740712/

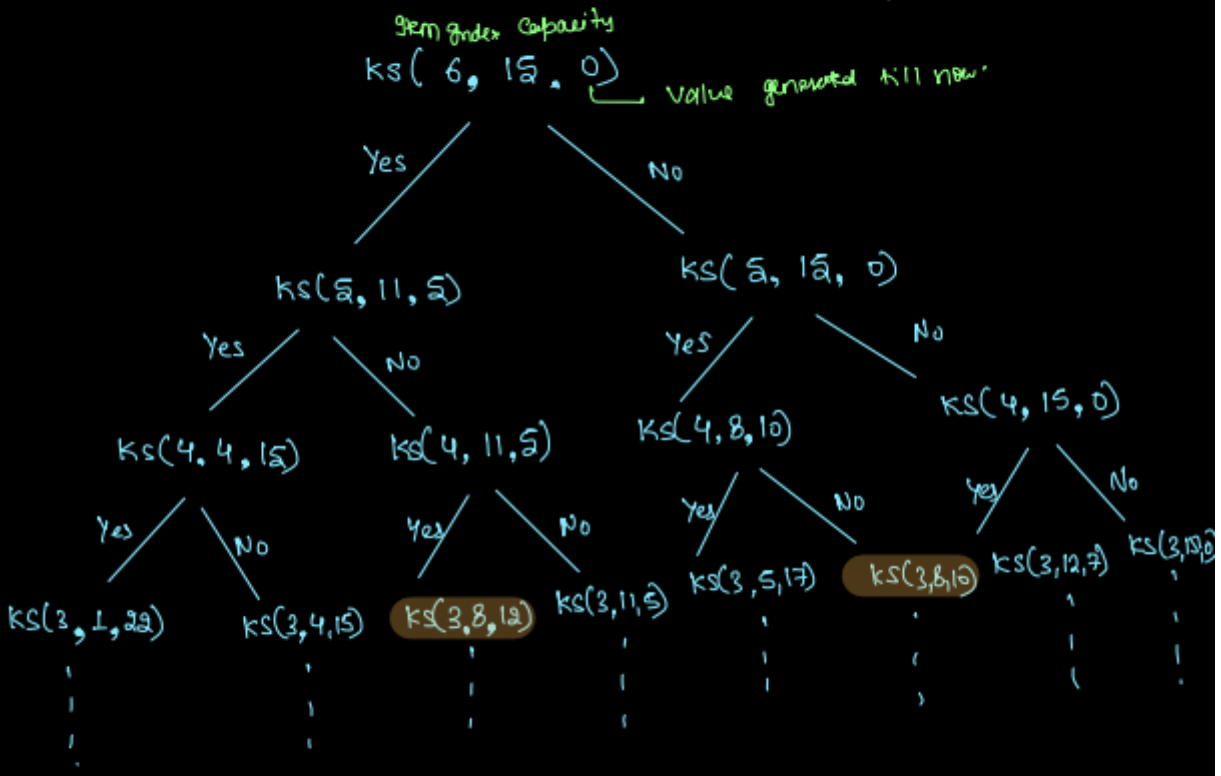




Logic of this problem is exactly similar to all subsequence, but we have to take care of capacity as well.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	$K=15$
wt:	4	1	5	4	3	7	4	
value:	3	2	8	9	7	10	5	

i



We can apply DP since we are going to solve already solved problem.

dp[ ][ ]
↑ item index
↳ capacity ⟶ dp[ ][ ] = new int[totalItems][cap+1];

dp[2][5] → max. profit we can generate from 2nd item with capacity 5.

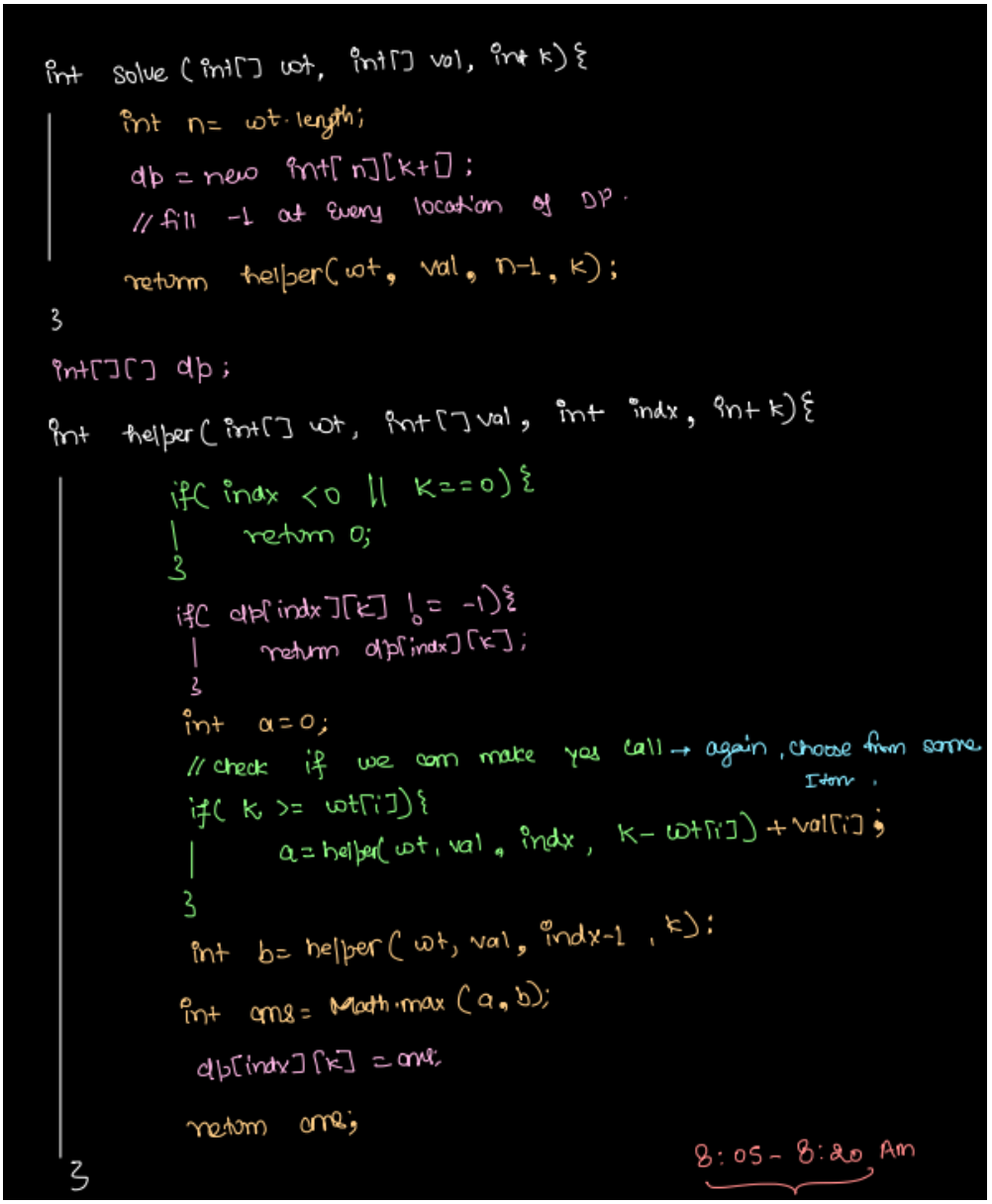
```
int Solve (int[] wt, int[] val, int k){
    int n= wt.length;
    dp = new int[n][k+1];
    // fill -1 at every location of DP.
    return helper(wt, val, n-1, k);
}
int[][] dp;
int helper(int[] wt, int[] val, int indx, int k){
    if( indx <0 || k==0){
        return 0;
    }
    if( dp[indx][k] != -1){
        return dp[indx][k];
    }
    int a=0;
    // check if we can make val call
    if( k >= wt[i]){
        a= helper(wt, val, indx-1, k- wt[i]) + val[i];
    }
    int b= helper(wt, val, indx-1, k);
    int ans= Math.max(a, b);
    dp[indx][k] =ans;
    return ans;
}
```

problem2

Unbounded Knapsack

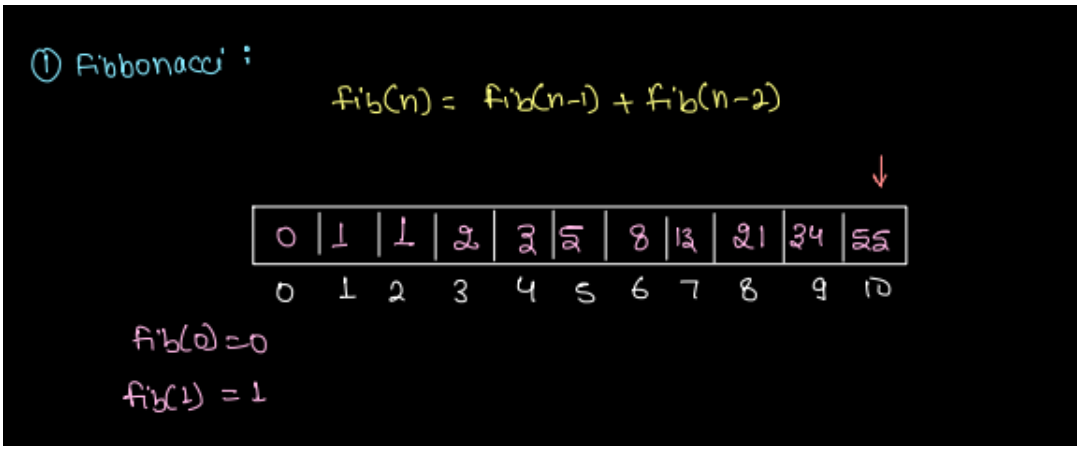
Given N items and K capacity, each with a weight and value, find max value which can be obtained by picking items such that total weight of picked items <= K.
Note:
i. Every Item can be picked unlimited no. of time.
ii. We can't take part Or fraction of item.

https://www.interviewbit.com/snippet/bb319f90bd9116958661/

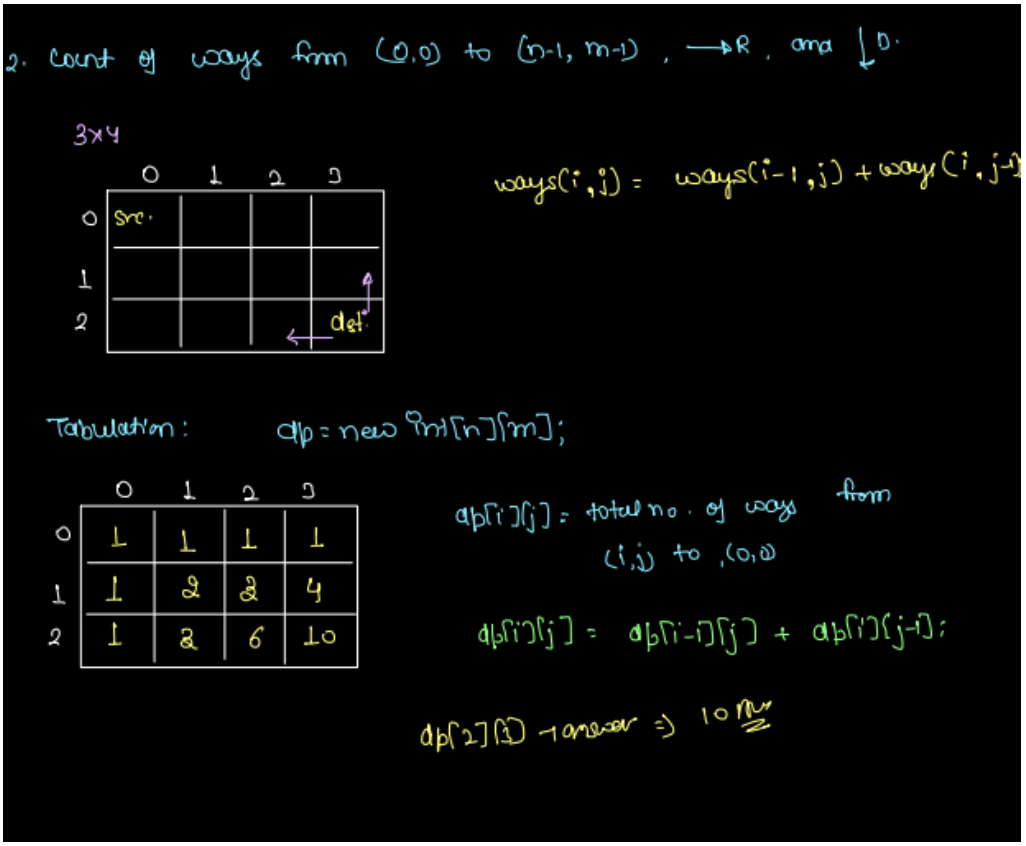


Fibonacci - using tabular dp

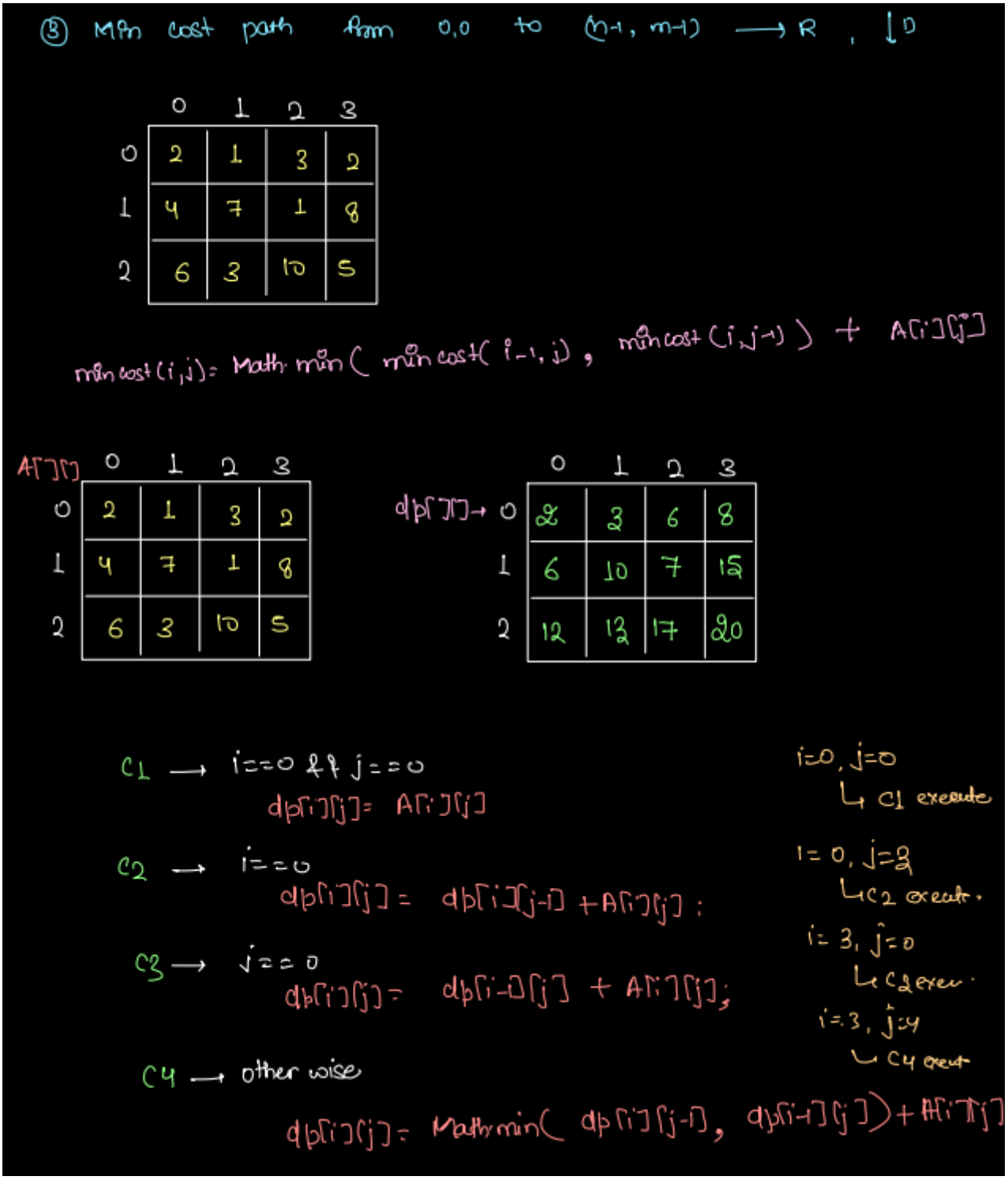
https://www.interviewbit.com/snippet/706d237066cc5bf92dcb/



No of ways to react source to destination in grid
https://www.interviewbit.com/snippet/25cb7962fbf08e0efb6a/



Min cost path from 0,0 to (n-1,m-1)
https://www.interviewbit.com/snippet/2f9e271495a5e1556bd7/


③ Min cost path from 0,0 to (m-1, m-1) → R , ↓D
0 1 2 3
0 2 1 3 2
1 4 7 1 8
2 6 3 10 5
minCost(i,j) = Math.min(minCost(i-1, j) , minCost(i, j-1)) + A[i][j]
A[][]
dp[][]→
C1 → i==0 && j==0
dp[i][j] = A[i][j]
C2 → i==0
dp[i][j] = dp[i][j-1] + A[i][j] ;
C3 → j==0
dp[i][j] = dp[i-1][j] + A[i][j];
C4 → other wise
dp[i][j] = Math.min(dp[i][j-1], dp[i-1][j]) + A[i][j]
i=0, j=0
↳ C1 execute
i=0, j=2
↳ C2 execute.
i=3, j=0
↳ C3 exec.
i=3, j=4
↳ C4 exec

Knap sack

4. 0 1 Knapsack

cap = 7

	I_0	I_1	I_2	I_3
wt:	3	6	4	2
val:	12	20	15	6

dp[][] = new int[total item][capacity + 1]

4 * 8

cap+1

total item

wt	val		0	1	2	3	4	5	6	7
3	12	I_0	0	0	0	12	12	12	12	12
6	20	I_1	0	0	0	12	12	12	20	20
4	15	I_2	0	0	0	12	15	15	20	27
2	6	I_3	0	0	6	12	18	18	21	27

capacity →

item ↓

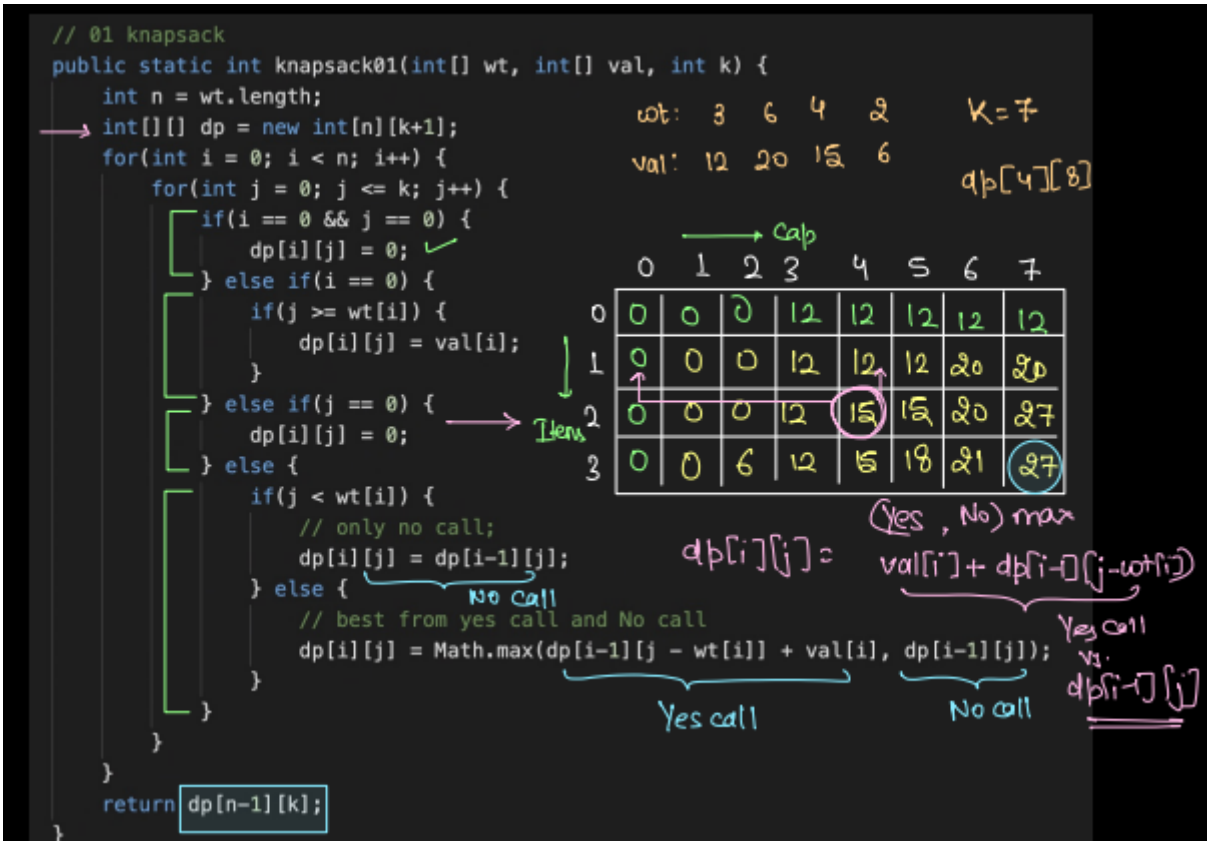
dp[i][j] → max value we can generate till ith index with j capacity.

dp[2][4] = (dp[1][4-4] + 15 , dp[1][4])

dp[i][k] = Math.max(dp[i-1][k - wt[i]] + val[i] , dp[i-1][k]);

If yes call made

No call



DSA: DP-4 - 31 - Aug 2023

- Longest Common Subseq.
- LCS : Tabulation,
- Longest Palindromic Subseq.
- Edit Distance

Longest Common Subsequence
Given two strings A and B, find lenght longest common subsequence.
S1: a,b,b,c,d,g,h
s2: b,a,c,h,e,g,f

https://www.interviewbit.com/snippet/2aae2f2b4e9e6838f318/

eg 1

S1 : a b b c d g h → bcg

S2 : b a c h e g f → bcg

→ longest common subseq.

bcg OR bch

ans = 3.

eg 2

S1 : a b b c d g f → acgf

S2 : a c h e g f → acgf

→ ans = 4

Idea 1: Brute force:

S1 → a b d

S2 → a e b

	a	e	b	ae	ab	eb	aeb
0	1	2	3	4	5	6	7

S2

Hashset

S1 →

-
a
b
d
ab
ad
bd
abd

length = ~~0~~ ~~1~~ 2 Ans.

Steps:

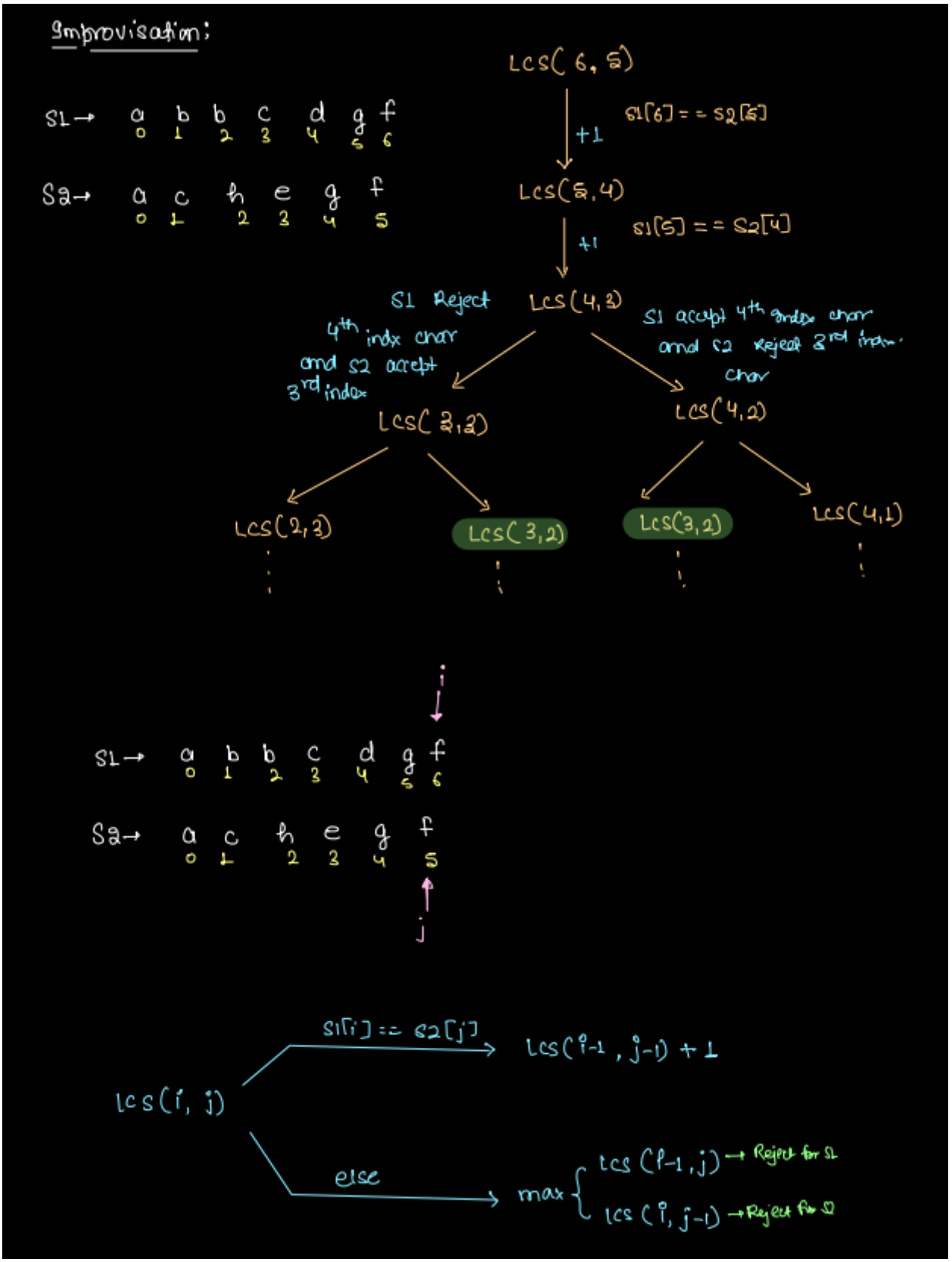
① make all subseq for S1 and put it in Hashset.

② make all subseq for S2 and put it in AL.

③ Start from index 0 in AL and check if it is available in set or not. If all maximise length.

T.C : $\underbrace{2^n}_{\text{for H.S}} + \underbrace{2^n}_{\text{for AL}} + \underbrace{2^n}_{\text{for checking}} = O(2^n)$

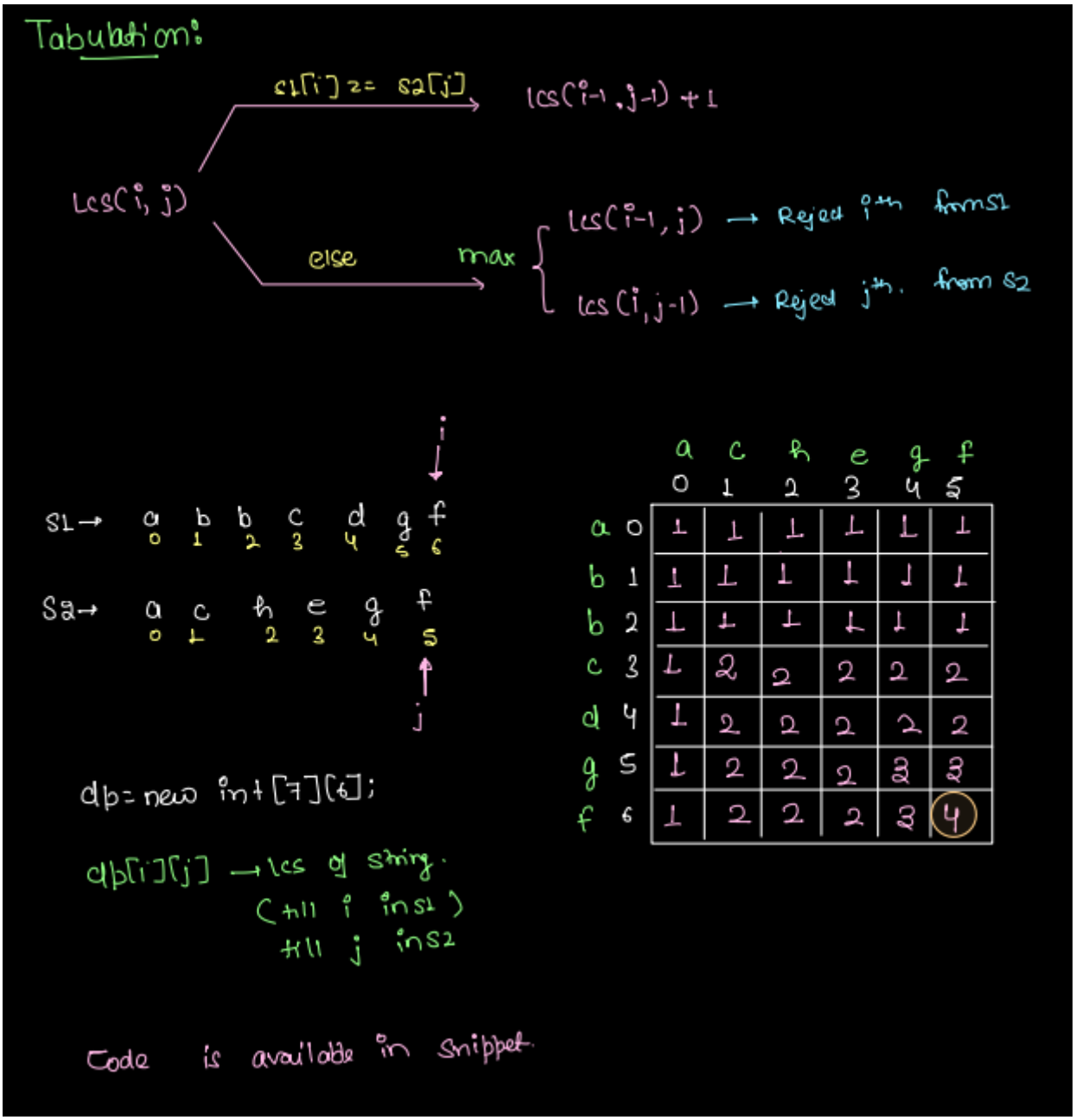
S.C : $O(2^n)$


Improvisation:
LCS(6, 5)
S1 → a b b c d g f
0 1 2 3 4 5 6
S2 → a c h e g f
0 1 2 3 4 5
S1[6] == S2[5]
+1
LCS(5,4)
S1[5] == S2[4]
+1
LCS(4,3)
S1 Reject 4th indx char and S2 accept 3rd index
S1 accept 4th index char and S2 reject 3rd index char
LCS(3,3)
LCS(4,2)
LCS(2,3)
LCS(3,2)
LCS(3,2)
LCS(4,1)
i
j
S1[i] == S2[j]
LCS(i-1, j-1) + 1
LCS(i, j)
else
max
LCS(i-1, j) → Reject for S1
LCS(i, j-1) → Reject for S2

```java
class Main {

    public static int lcs_Helper(String s1, String s2, int i, int j) {
        if(i < 0 || j < 0) {
            return 0;
        }

        if(dp[i][j] != -1) {
            return dp[i][j];
        }

        int ans = 0;
        if(s1.charAt(i) == s2.charAt(j)) {
            ans = lcs_Helper(s1, s2, i-1, j-1) + 1;
        } else {
            int l1 = lcs_Helper(s1, s2, i-1, j);
            int l2 = lcs_Helper(s1, s2, i, j-1);
            ans = Math.max(l1, l2);
        }
        return dp[i][j] = ans;
    }

    public static int[][] dp;

    public static int lcs(String s1, String s2) {
        int n = s1.length();
        int m = s2.length();

        dp = new int[n][m];
        // fill dp with -1
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < m; j++) {
                dp[i][j] = -1;
            }
        }

        return lcs_Helper(s1, s2, n-1, m-1);
    }

    public static void main(String args[]) {
        // String s1 = "abbcdgf";
        // String s2 = "achegf";

        String s1 = "abbcdgh";
        String s2 = "bachegf";

        System.out.println(lcs(s1, s2));
    }
}
```

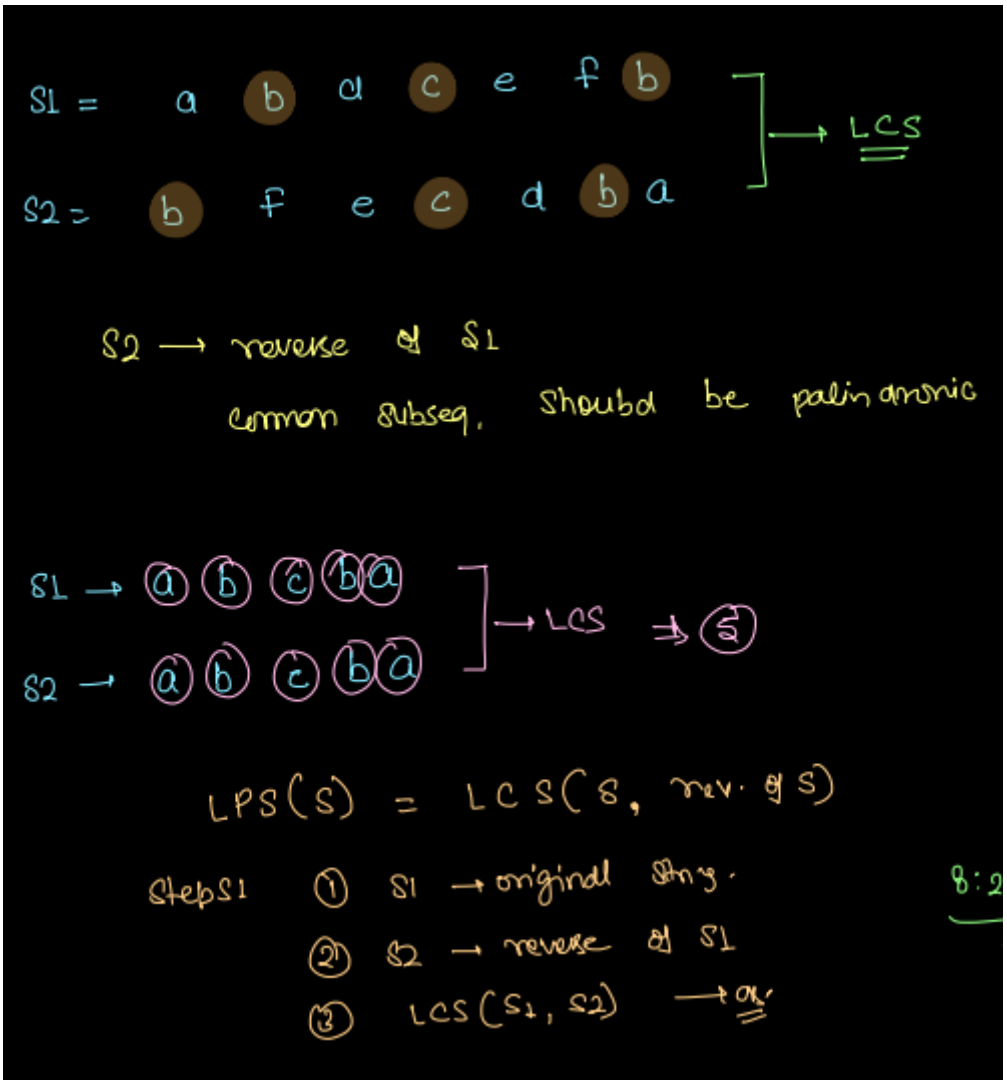
T.C: O(n*m)

S.C: O(n*m)



Longest Palindromic Subsequence
Given B, find lenght longest Palindromic subsequence.
A: s c a l a r
A: a b d c e f b

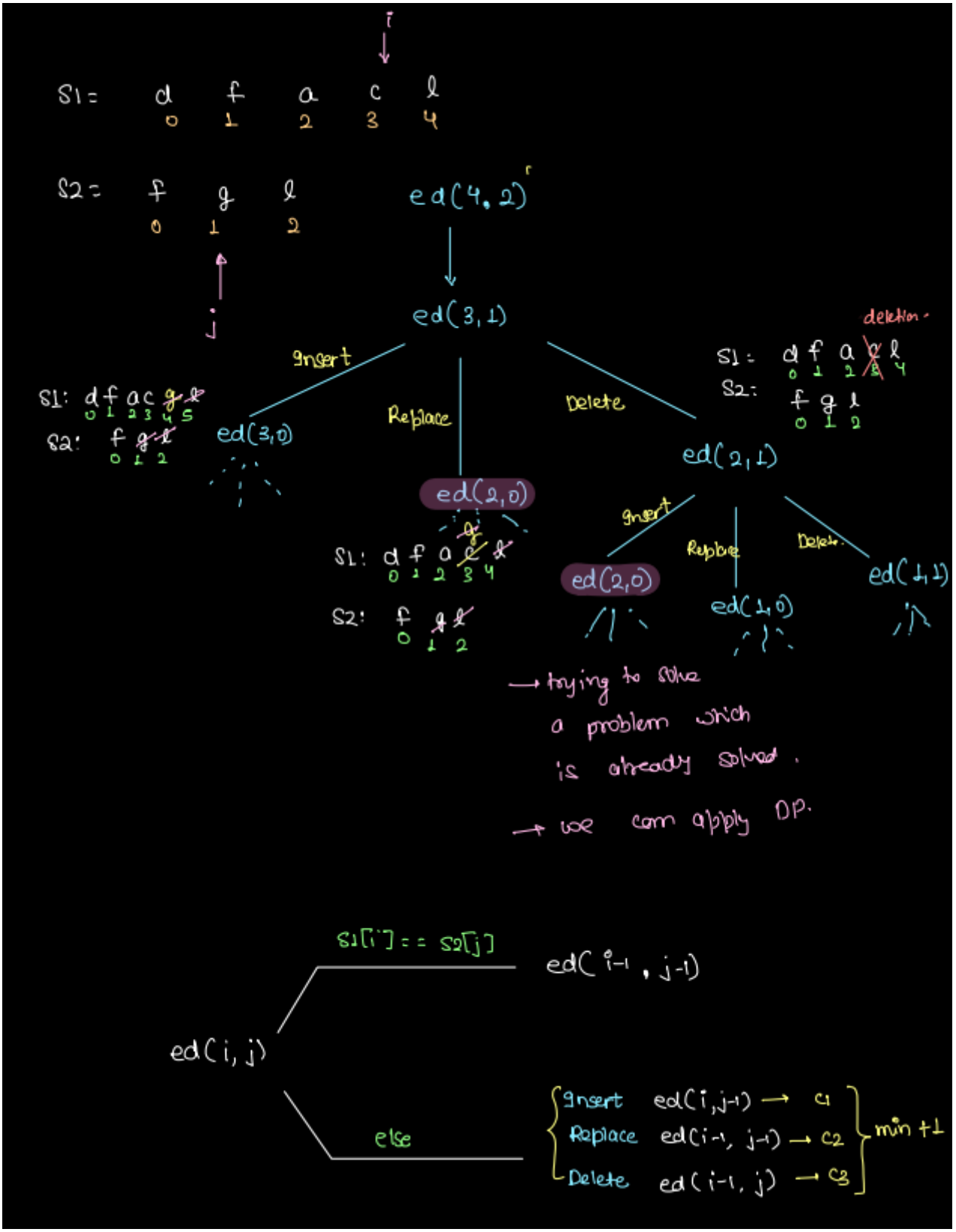
https://www.interviewbit.com/snippet/23c734e9d0bbcfb21c52/

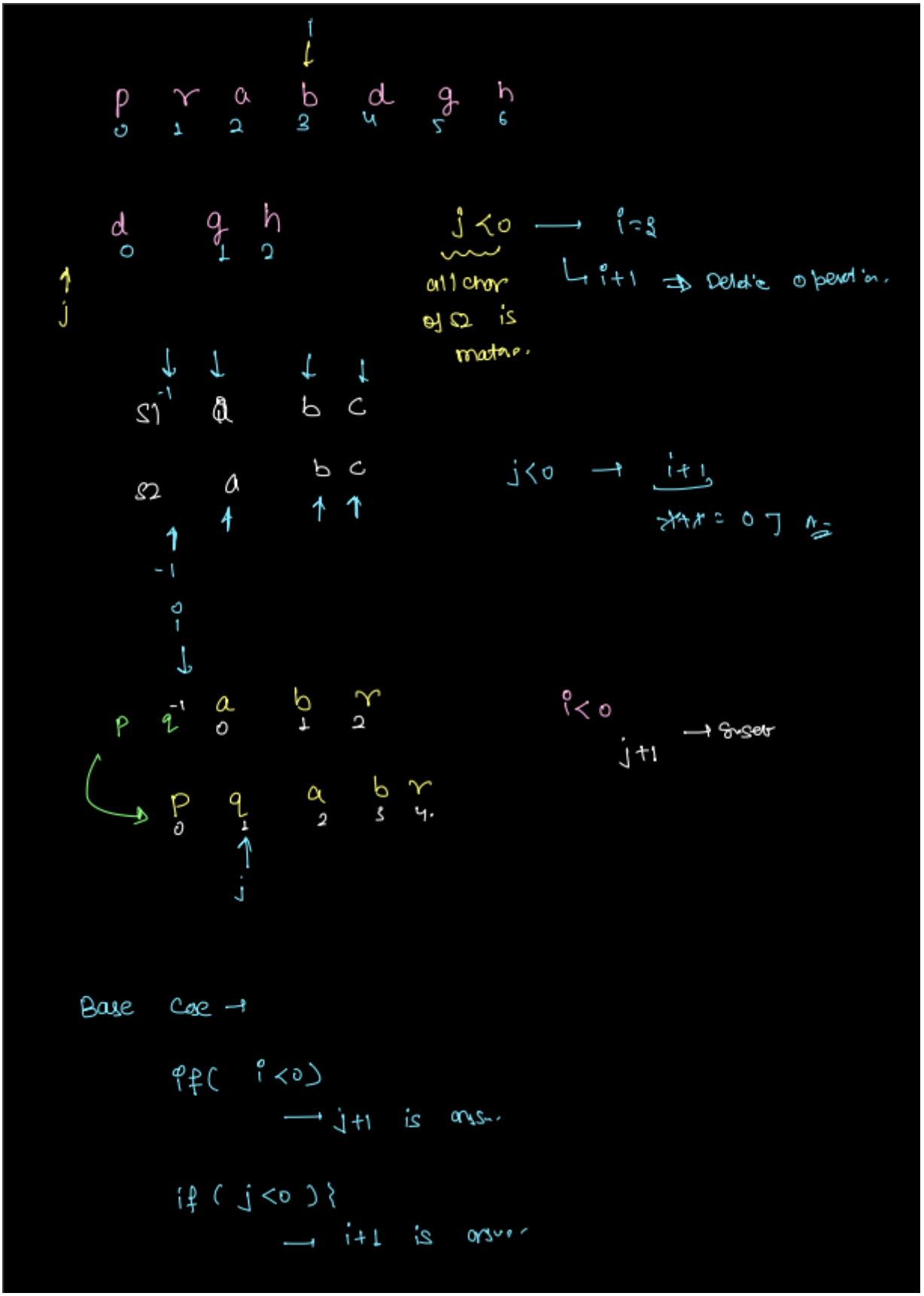


Edit Distance
Given two strings s1 and s2, minimum operations to be performed in s1 so that it becomes equals to s2.
Operations allowed in s1:
i. We can insert any char in s1 at any postion.
ii. We can replace any char in s1 at any postion.
iii. we can delete any char in s1 at any postion.

s1 = d f d c i
s2 = f g i

https://www.interviewbit.com/snippet/6001f5a820bf9ec6e90d/


i
S1 = d f a c l
0 1 2 3 4
S2 = f g l
0 1 2
j
ed(4, 2)
ed(3, 1)
Insert
Replace
Delete
S1: d f a c g l
0 1 2 3 4 5
S2: f g l
0 1 2
ed(3,0)
ed(2,0)
S1: d f a c
0 1 2 3
S2: f
0
ed(2,1)
Insert
Replace
Delete
S1: d f a c l
0 1 2 3 4
S2: f g l
0 1 2
deletion
ed(2,0)
ed(1,0)
ed(1,1)
→ trying to solve a problem which is already solved.
→ we can apply DP.
S1[i] == S2[j]
ed(i-1, j-1)
ed(i, j)
else
Insert ed(i,j-1) → c1
Replace ed(i-1, j-1) → c2
Delete ed(i-1, j) → c3
min +1


p r a b d g h
0 1 2 3 4 5 6
d g h
0 1 2
j
j < 0 → i = 2
all char of s2 is match.
↳ i+1 ⇒ Delete operation.
s1 a b c
s2 a b c
-1
0
1
p q a b r
-1 0 1 2
p q a b r
0 1 2 3 4
j
i < 0
j+1
→ Insert
Base Case →
if(i < 0)
→ j+1 is ans.
if (j < 0){
→ i+1 is ans.

```java
class Main{
    public static int[][] dp;
    public static int minOperations(String str1, String str2, int i, int
        if(i < 0 ){
            return j + 1;
        }
        if(j < 0){
            return  i + 1;
        }

        if(dp[i][j] != -1){
            return dp[i][j];
        }
        if(str1.charAt(i) == str2.charAt(j)){
            return dp[i][j] = minOperations(str1,str2,i-1,j-1);
        }else{
            // a = insert, b = repalce, c = delete
            int a = minOperations(str1,str2,i,j-1);
            int b = minOperations(str1,str2,i-1,j-1);
            int c = minOperations(str1,str2,i-1,j);
            return dp[i][j] = Math.min(a,Math.min(b,c)) + 1;
        }

    }

    public static void main(String args[]) {
        String A = "dfaex";
        String B = "fxz";
         int n = A.length();
         int m = B.length();
        dp = new int[n][m];

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                dp[i][j] = -1;
            }
        }
        System.out.print(minOperations(A,B, n-1, m-1));
    }
}
```

DSA: DP-5 - 2 - Sep 2023

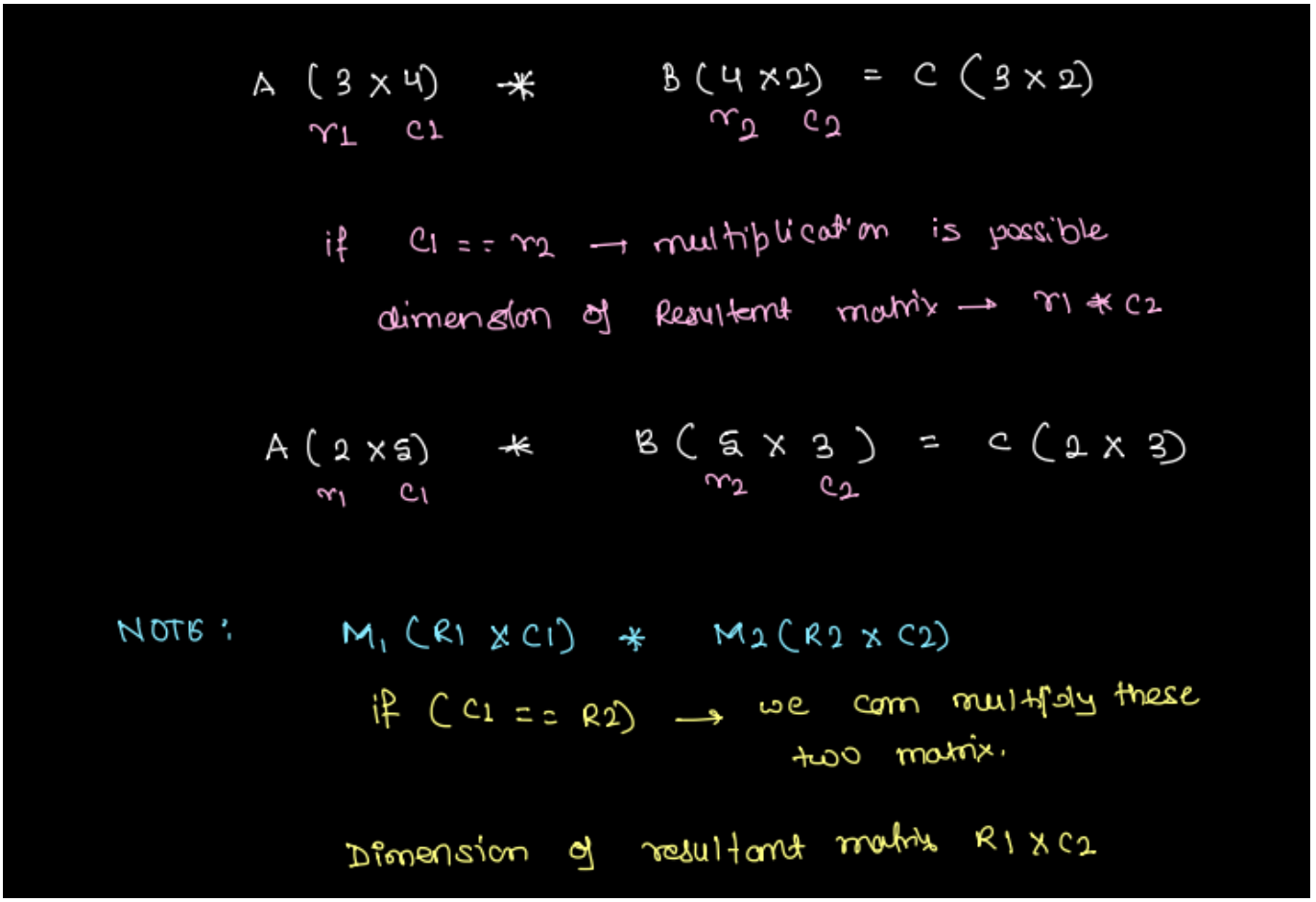
- **Matrix Multiplication**
- **Matrix Chain Multiplication**
- **Longest Increasing Subseq.**
- **Stock Buy Sell**

Matrix Multiplication Basics ->

Note: It is required to multiuply 2 metrics first matrics column equals to second matrics row like below dimention c1 == r2 and result should be r1 x c2.

3 x 4 , 4 x 2

r1 x c1, r2 x c2



Representation of Dimension of matrix:

arr[] → [3 5 7 4] (indices 0 1 2 3)

M1 M2 M3

3x5 5x7 7x4

arr[] → [1 2 3] (indices 0 1 2)

M1 M2

1x2 2x3

A[] → [2 3 4 2 6 5] (indices 0 1 2 3 4 5)

m1 m2 m3

m1 * m2 * m3

(2x3) * (3x4) * (4*2)

(2x4) * (4 X2)

2x2

Dimension of resultant matrix if we multiply matrix from 0 to 2 => 2 X 2

Dimension of resultant matrix if we multiply matrix from 2 to 5 => 4 X 5

Generalise it, dimension of resultant matrix b/w i - j index = arr[i] x arr[j]

A[] → [2 3 4 2 6 5] (indices 0 1 2 3 4 5)

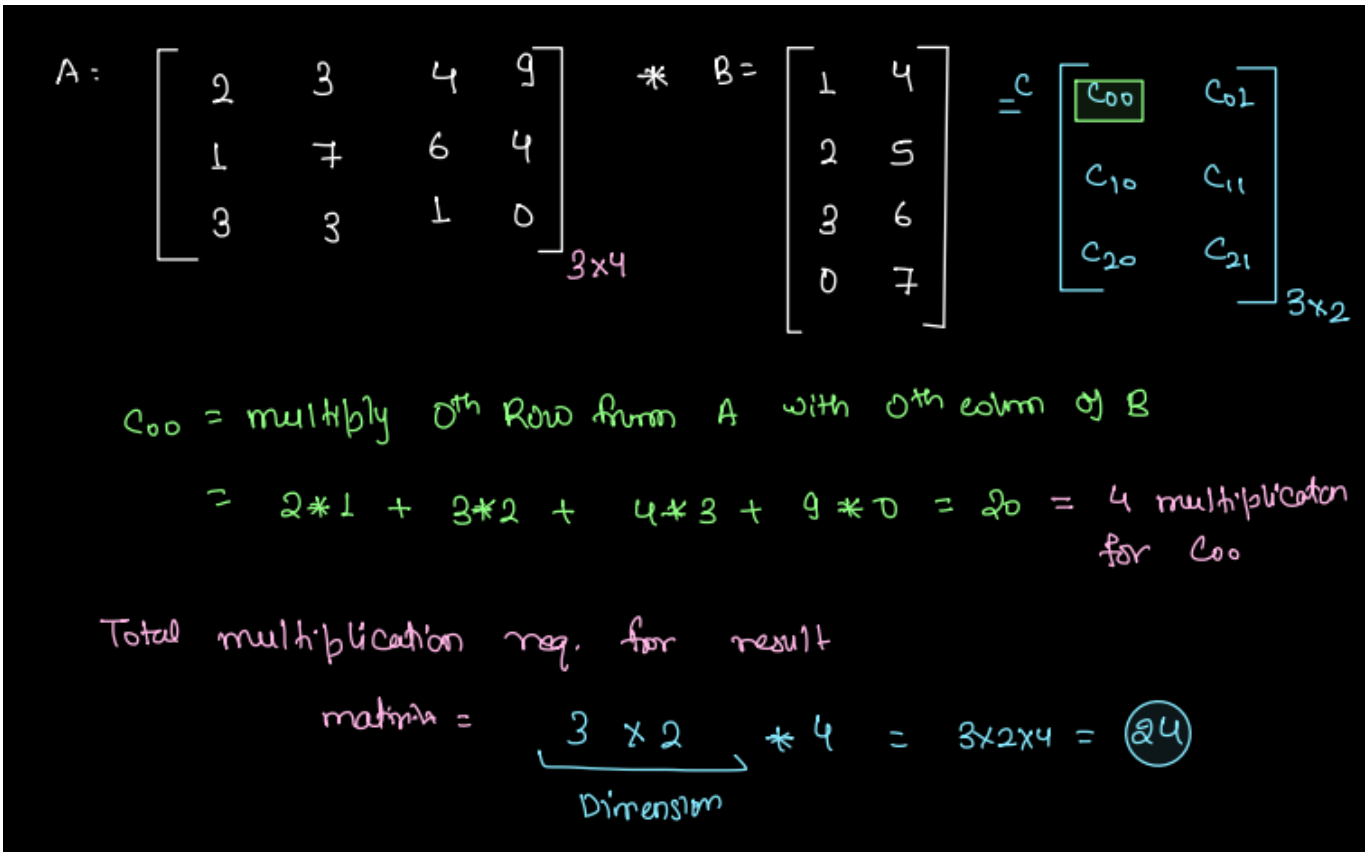
m1 m2 m3

m1 * m2 * m3

= (4 X 2) * (2 X 6) * (6 X 5)

= (4 x 6) * (6 X 5)

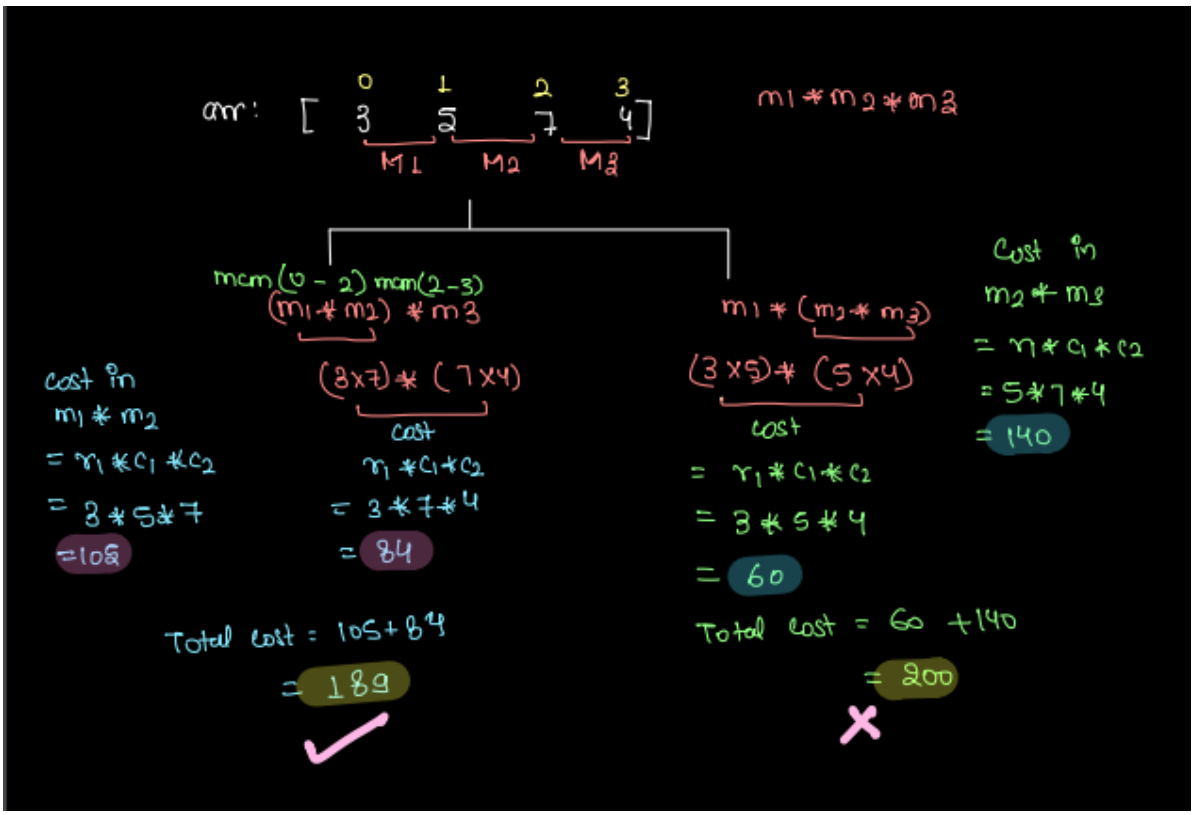
= (4 x 5) Dimension

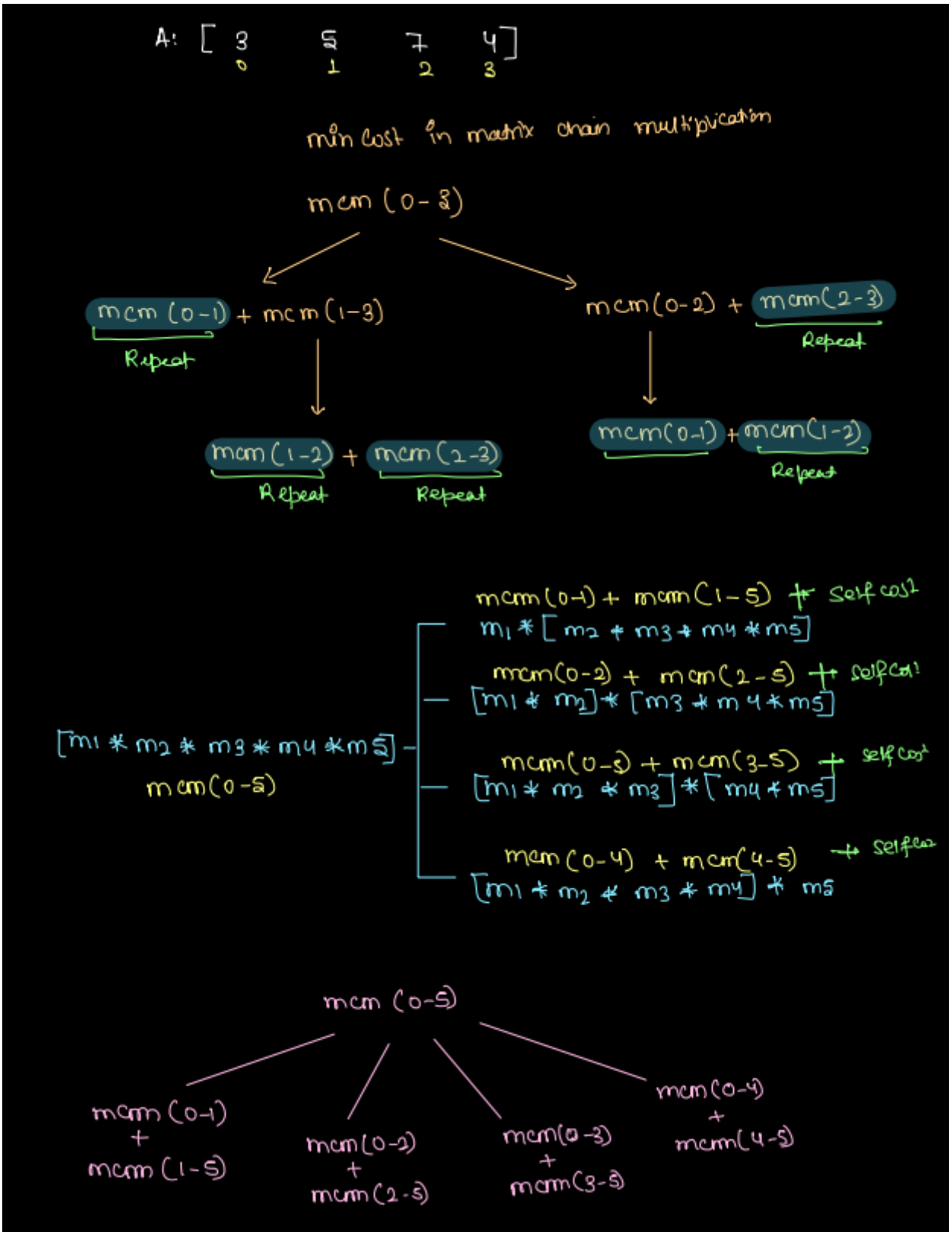


Problem: Matrix chain multiplication

Given an array of integers A representing chain of 2-D matrics such that the dimentions of ith matrix is A[i-1] x A[i]. Find the most effecient way to multiply these martics together. The probelm is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

Return the minimum number of multiplicaitons needed to multiply the chain.




A: [3 5 7 4]
0 1 2 3
min cost in matrix chain multiplication
mcm (0-3)
mcm (0-1) + mcm (1-3)
Repeat
mcm (0-2) + mcm (2-3)
Repeat
mcm (1-2) + mcm (2-3)
Repeat Repeat
mcm(0-1) + mcm(1-2)
Repeat
[m1 * m2 * m3 * m4 * m5]
mcm(0-5)
mcm(0-1) + mcm(1-5) + self cost
m1 * [m2 * m3 * m4 * m5]
mcm(0-2) + mcm(2-5) + self cost
[m1 * m2] * [m3 * m4 * m5]
mcm(0-3) + mcm(3-5) + self cost
[m1 * m2 * m3] * [m4 * m5]
mcm(0-4) + mcm(4-5) + self cost
[m1 * m2 * m3 * m4] * m5
mcm (0-5)
mcm (0-1) + mcm (1-5)
mcm(0-2) + mcm(2-5)
mcm(0-3) + mcm(3-5)
mcm(0-4) + mcm(4-5)

mcm (i, j) K = i+1 to K < j

Self cost

Min is ans for mcm (i, j)

- K = i+1 = mcm(i, i+1) + mcm(i+1, j) + arr[i] * arr[k] * arr[j]
- K = i+2 = mcm (i, i+2) + mcm(i+2, j) + arr[i] * arr[k] * arr[j]
- K = i+3 = mcm (i, i+3) + mcm (i+3, j) + arr[i] * arr[k] * arr[j]
- ⋮
- K = j-1 = mcm (i, j-1) + mcm(j-1, j) + arr[i] * arr[k] * arr[j]

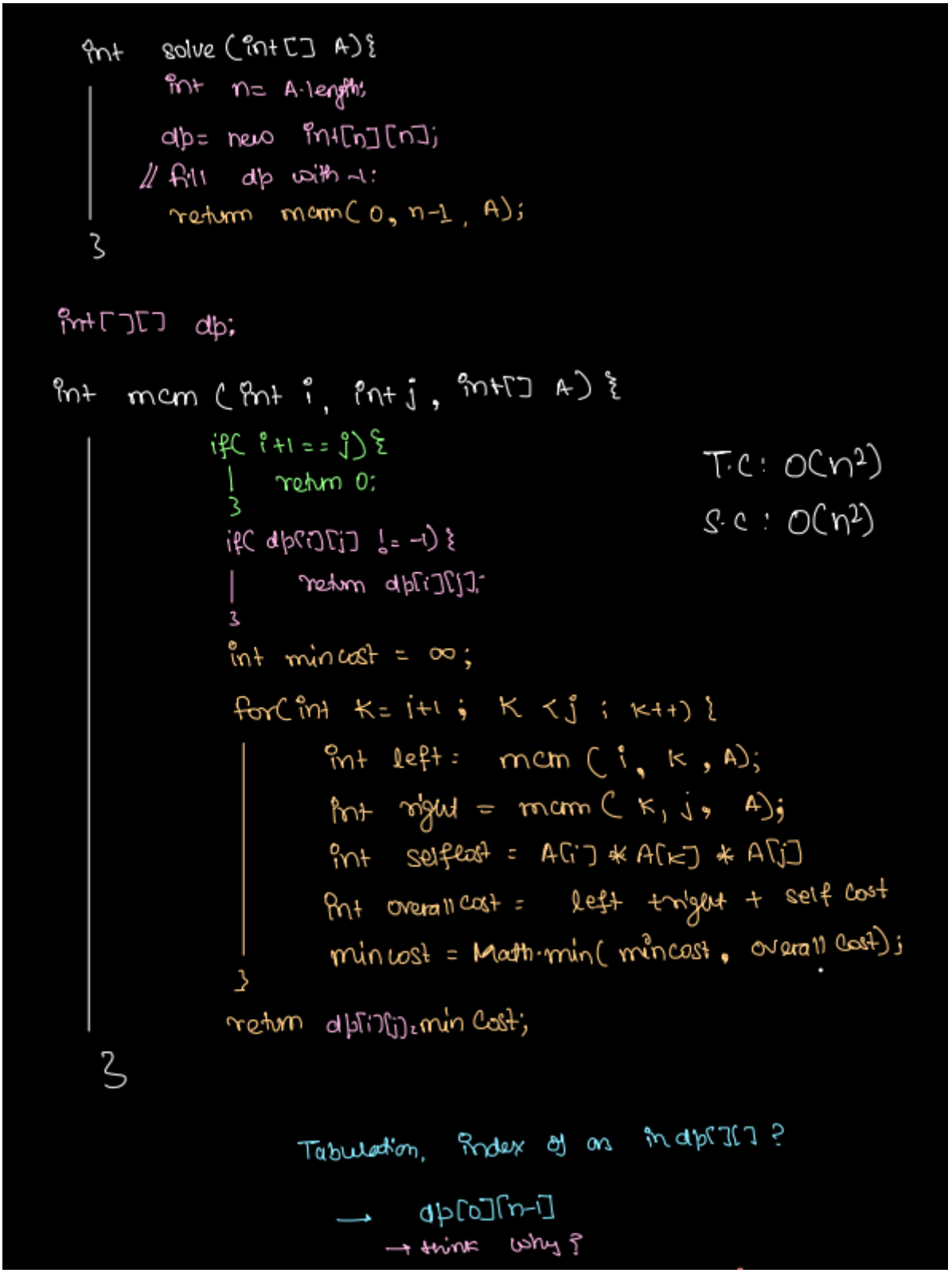
mcm (i, i+3) + mcm (i+3, j)

i, K — left; K, j — Right

dimension → arr[i] X arr[k] arr[k] X arr[j]

cost of self mcm = arr[i] * arr[k] * arr[j]

```
int mcm (int i, int j, int[] A) {
    if( i+1 == j){
        return 0;
    }
    int mincost = ∞;
    for(int K = i+1 ; K < j ; K++) {
        int left = mcm ( i, K , A);
        int right = mcm ( K, j, A);
        int selfcost = A[i] * A[K] * A[j]
        int overallcost = left + right + self cost
        mincost = Math.min( mincost, overall cost);
    }
    return minCost;
}
```



Problems: Longest increasing subsequence of a given array of integers, A. In other words, find a subsequence of array in which the subsequence's elements are in strictly incresing order, and in which the subseuence is as long as possible. In this case, return the lenght of the longest increasing subsequence.

A[] : [9,2,4,3,10]

A: [2 -1 6 3 7 9]

2 6 7 9
2 3 7 9
-1 3 7 9
-1 6 7 9

} longest = 4 Ans.

A: [2 3 8 1 3]

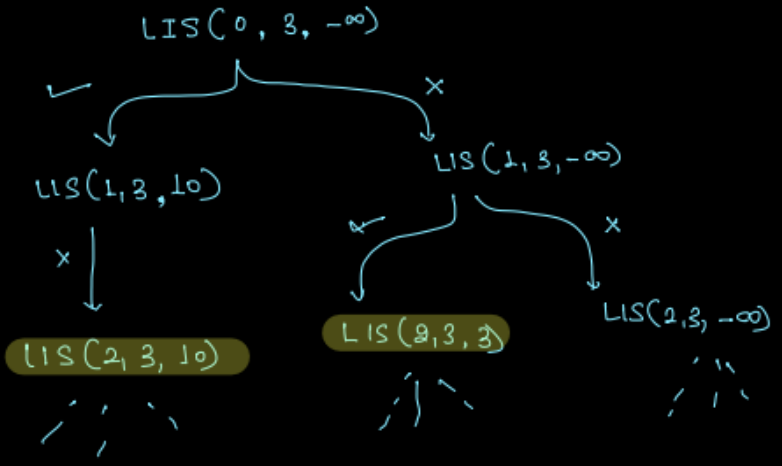
2, 3, 8 → ans = 3

Brute force: Generate all subseq. and check if it is strictly increasing or not. If it is strictly increasing maximise length.

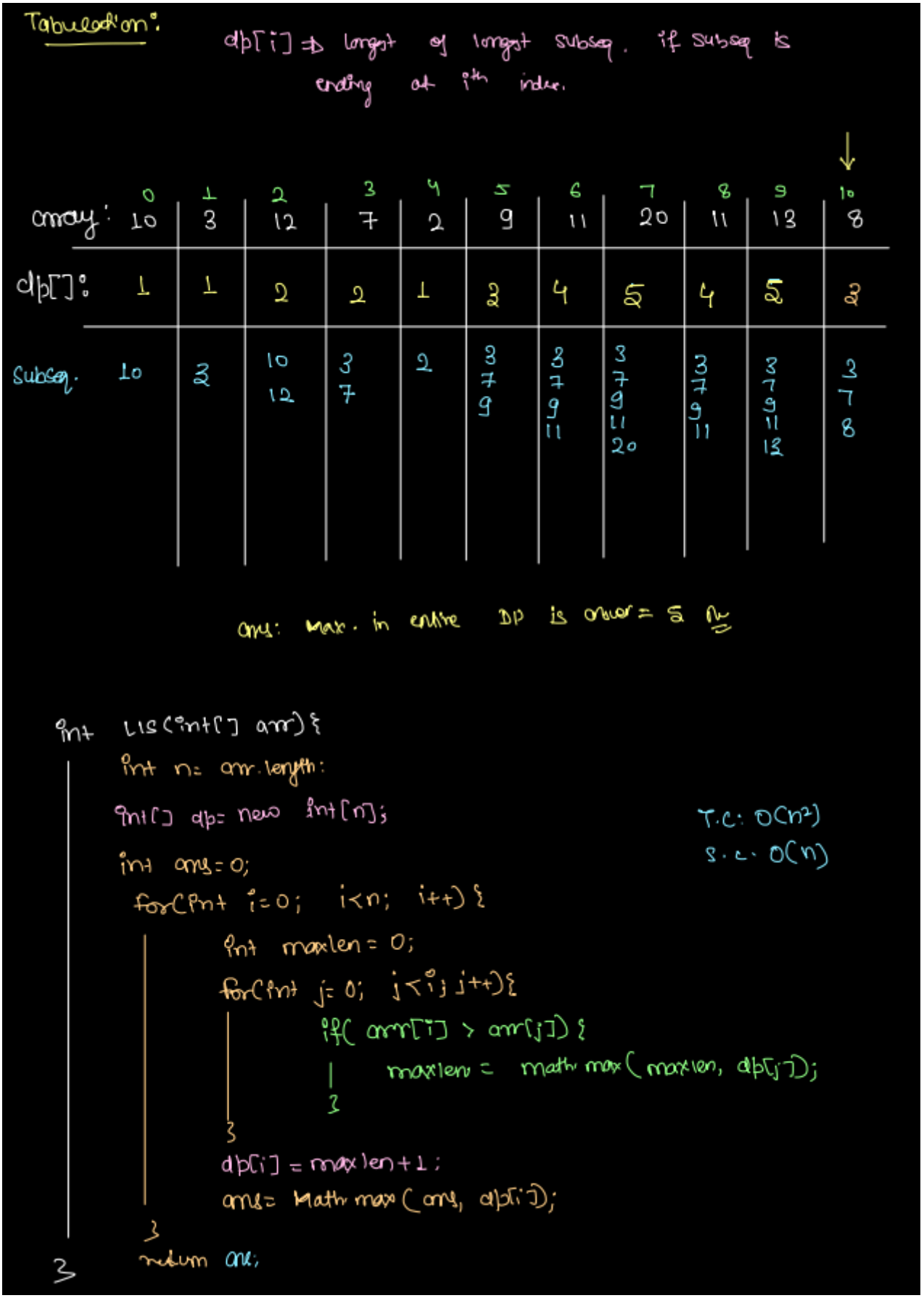
T.C: $2^n * n$
- 2^n: for generation of subseq
- n: check if it is in increasing order

Optimise thinking

[10 3 12 7]
0 1 2 3



NOTE: Since in memoisation, structure of Recursion and dependency on Recursion is complex, so it is not easy way to calculate LIS.



Stock buy and sell

Say you have an array, A, for which the ith element is the price of a given stock on day i. If you were only permitted to complte at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit. return the maximum possible profit.

price : [3 5 2 1 4 5 2]
0 1 2 3 4 5 6

max. profit we can generate only if buying at min price and selling at max. price.

~~Calculate max & min, ans = max - min~~

price : [3 5 2 1 4 5 2]
0 1 2 3 4 5 6

profit → 0 3 0 0 3 4 1

max = 4

max profit we can get if today is selling day.

```
int stockBuySell(int[] arr){
    maxprofit = 0;
    minprice = 0;
    for(int i=0; i<n; i++){
        minprice = Math.min(minprice, arr[i]);
        int profit = arr[i] - minprice;
        maxprofit = Math.max(maxprofit, profit);
    }
    return maxprofit;
}
```

T.C. O(n)
S.C. O(1)

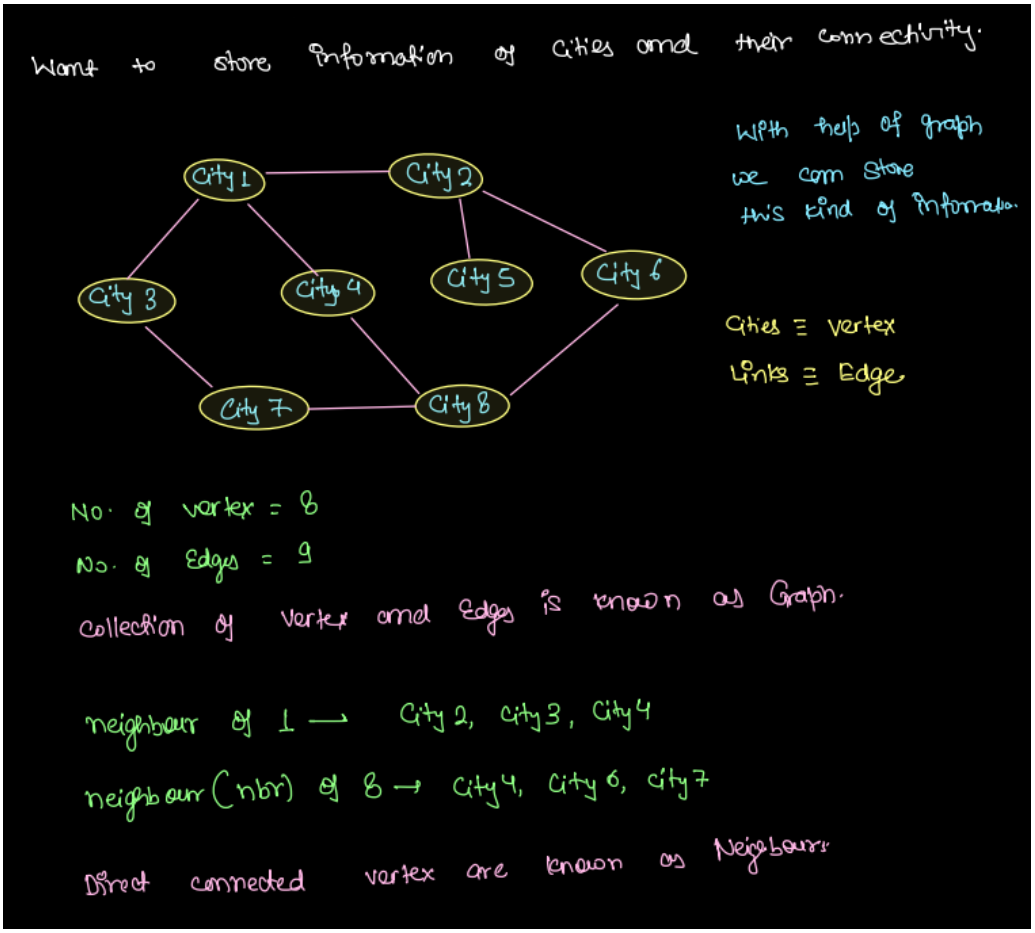
TODO:
Stock Buy Sell -2
Stock Buy Sell -3

DSA: Graphs 1 - 5 - Sep 2023

- **Introduction to graph**
- **Type of Graph**
- **How to store data in graph?**
- **Adjacency Matrix**
- **How to travel with Adjacency matrix implementation**
- **Implementation of directed weighted graph with adjacency matrix**
- **Major disadvantage of Adjacency Matrix**
- **Adjacency List Implementation**
- **How to implement weighted graph in adjacency list**
- **BFS Algorithm**
- **is Path available from source to destination**

Graph Introduction: A graph is a collection of vertices and edges.

- Vertex: In a graph, each individual point is referred to as a vertex.
- Edges: Edges are the links that connect these vertices.
- Neighbor: A vertex directly connected to another vertex is known as a neighbor.

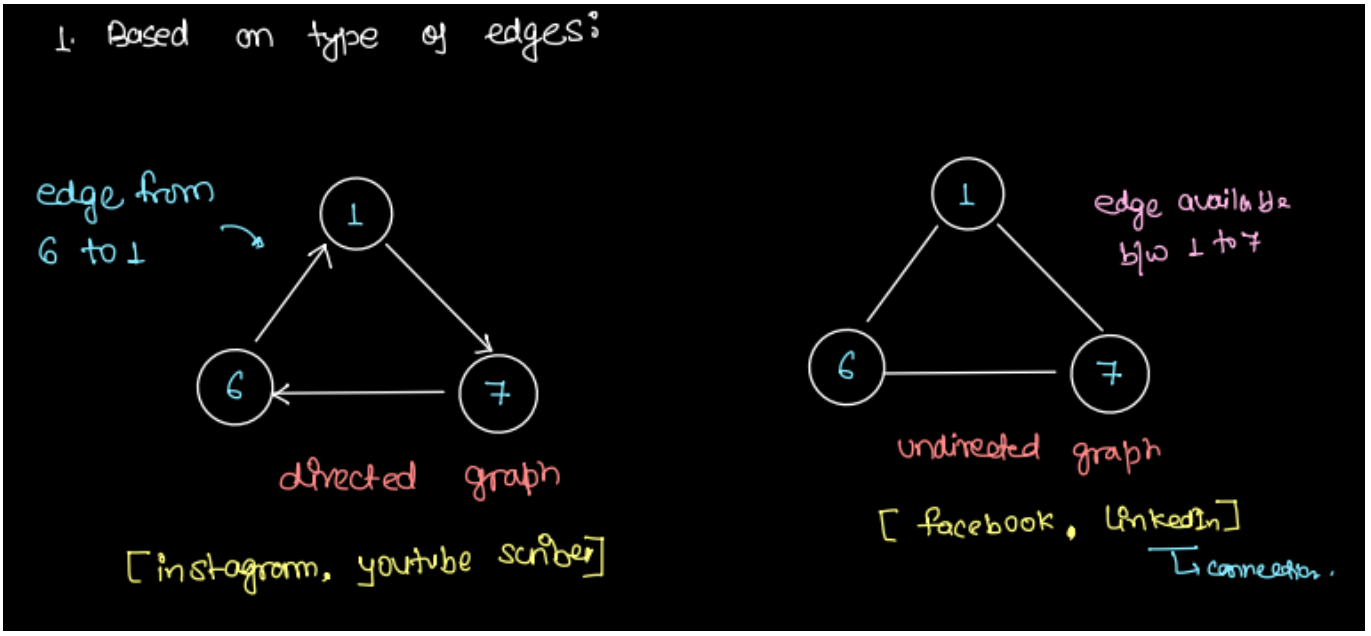


Type of Graph -> Three are numerse type of graph Based on edges there are two graphs Directed and UnDirected graph

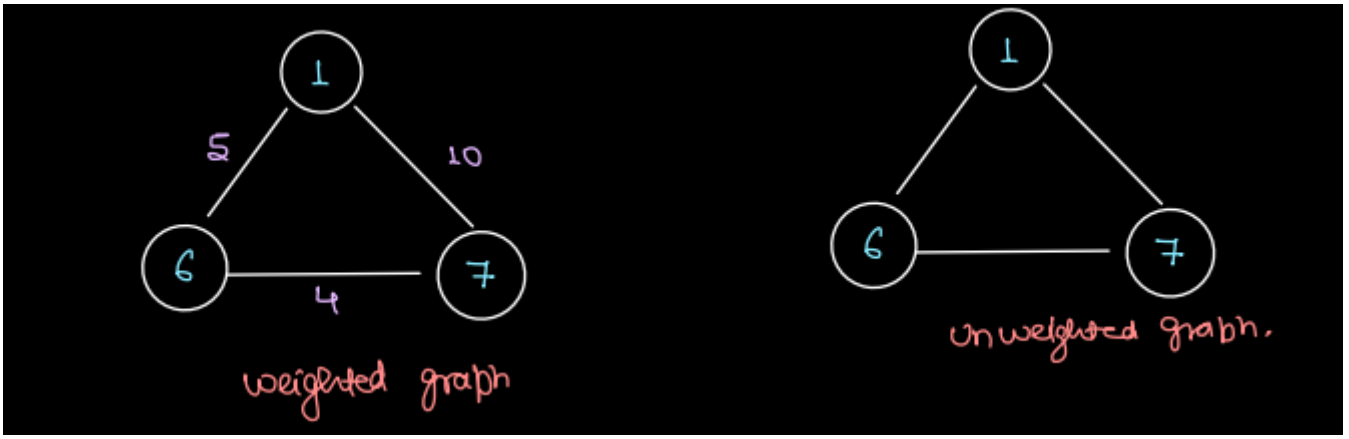
1. **Based in type of edges:**

Directed graph -> If arrows available on edge of graph it is called directed graph and arrow pointed towards vertex that vertex called neigbour. example Instagram, Youtube subscribe..

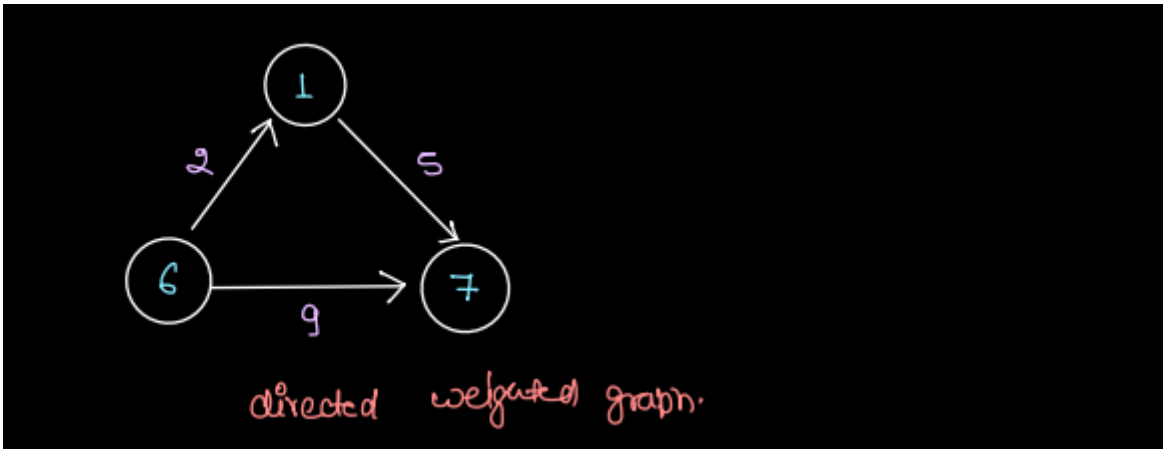
UnDirected graph -> If arrows not available on edge of graph it is called undirected graphs and all direct linked vertex called its never like before and after. example facebook, linkedin



2. Based on Edge wt. present or not:



3. Combination of above types are also possible.



How to store data in graph?

There is two famous implementation of graph is available

1. **Adjacency Matrix**
2. **Adjacency List**

Adjacency Matrix -> Any matrix can we connected with available vertex.
https://www.interviewbit.com/snippet/fe7af8388ec4667d9b0b/

Main Logic ->

1. Generate a 2D integer matrix called "graph" with dimensions vertex X vertex. Iterate through a loop until < edge.length.

2. Create two variables, u (source) and v (destination), and assign them the values from edge[i][0] and edge[i][1], respectively.

3. Use u and v as indices to update the graph by adding an edge from source to destination and another edge from destination to source. Set graph[u][v] and graph[v][u] to 1 for both of these edges. Finally, print the resulting graph.

1. Adjacency Matrix:

Vertex = 7 , Edges = 8

```
int[][] graph = new int[vtx][vtx];
```

	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	0	0	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	1	0	0
4	0	0	0	1	0	1	1
5	0	0	0	0	1	0	1
6	0	0	0	0	1	1	0

Edge →

v_1	v_2	
0	3	✓
0	1	✓
2	3	✓
3	4	✓
1	2	✓
4	5	✓
4	6	✓
5	6	✓

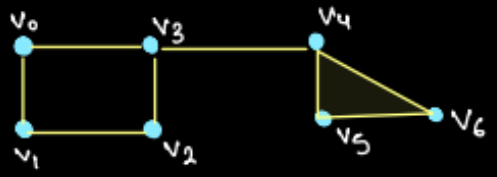
i →

```
int u = edge[i][0];
int v = edge[i][1];
// Edge b|w u & v
graph[u][v] = 1;
graph[v][u] = 1;
```

undirected graph

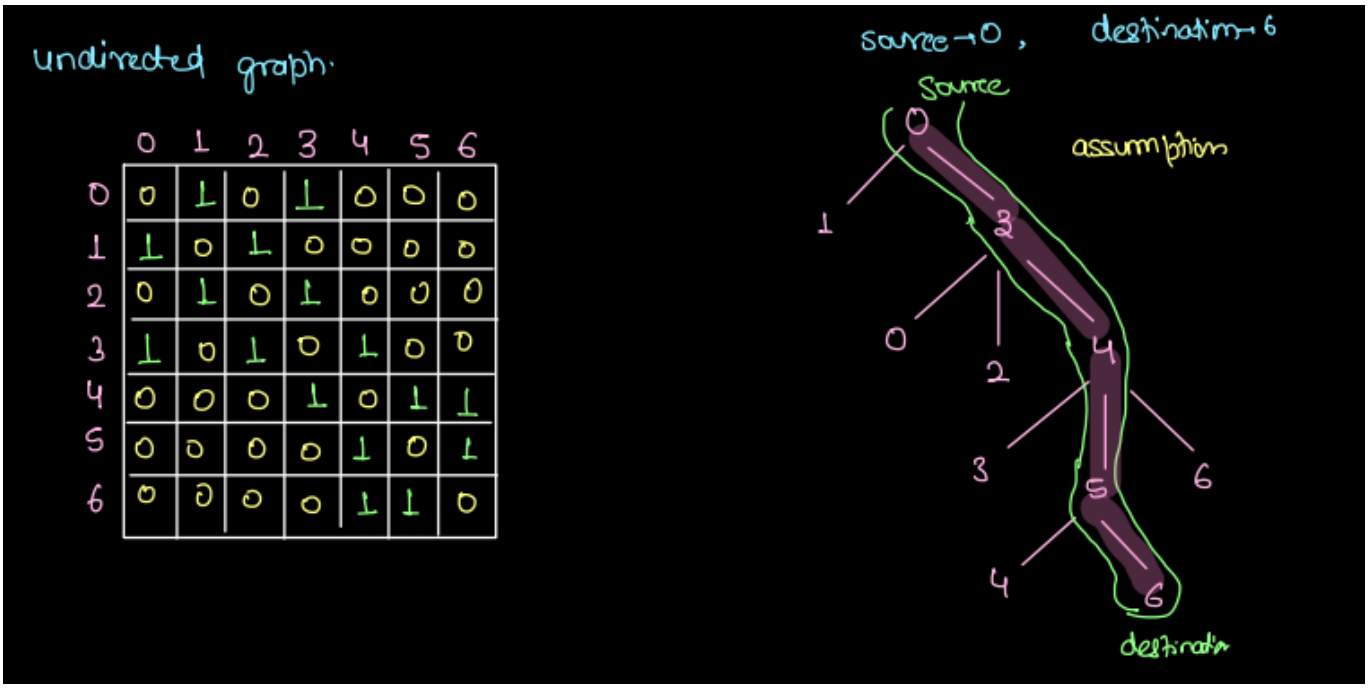
```
int u = edge[i][0];   u=2
int v = edge[i][1];   v=3
graph[u][v] = 1;      graph[2][3] = 1
graph[v][u] = 1;      graph[3][2] = 1
```

	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	0	0	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	1	0	0
4	0	0	0	1	0	1	1
5	0	0	0	0	1	0	1
6	0	0	0	0	1	1	0



unweighted undirected graph.

How to travel with Adjacency matrix implementation



implementation of directed weighted graph with adjacency matrix
https://www.interviewbit.com/snippet/d851c7c91ba58b02bbfc/

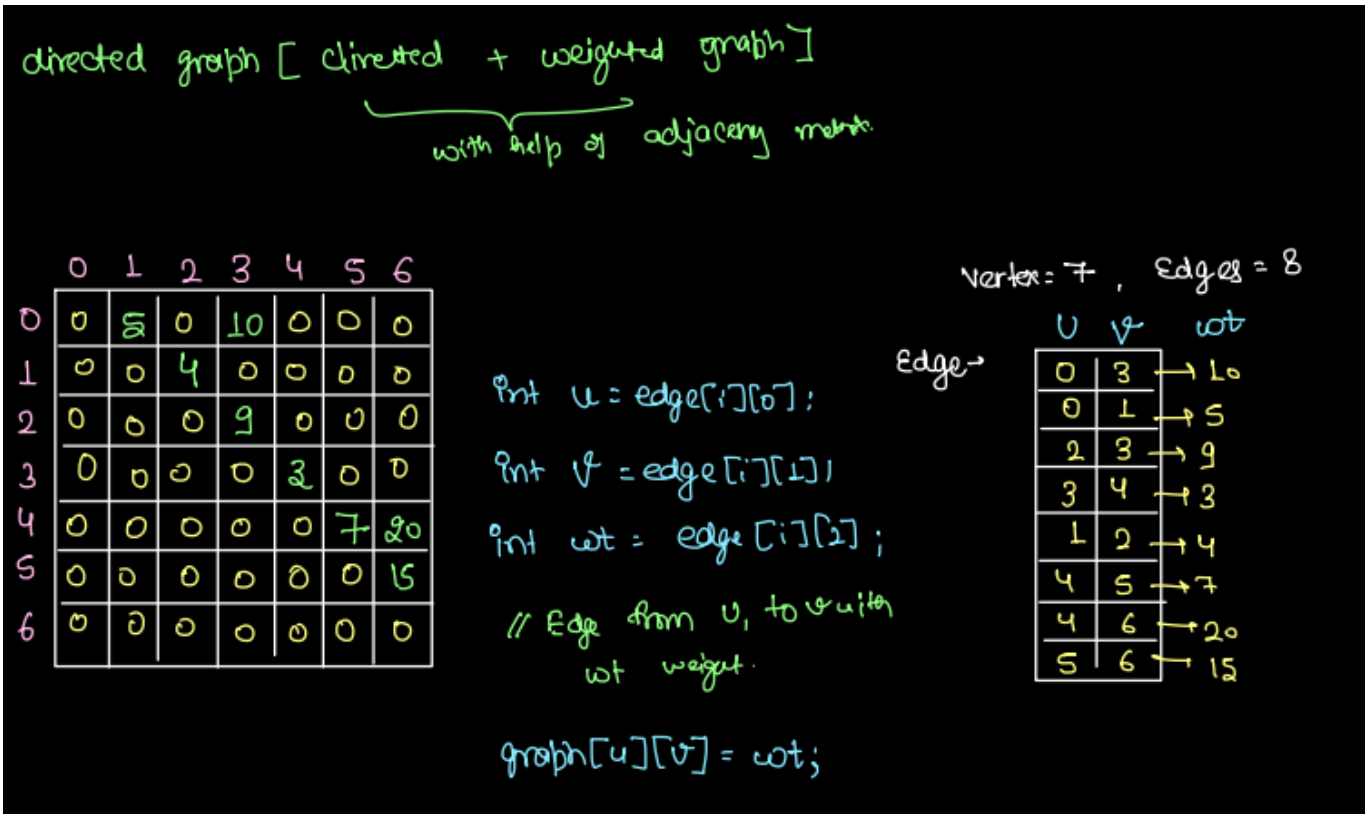
Main Logic ->

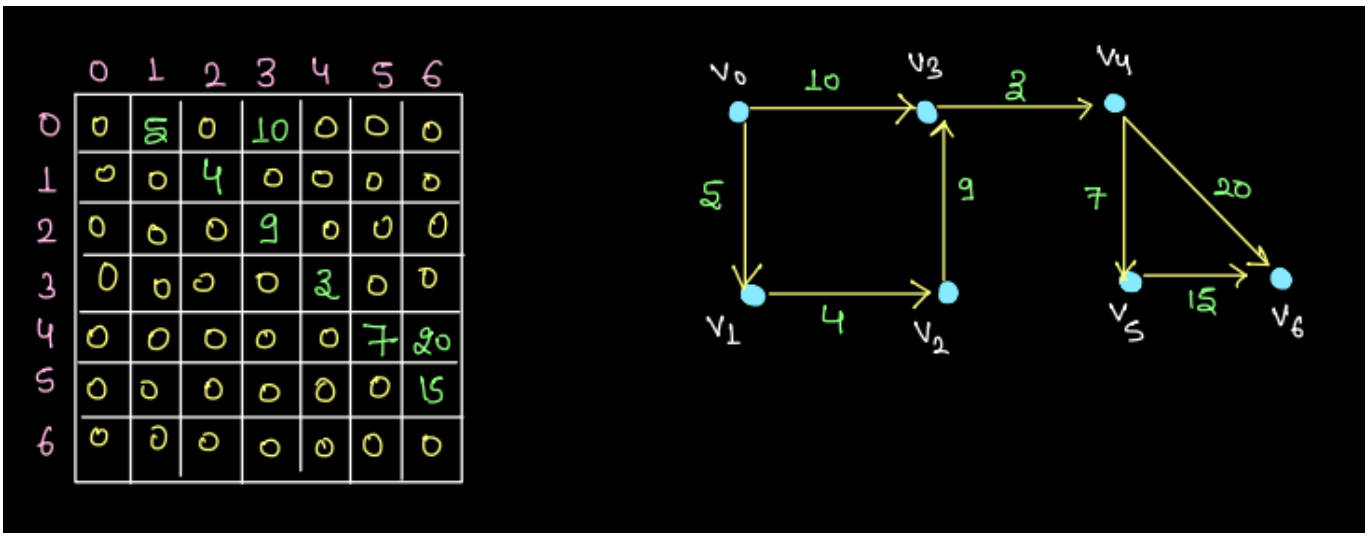
1. Define a class called "Pair" with three integer fields: "u" (source), “v” (destination) and "wt" (weight).
2. Generate a 2D integer matrix called "graph" with dimensions vertex X vertex. Iterate through a loop until < edge.length.

3. Create three variables, u (source), v (destination) and wt (weight), and assign them the values from edge[i][0], edge[i][1] and edge[i][2], respectively.

4. Use u and v as indices to update the graph by adding an edge from source to destinatio. Set graph[u][v] and assign wt.

5. Finally, print the resulting graph.





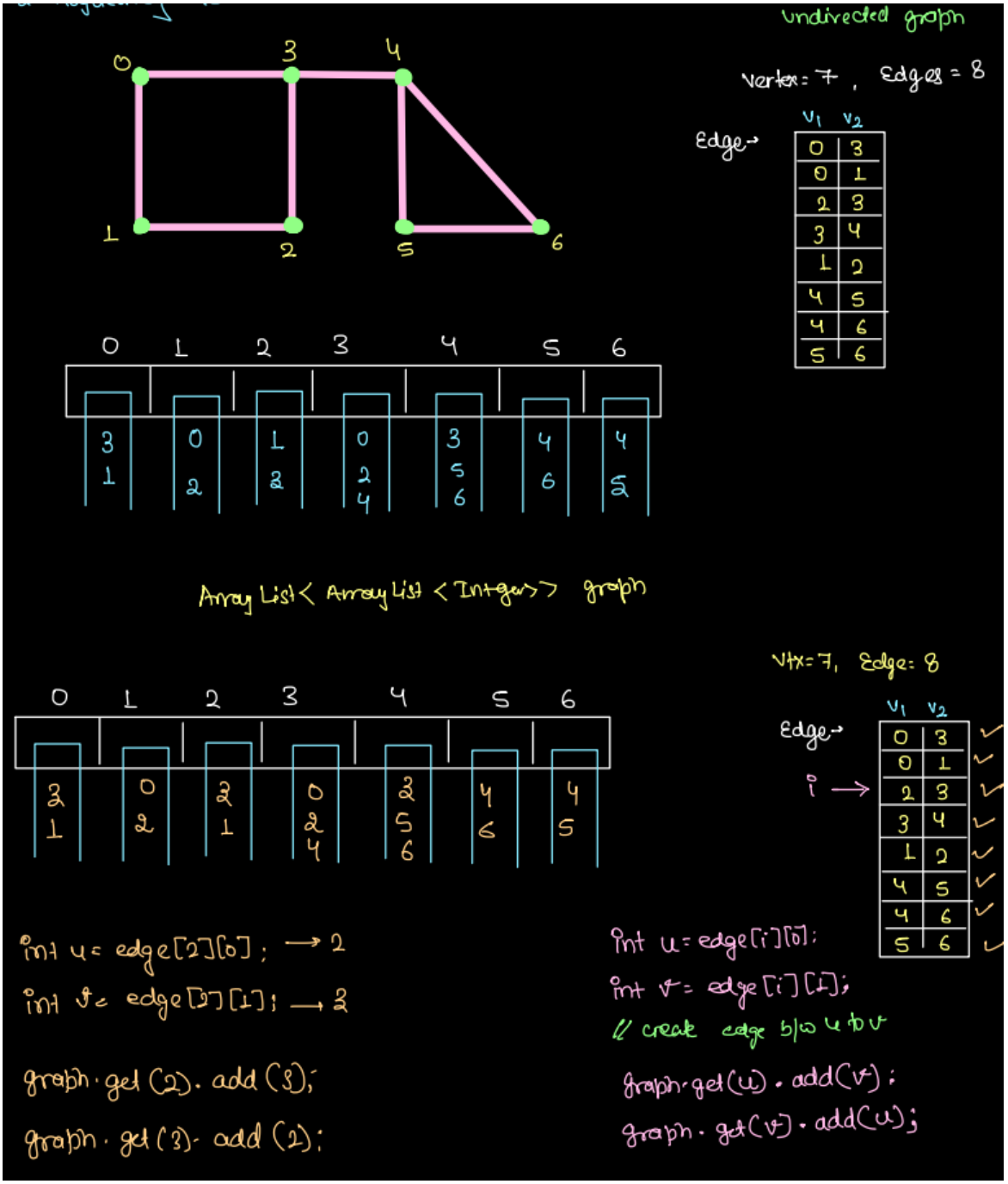
Major disadvantages of adjacency matrix: ->
Major disadvantage is space wastage, that is why most of the time, we will deal with adjacency list options

Adjacency List Implementation
https://www.interviewbit.com/snippet/26fbab66f7e354ec4a82/

Main Logic -> Create a Graph:

a. Initialize a graph as an ArrayList of ArrayLists of Integers.

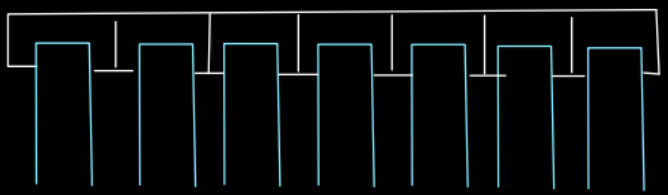
2. Initialize the Graph:
   a. Loop through the number of vertices (vtx) and add an empty ArrayList for each vertex in the graph.
3. Add Edges to the Graph:
   a. Loop through the edges (edge[i]) in the input array.
   b. Extract two integers, 'u' and 'v', from edge[i].
   c. Add 'v' to the list of neighbors for vertex 'u' in the graph and ‘u’ to ‘v’. and assign 1 to it. graph[u][v] = 1 and graph[v][u] = 1.



Vtx=7, Edge=8

```
ArrayList<ArrayList<Integer>> graph = new AL<>();

for(int v=0; v<vtx; v++) {
    graph.add(new ArrayList<>());
}
```

0	3
0	1
2	3
3	4
1	2
4	5
4	6
5	6



→ e

```
int u = edge[i][0];
int v = edge[i][1];
graph.get(u).add(v);
graph.get(v).add(u);
```

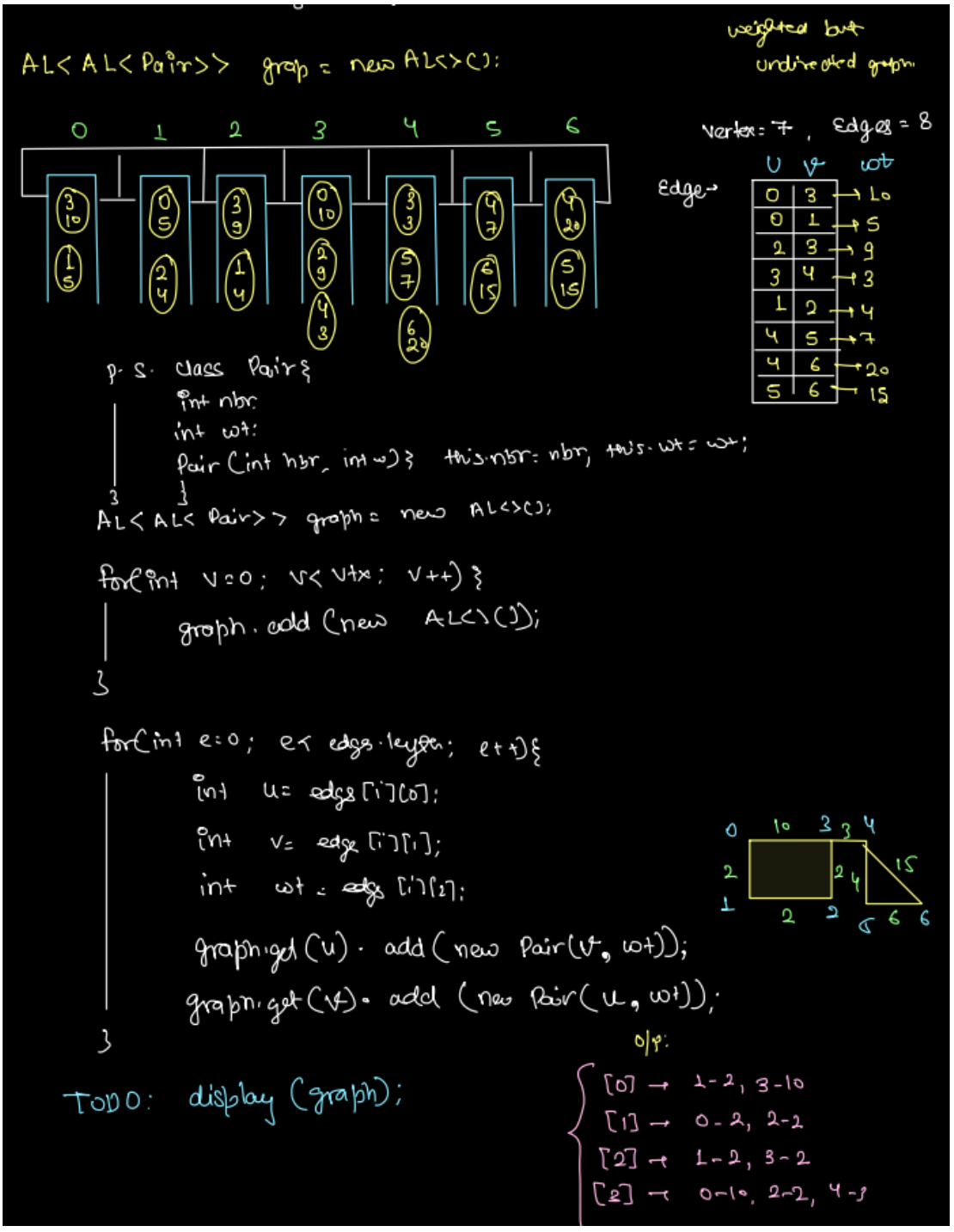
```java
import java.util.*;

class Main {


    public static void display(ArrayList<ArrayList<Integer>> graph) {
        for(int v = 0; v < graph.size(); v++) {
            // graph.get(v) -> vth index ArrayList
            System.out.print("[" + v + "] -> ");
            for(int nbr : graph.get(v)) {
                System.out.print(nbr + " ");
            }
            System.out.println();
        }
    }

    public static void main(String args[]) {
        int[][] edges = {
        {0, 3},
        {0, 1},
        {2, 3},
        {3, 4},
        {1, 2},
        {4, 5},
        {4, 6},
        {5, 6}
        };
        int vtx = 7;

        // create graph

        ArrayList<ArrayList<Integer>> graph = new ArrayList<>();
        for(int v = 0; v < vtx; v++) {
            graph.add(new ArrayList<>());
        }

        // container is ready now
        // add edges in the graph
        for(int e = 0; e < edges.length; e++) {
            int u = edges[e][0];
            int v = edges[e][1];

            graph.get(u).add(v);
            graph.get(v).add(u);
        }

        display(graph);

    }
}
```

graph:

```
[
0. 3, 1
1. 0, 2
2. 3, 1
3. 0, 2, 4
4. 3, 5, 6
5. 4, 6
6. 4, 5
]
```

How to implement weighted graph in adjacency list
https://www.interviewbit.com/snippet/8f196ec6d661adda42ea/

Main Logic ->
Step 1: Define a class called "Pair" with three integer fields: "u" (source), “v” (destination) and "wt" (weight).
Step 2: Create an undirected graph using the "Pair" type. To represent this graph, initialize an empty list called "graph," which will contain lists of pairs.
Step 3: Initialize the "graph" list with empty lists for each vertex. The number of vertices should be specified in problem.
Step 4: Populate the graph with edges using a 2D array called "edges." Iterate through each row of this array and extract three values: "u" (source), "v" (destination), and "wt" (weight). Add edges from "u" to "v" and from "v" to "u" in the "graph" list and assign weight to it.



BFS (Breadth first Search)
https://www.interviewbit.com/snippet/0968cdc31adbfd705c83/

Main Logic ->

Step 1: Building the Graph for BFS

1. Initialize a graph as an ArrayList of ArrayLists of Integers.
2. Initialize the Graph:
    a. Iterate through the number of vertices (vtx) and add an empty ArrayList for each vertex in the graph.
3. Add Edges to the Graph:
    a. Iterate through the edges (edge[i]) in the input array.
    b. Extract two integers, 'u' and 'v', from edge[i].
    c. Add 'v' to the list of neighbors for vertex 'u' in the graph and 'u' to 'v', assigning a weight of 1 to both edges. Set graph[u][v] = 1 and graph[v][u] = 1.

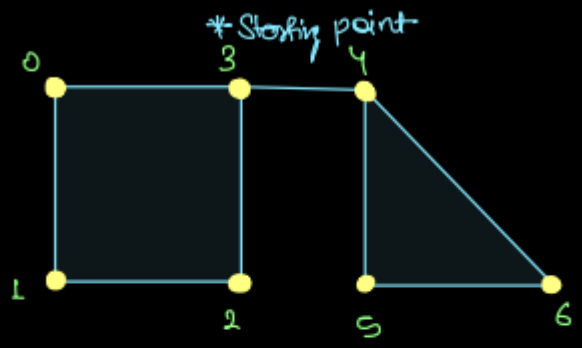
Step 2: Implementing the BFS Algorithm

1. Create a function called "BFS" with a return type of void, accepting an ArrayList of ArrayLists of Integer (graph) and an integer src.
2. Create a variable 'n' to store the length of the graph.
3. Create a queue of integers and add the source vertex (src) as the first element.
4. Create a boolean array 'vis' of length 'n' and set the element at the source index to true.
5. Run a loop while the queue is not empty (qu.size > 0).
    a. Remove an element from the queue and assign it to an integer variable 'rem'.
    b. Print the removed vertex.
    c. Check the condition: if (!vis[rem]), then do the following:
        i. Set vis[rem] to true.
        ii. Get the neighbors of the removed vertex from the graph using 'rem' as the index.
        iii. Iterate through the list of neighbors using a foreach loop with 'nbr' as the iterator variable.
        iv. Add 'nbr' to the queue.

At the end, call the BFS function and pass the graph and src as arguments.

exactly same as level order traversal of tree.

data structure use ⇒ queue



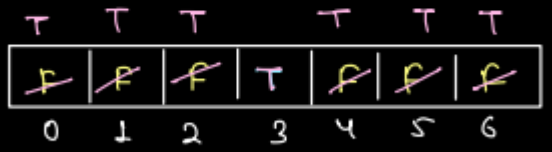
undirected graph

Steps of BFS

→ Remove
→ Print
→ add unvisited nbrs
  ↳ mark that nbr as visited one

queue

3, 0, 2, 4, 1, 5, 6

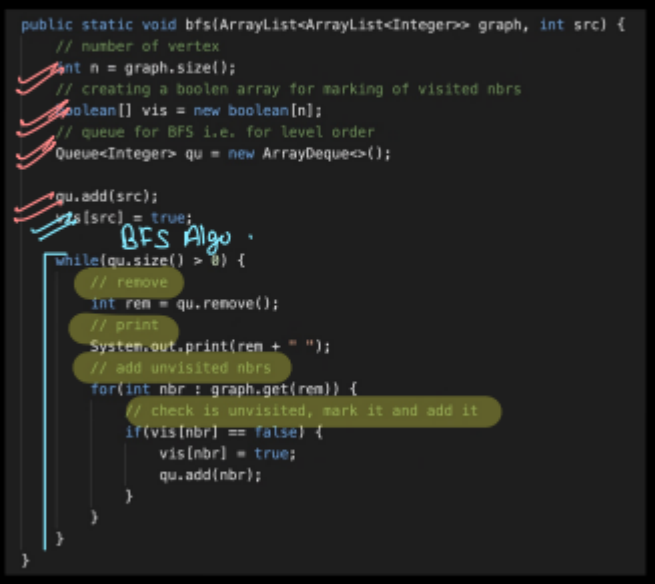


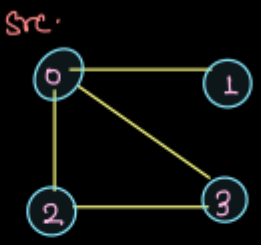
3 0 2 4 1 5 6

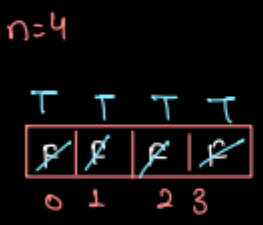
dist → 0 1 2

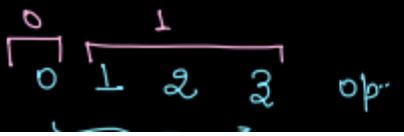
NOTE: Either starting pt. or source will given OR we can start from any point.

```java
public static void bfs(ArrayList<ArrayList<Integer>> graph, int src) {
    // number of vertex
    int n = graph.size();
    // creating a boolen array for marking of visited nbrs
    boolean[] vis = new boolean[n];
    // queue for BFS i.e. for level order
    Queue<Integer> qu = new ArrayDeque<>();

    qu.add(src);
    vis[src] = true;

    while(qu.size() > 0) {
        // remove
        int rem = qu.remove();
        // print
        System.out.print(rem + " ");
        // add unvisited nbrs
        for(int nbr : graph.get(rem)) {
            // check is unvisited, mark it and add it
            if(vis[nbr] == false) {
                vis[nbr] = true;
                qu.add(nbr);
            }
        }
    }
}
```



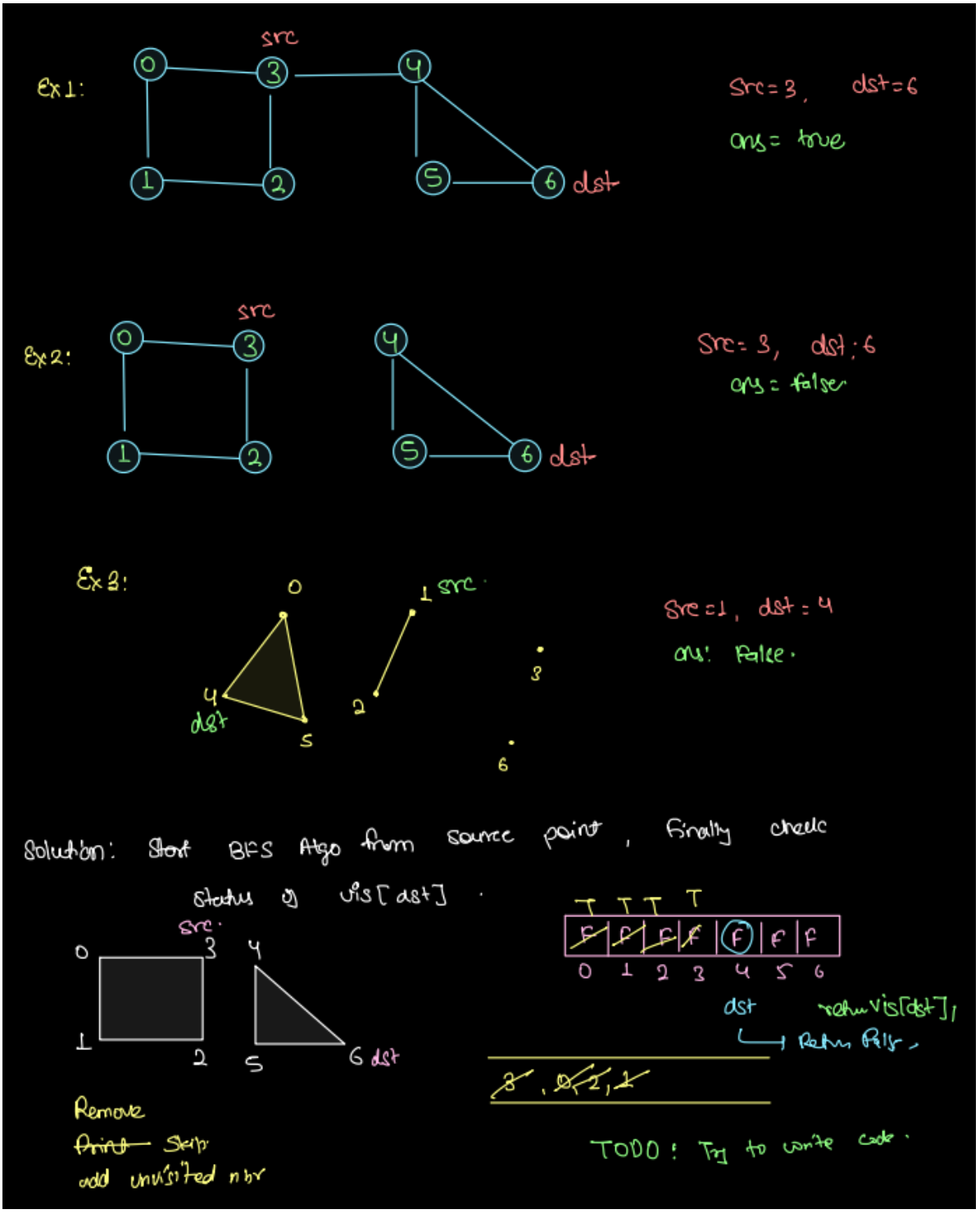


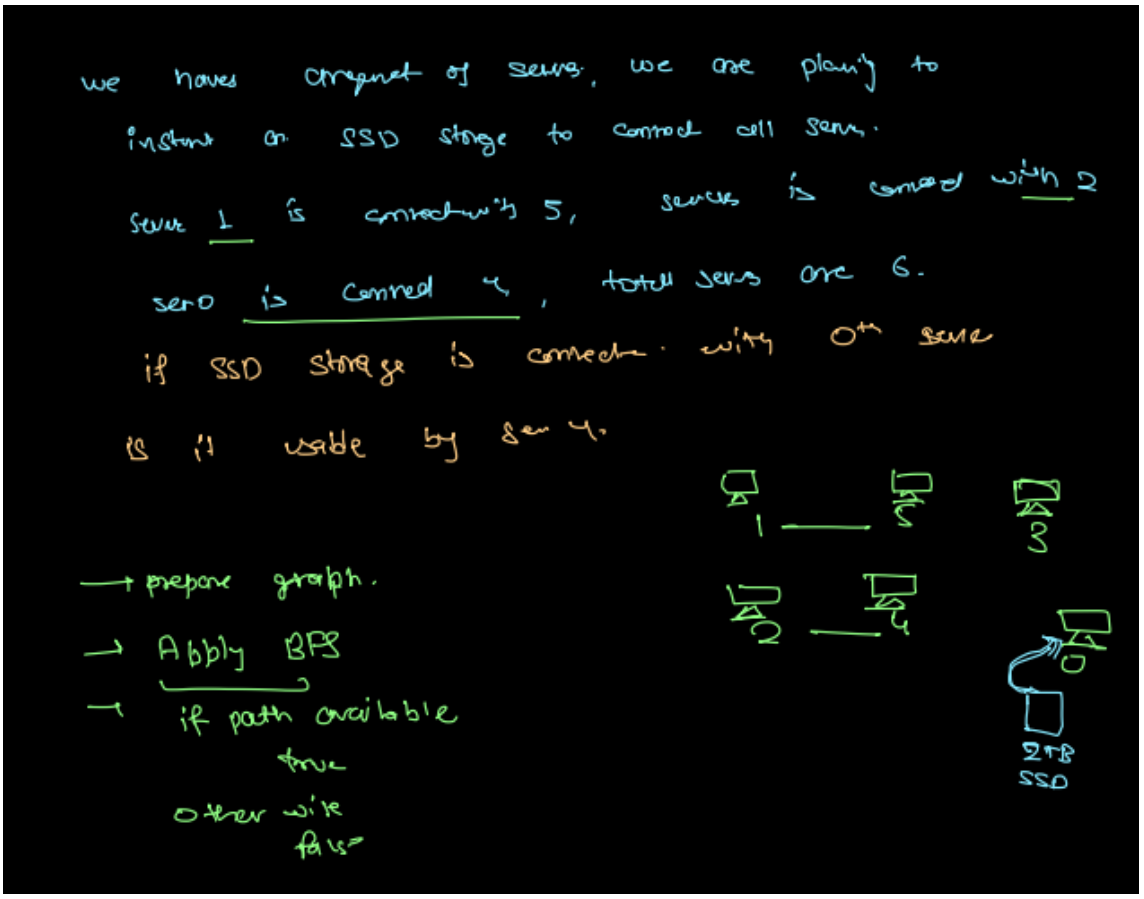




Is Path available from source to destination
Given on undirected graph, source node and destination node.
Check if there is a path available from source to destination or not.

https://www.interviewbit.com/snippet/e90842f34e2eb8af58a4/
Main Logic ->
use Same logic as BFS algo just return visited[destination] and change bfs function return type void to int.;



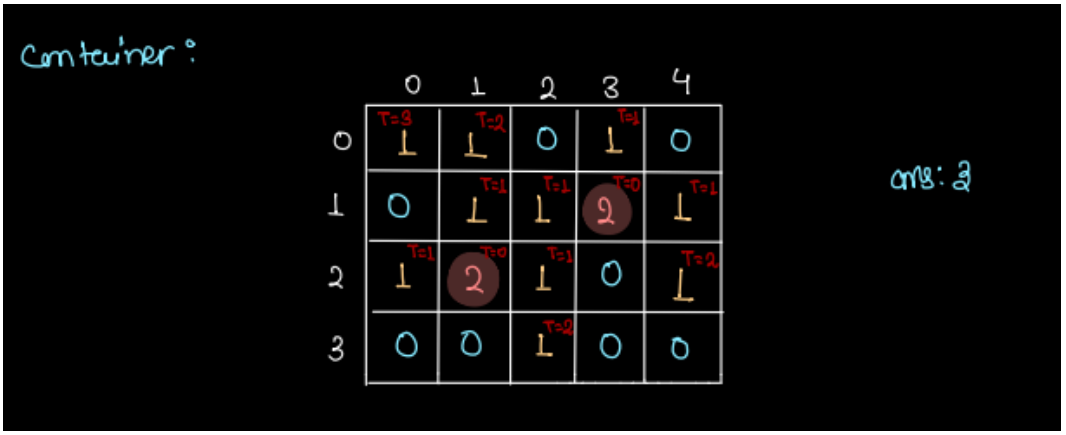


DSA: Graphs 2 - 7 - Sep 2023

- **Rotten Oranges**
- **DFS(Depth First Search)**
- **Connected Components**

Problem: Rotten Oranges:
Given mat[N][M], where any cell can have one of the value:
0 -> Empty cell
1 -> Fresh Orange
2 -> Rotten Orange
Every minute any fresh ornage adjacent (top, right, bottom, left) to rotten orange becomes rotten. Find min time when all oranges become rotten.

If not possible to rott every ornage, reutrn -1.



Main Logic:

To identify and process rotten oranges, follow the steps below:

1. Define a class named "Pair" with three integer fields: "i," "j," and "time."
2. Create a function called "rottenOranges" inside your program.
3. Initialize a queue called "qu" of type "Pair."
4. Create two variables, "n" (representing the length of the graph) and "m" (representing the length of the 0th index).
5. Initialize two integers, "totalOrange" and "time," both set to 0.
6. Add all the rotten oranges to the queue.
    - Iterate through the graph using variables "n" and "m."
    - Add the following condition: If the value of the graph at (i, j) is equal to 2, add it to the queue and increment "totalOrange." Otherwise, if the value at (i, j) is 1, increase "totalOrange."
7. Implement a Breadth-First Search (BFS) algorithm:
    a. Continue iterating as long as the queue "qu" has items.
    b. Remove an item from the queue and assign it to a variable called "rem," which has the type "Pair."
    c. Create two variables, "i" and "j," and one variable "t," initialized with the values from "rem.i," "rem.j," and "rem.time" respectively.
    d. Assign the value of "t" to "time" and decrement "totalOrange."
    e. Check the neighboring oranges:
        a. Create two arrays, "xAxis" and "yAxis," outside the while loop and fill them with values [0, -1, 0, 1] and [-1, 0, 1, 0] respectively.
        b. Run a loop for "i" from 0 to 3, and within the loop, create two variables, "r" and "c," initialized as "i + xAxis[i]" and "j + yAxis[i]."
        c. Add a condition: If "r" is greater than or equal to 0 and less than "n," and "c" is greater than or equal to 0 and less than "m," and the value of "graph[r][c]" is equal to 1, then add a new "Pair" with values (r, c, t + 1) to the queue, and set the value of "graph[r][c]" to 2 after the loop.
        d. Return the result: If "totalOrange" is not equal to 0, return -1; otherwise, return "time."

	0	1	2	3	4
0	1	1	0	1	0
1	0	1	1	2	1
2	1	2	1	0	1
3	0	0	0	1	0

t=0 → (1,3), (2,0)

t=1 → (0,3), (1,0) (1,2) (1,4) (2,0) (2,2)

t=2 → (0,1) (2,4)

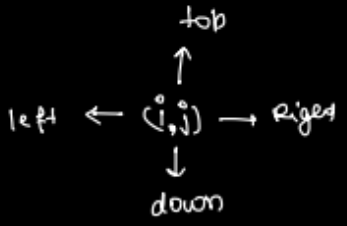
t=3 → (0,0)

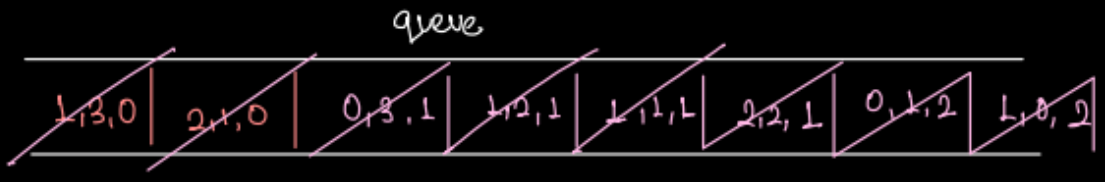
↳ left and still fresh.

ans → -1 : not possible to rott all oranges

	0	1	2	3
0	0	2	0	2
1	1	2	1	2
2	0	1	2	0

```
class Pair {
    int i;   ]
    int j;   ] coordinate
    int t;   ] time
}
```





1,3 → 0

2,1 → 0

0,3 → 1

1,2 → 1

1,1 → 1

2,2 → 1

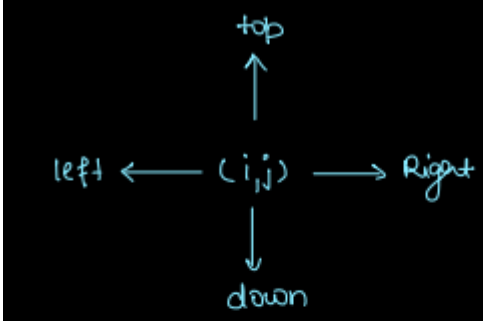
0,1 → 2

1,0 → 2

maximise answer of time.

① Remove
② work
③ Add unvisited nbr → mark them

max time Required is 2 secs.



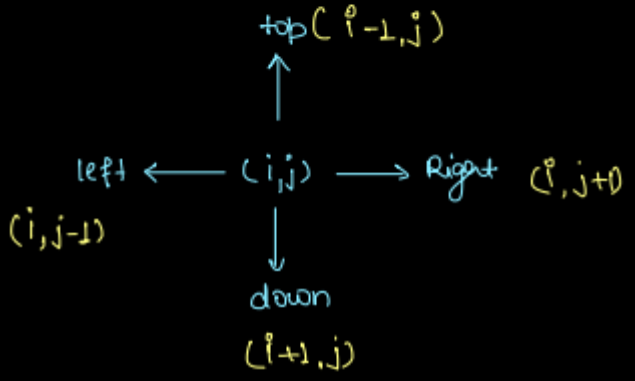
Brute force → direct method.

// top check

// Left check

// Down check

// Right check



i=2, j=3

```
for(int d=0; d<4; d++){
    int r= i+ xdir[d];
    int c= j+ ydir[d];
    SOP(r + ", " + c);
}
```

3

	T	L	D	R
xdir →	-1	0	1	0
ydir →	0	-1	0	1

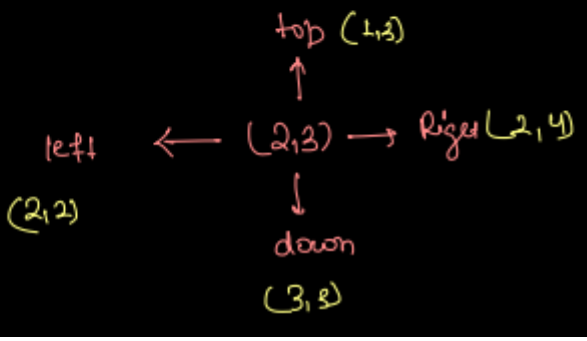
i=2
j=3

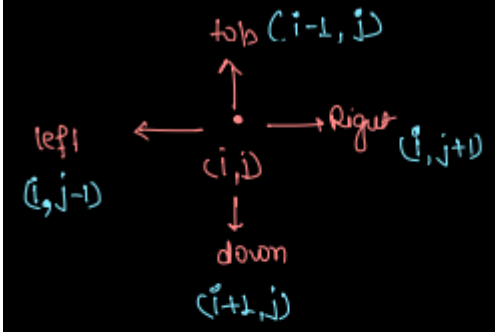
d=0 r= 2+(-1) =1 c=3+0 =3

d=1 r= 2+0 =2 c=3+(-1)=2

d=2 r= 2+1 =3 c=3+0 =3

d=3 r= 2+0 =2 c= 3+1 =4





	T	L	D	R
xdir →	-1	0	1	0
ydir →	0	-1	0	1

```java
import java.util.*;

class Main {

    /*
     * if mat[i][j] is :
     * 0 -> empty cell
     * 1 -> fresh Oranges
     * 2 -> Rotten Oranges Available
     *
     * if at any time T, (i,j) is rotted, at T+1, if rott adjacent element
     *
     * Adjacent Possibility for i,j:
     * top -> i-1, j
     * left -> i, j-1
     * down -> i+1, j
     * right -> i, j+1
     *
     * if it is not possible to rott all oranges, we have to return -1
     */
    public static class Pair {
        int i;
        int j;
        int time;

        public Pair(int i, int j, int time) {
            this.i = i;
            this.j = j;
            this.time = time;
        }
    }

                    // Top, left, down, right
    public static int[] xdir = {-1, 0, 1, 0};
    public static int[] ydir = {0, -1, 0, 1};

    public static int rottenOranges(int[][] mat) {
        int n = mat.length;
        int m = mat[0].length;
        // 1. make a queue of type Pairs
        Queue<Pair> qu = new ArrayDeque<>();

        // 2. Since multiple starting point possible,
        // add all of them in queue
        int totalOranges = 0;
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < m; j++) {
                if(mat[i][j] == 2) {
                    // rotted at time t = 0 -> add it in queue
                    qu.add(new Pair(i, j, 0));
                    totalOranges++;
                }
                else if(mat[i][j] == 1) {
                    // fresh orange
                    totalOranges++;
                }
```

Pair → helping us as vertex of graph

Pair

direction array which is helping to iterate in 4 different direction.

queue for BFS Algo.

Time Complexity of BFS Algorithm.

$= O(V+E)$

OR

E is $O(V^2)$

$= O(V^2)$

Depth First Search (DFS)

DFS is a graph traversal algorithm that allows us to systematically explore all the vertices and edges of a graph. It starts at a specific vertex (usually called the "starting vertex" or "source") and explores as deeply as possible along each branch before backtracking. DFS is used to visit every vertex in the graph.

Here's how DFS works in the context of a graph:

1. Start at a Vertex: Begin at a chosen starting vertex in the graph.
2. Mark as Visited: Mark the starting vertex as visited to keep track of it.
3. Explore Neighbors: Move to an unvisited neighbor of the current vertex. If there are multiple neighbors, pick one (the choice may vary depending on your implementation).
4. Repeat: Continue steps 2 and 3 recursively for the newly visited vertex.
5. Backtrack: If there are no unvisited neighbors from the current vertex, backtrack to the previous vertex (the one from which you came) and explore any remaining unvisited neighbors from there.
6. Repeat: Continue this process until all vertices have been visited or processed.
7. DFS can be implemented using either recursion or an explicit stack data structure. In the code example provided earlier, we used recursion to implement DFS. The recursion stack implicitly handles the backtracking.

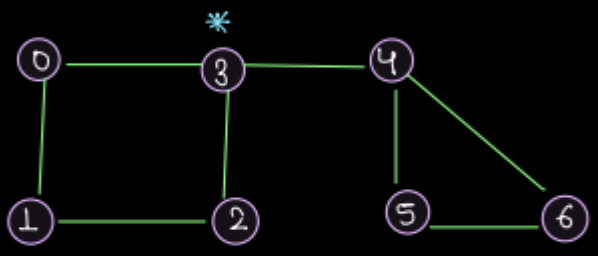
Use Cases of DFS:

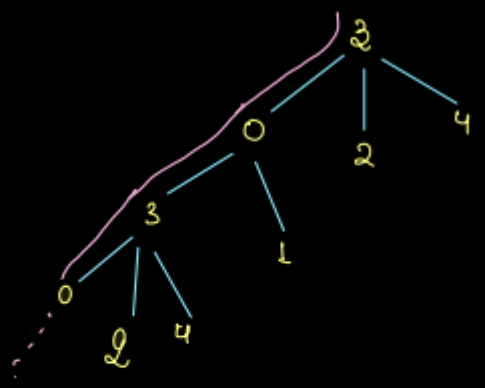
DFS has several practical applications in graph-related problems, including but not limited to:

- Connected Components: DFS can be used to find connected components in an undirected graph.
- Pathfinding: It can be used to find paths between two vertices in a graph.
- Topological Sorting: DFS can be used to perform topological sorting in directed acyclic graphs (DAGs).
- Cycle Detection: DFS can be used to detect cycles in a graph.
- Traversal: DFS can be used to traverse and explore a graph's structure.

DFS is a fundamental algorithm in graph theory and is widely used in computer science and various applications, including network routing, maze solving, and artificial intelligence.

Depth First Search :

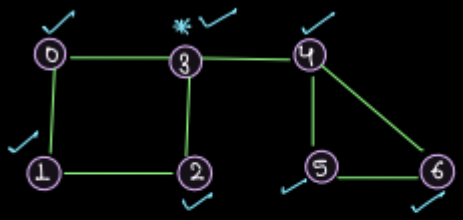


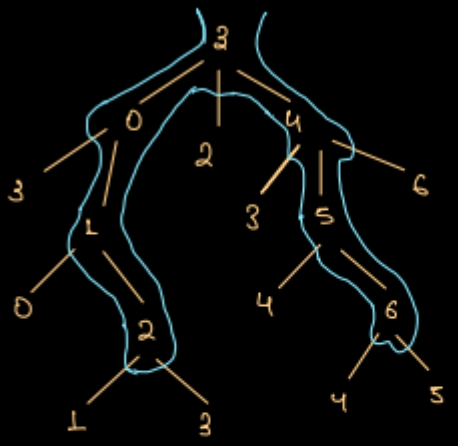


Infinite levels

→ Donot visit already visited vertex again.

Travel with marking;

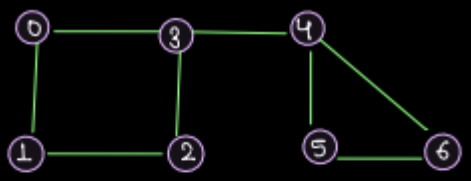




3 → 0 → 1 → 2

↘ 4 → 5 → 6

order 3 → 0 → 1 → 2 → 4 → 5 → 6 [Depth first search]

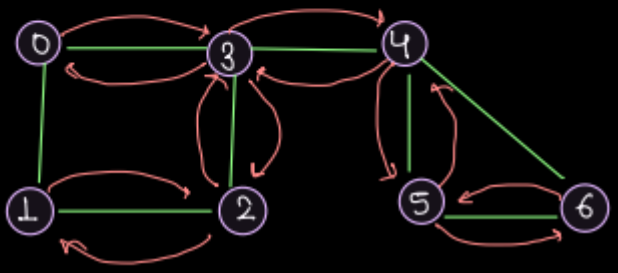



```java
// DFS
public static void DFS(ArrayList<ArrayList<Integer>> graph) {
    // graph vertex count
    int n = graph.size();
    // making a visited array to avaoid repeated calls
    boolean[] vis = new boolean[n];
    // making initial source as 0
    int src = 0;
    vis[src] = true;
    // call to DFS helper function
    dfsHelper(graph, src, vis);
}

public static void dfsHelper(ArrayList<ArrayList<Integer>> graph, int src,
                             boolean[] vis) {
    // 1. print the source point
    System.out.print(src + " ");
    // 2. move toward unvisited neighbour
    for(int nbr : graph.get(src)) {
        // unvisited nbr
        if(vis[nbr] == false) {
            // mark it and move toward it
            vis[nbr] = true;
            dfsHelper(graph, nbr, vis);
        }
    }
}
```

o/p stream:

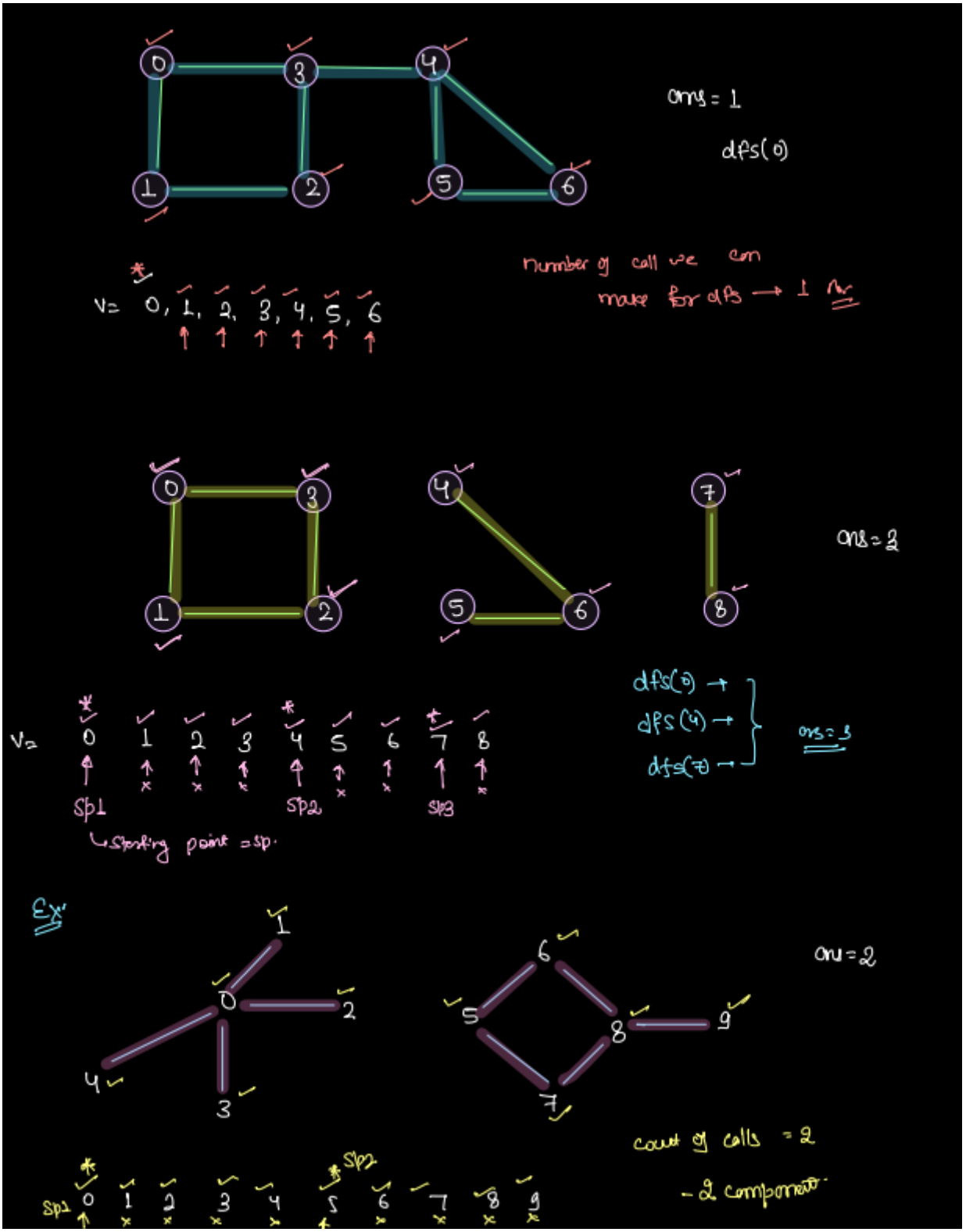
0 → 1 → 2 → 3
↓
6 ← 5 ← 4

o/p: 0 → 1 → 2 → 3 → 4 → 5 → 6



Connected Component

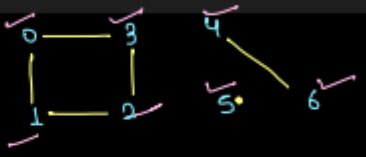
Given an undirected graph, find total number of connected component



```java
public static int solveConnectedComponents(ArrayList<ArrayList<Integer>> graph) {
    // number of vertex in graph
    int n = graph.size();
    // make a boolean visited array
    boolean[] vis = new boolean[n];
    int count = 0;
    for(int v = 0; v < n; v++) {
        if(vis[v] == false) {
            // mark it and increase count of component
            vis[v] = true;
            count++;
            // make call
            connectedComponentDFS(graph, v, vis);
        }
    }
    return count;
}
public static void connectedComponentDFS(ArrayList<ArrayList<Integer>> graph,
            int src, boolean[] vis) {
    for(int nbr : graph.get(src)) {
        // unvisited nbr
        if(vis[nbr] == false) {
            // mark it and move toward it
            vis[nbr] = true;
            connectedComponentDFS(graph, nbr, vis);
        }
    }
}
```

Dry Run:

```java
int[][] edges = {
    {0,3}, {0,1}, {1,2}, {2,3}, {4,6}
};
```




```java
public static int solveConnectedComponents(ArrayList<ArrayList<Integer>> graph) {
    // number of vertex in graph
    int n = graph.size();
    // make a boolean visited array
    boolean[] vis = new boolean[n];
    int count = 0;
    for(int v = 0; v < n; v++) {
        if(vis[v] == false) {
            // mark it and increase count of component
            vis[v] = true;
            count++;
            // make call
            connectedComponentDFS(graph, v, vis);
        }
    }
    return count;
}
public static void connectedComponentDFS(ArrayList<ArrayList<Integer>> graph,
            int src, boolean[] vis) {
    for(int nbr : graph.get(src)) {
        // unvisited nbr
        if(vis[nbr] == false) {
            // mark it and move toward it
            vis[nbr] = true;
            connectedComponentDFS(graph, nbr, vis);
        }
    }
}
```

n:7

T	T	T	T	T	T	T
0	1	2	3	4	5	6

count: 0 1 2 3 no. of components.

call

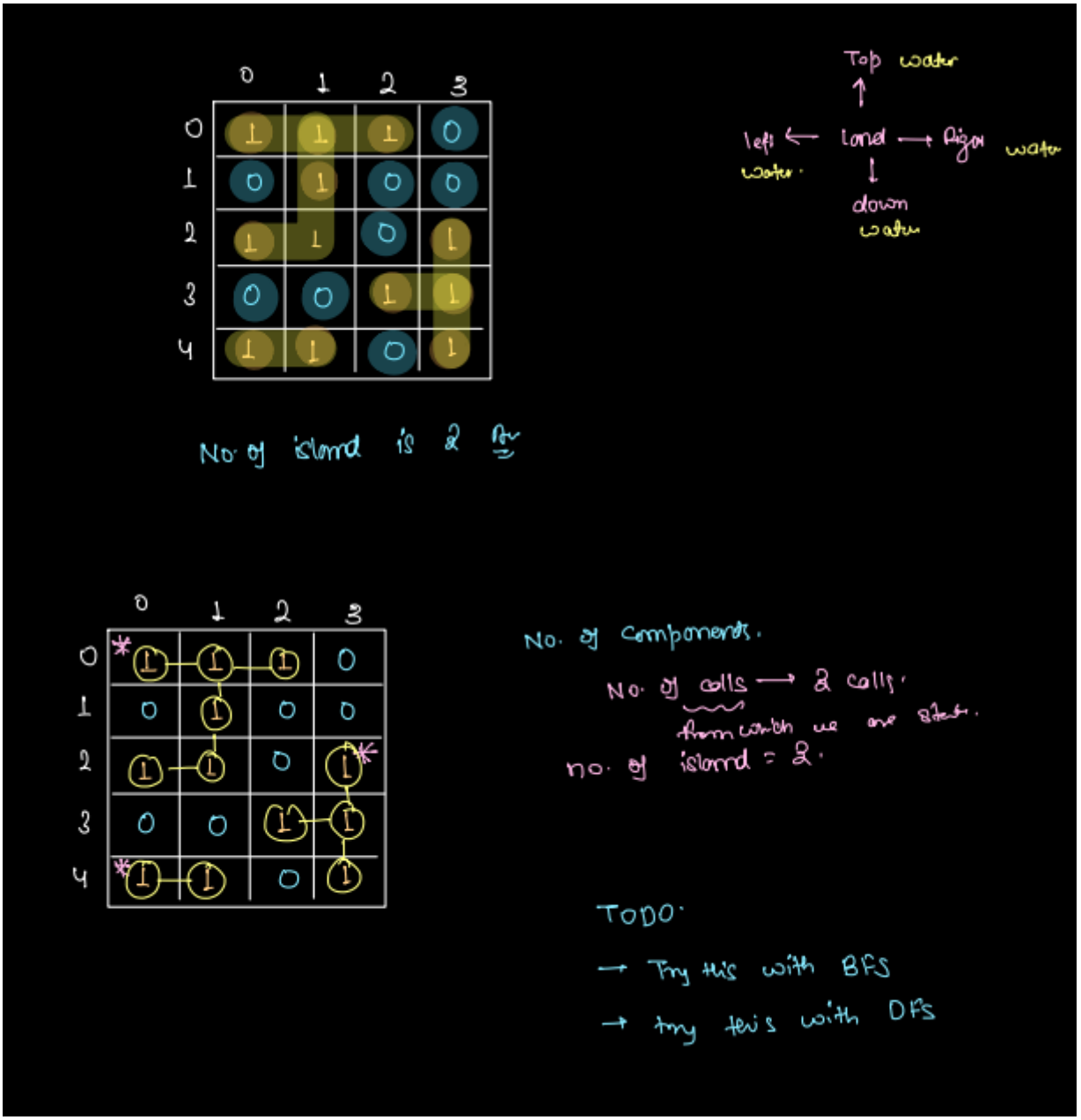
0 → ✓
1 → ×
2 → ×
3 → ×
4 → ✓
5 → ✓
6 → ×

No. of Islands

Given mat[N][M], where 0 represent water cell and 1 represent land cell.

Find total number of islands.

Note: An Islands can be formed by connecting adjcent land cells (T,L,D,R).



DSA: Graphs 3 - 9 - Sep 2023

- **Topological Sort**
- **Cycle in Directed Graph**
- **Dijkstra's Algorithm - Google map**

Topological Sort -> a topological order refers to a linear ordering of the vertices in a directed acyclic graph (DAG) such that for every directed edge (u, v), vertex u comes before vertex v in the ordering. In simpler terms, it's a way to arrange the vertices of a DAG such that there are no cycles, and the direction of edges is respected in the ordering.

Kahn's algorithm -> also known as Kahn's topological sorting algorithm, is a method used to find a topological order in a directed acyclic graph (DAG). The algorithm was developed by Arthur Kahn in the 1960s. Here's a simple explanation of how Kahn's algorithm works:

1. Initialize: Start by initializing two data structures:
    a. A queue to store vertices with no incoming edges.
    b. An array or list to store the topological order.
2. Find vertices with no incoming edges: Go through all the vertices in the graph and identify those with no incoming edges (in-degree is 0). These are the vertices that can be considered as the starting points in the topological order.
3. Process vertices: Take one of the vertices with no incoming edges and add it to the topological order. Then, remove this vertex from the graph by updating the in-degrees of its neighboring vertices (decrementing their in-degrees by 1).
4. Repeat: Continue this process by repeatedly finding vertices with no incoming edges, adding them to the topological order, and updating the in-degrees of their neighbors until all vertices are processed.
5. Check for cycles: If, at any point during the algorithm, there are no vertices with in-degrees of 0 left in the graph, and there are still vertices remaining, it means the graph has a cycle (it's not a DAG), and a topological order cannot be found.
6. Result: Once all vertices are processed and added to the topological order, the list or array you've been building will contain the topological ordering of the vertices.

Indegree Array -> Indegree menas how many number of vertex is pointing towards U vtx

Directed Acyclic Graph (DAG) -> A Directed Acyclic Graph (DAG) is a specific type of graph data structure that consists of a set of vertices (nodes) and a set of directed edges (arcs) connecting these vertices. In a DAG:

Acyclic: The term "acyclic" means that there are no cycles in the graph. A cycle is a sequence of edges that starts and ends at the same vertex, allowing you to visit the same vertex multiple times by following a closed path.
DAGs are commonly used in various domains and applications due to their specific properties: Dependency Management, Scheduling, Compiler Optimization, Network Routing, Data Flow Analysis

Problem 1 ->
Topological Sort
Linear Ordering of graph such that for every directed edge u to v, vertex u comes before v in order (Applicable only for Directed Acyclic Graph)
https://www.interviewbit.com/snippet/6ce93685ac6e9d1cdca9/

```
int[][] edges = {{0, 3, 30},{0, 1, 10},{2, 3, 5},{3, 4, 4},
                 {1, 2, 10},{4, 5, 7},{4, 6, 12},{5, 6, 3}};
```

int vtx = 6;

Main logic ->
We will use Kahn's Algorithm using following steps
Steps:

4. Create a Graph:
    a. Initialize a graph as an ArrayList of ArrayLists of Integers.
5. Initialize the Graph:
    a. Loop through the number of vertices (vtx) and add an empty ArrayList for each vertex in the graph.
6. Add Edges to the Graph:
    a. Loop through the edges (edge[i]) in the input array.
    b. Extract two integers, 'u' and 'v', from edge[i].
    c. If the edges are 1-based indexed, subtract 1 from 'u' and 'v'.
    d. Add 'v' to the list of neighbors for vertex 'u' in the graph.
7. Topological Sort:
    a. Initialize an integer array 'inDegree' to store the in-degrees of vertices.
    b. Calculate the number of vertices (vtx) based on the graph size.
    c. Create an array 'inDegree' of size 'vtx' to store in-degrees.
    d. Iterate through the graph's adjacency lists to update in-degrees.
    e. Add vertices with in-degree 0 to a queue.
    f. Create a queue of integers (use PriorityQueue for lexicographic order if required).
8. Process Vertices:
    a. While the queue is not empty:
        i. Remove an element 'rem' from the queue.
        ii. If edges are 1-based indexed, add 1 to 'rem' for result matching.
        iii. Print or process 'rem'.
9. Returning the Result (Optional):
    a. If the question asks to return an int[] with the topological order, maintain an index variable (e.g., 'idx') outside the while loop.
    b. Add 'rem' to the result array at 'idx' and increment 'idx' for the next element.
10. Handling Edge Cases (Optional):
    a. If the graph size doesn't match the number of processed vertices ('idx'), return a new int[0] to indicate that no valid topological order exists.

u v

4→0 ✓

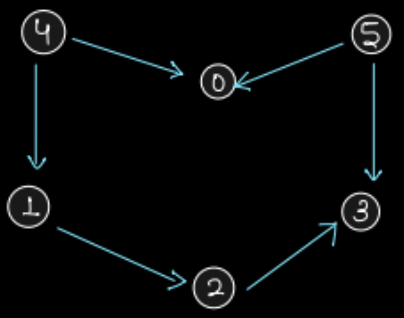
4→1 ✓

5→0 ✓

1→2 ✓

2→3 ✓

5→3 ✓



3 1 0 2 5 4 ✗

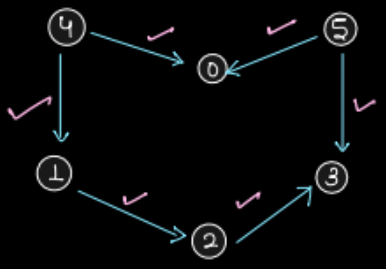
4 0 5 3 1 2 ✗

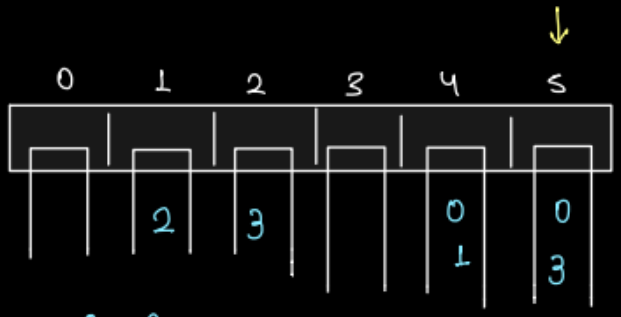
4 5 0 2 1 3 ✗

4 5 0 1 2 3 ✓

5 4 0 1 2 3 ✓

Kahn's Algorithm :





Steps in Queue

→ Remove front from que

→ Print Removed elent

→ Decrease indegree of nbrs of rem. vertex. If after decrement indgree become 0. add it in queue.

Steps:

1. Create indgree array.
2. add vertex with 0-indgree in queue.
3. Just travel and proces all the vertex

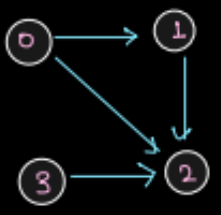
	0	1	2	3	4	5
indgree:	~~2~~	~~1~~	~~1~~	~~2~~	0	0
	~~1~~	0	0	~~1~~		
	0			0		

~~4~~ ~~5~~ ~~1~~ ~~0~~ ~~2~~ ~~3~~

queue

op: 4 5 1 0 2 3

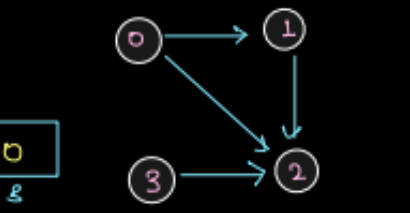
```
void topologicalSort(ArrayList<ArrayList<Integer>> graph) {
    // creating indegree array
    int vtx = graph.size();
    int[] indegree = new int[vtx];
    for(int v=0; v<vtx; v++) {
        for(int nbr: graph.get(v)) {
            indegree[nbr]++;
        }
    }
    // Create a queue and add all vertex
    where indegree is 0
    Queue<Integer> que = new ArrayDeque<>();
    for(int v=0; v<vtx; v++) {
        if(indegree[v] == 0) {
            que.add(v);
        }
    }
    //proces the que
    while(que.size() > 0) {
        //1. Remove from que,
        int rem = que.remove();
        //2. Print removed element
        System.out.(rem + " ");
        //3. Decrese indegree of nbr. of
           removed vertex, if indegree become
           0, then add it in que.
        for(int nbr: graph.get(rem)) {
            indegree[nbr]--;
            if(indegree[nbr] == 0) {
                que.add(nbr);
            }
        }
    }
```



vtx = 4

indegree

0	1	2	3
0	1	3	0



queue

0 | 3 | 1 | 2

topological sort

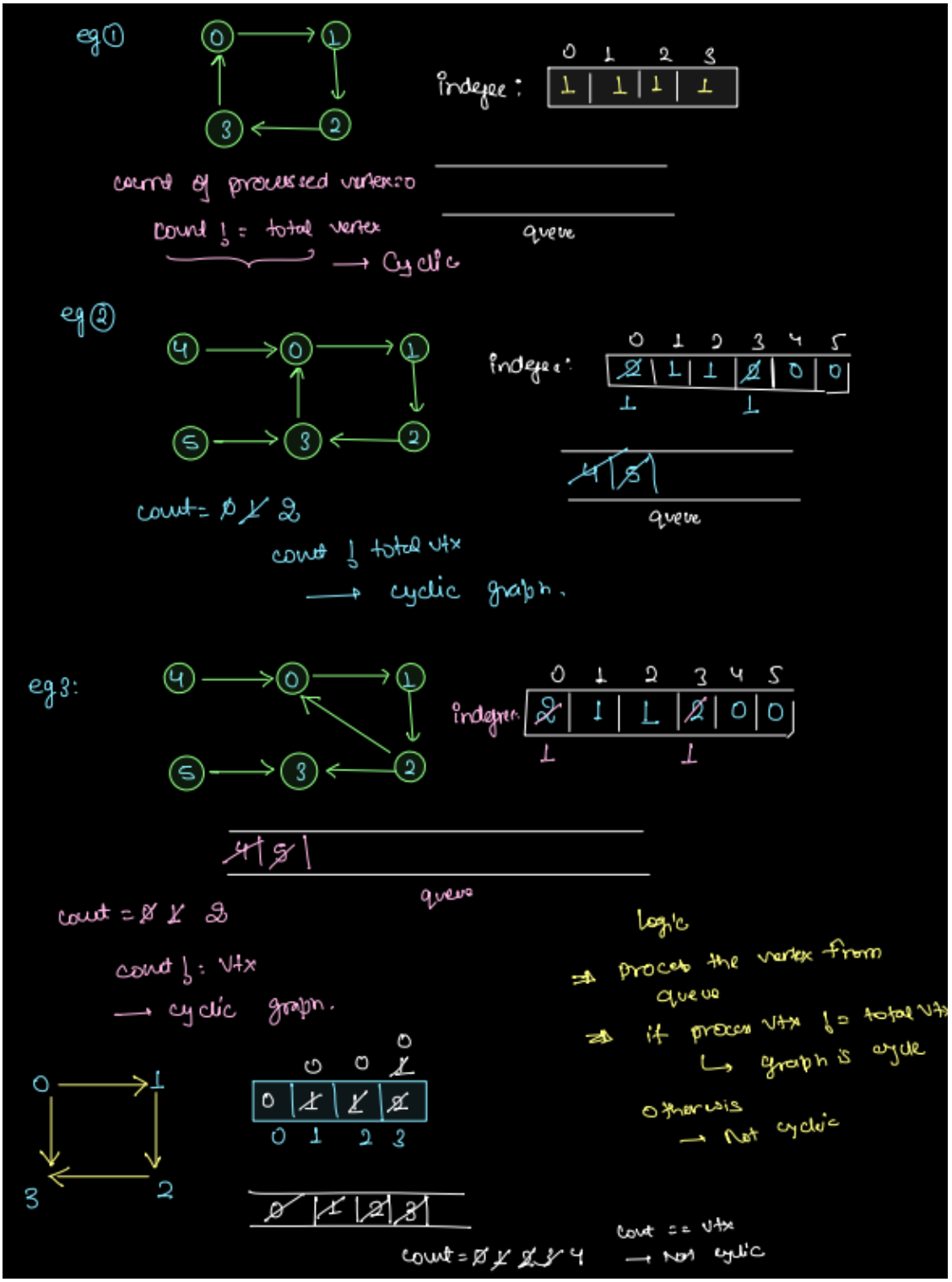
T.C : O(V+E)

S.C : O(V)

Problem 2 -> Cycle detection in directed graph
https://www.interviewbit.com/snippet/e66195593d003b08d82d/

Main Logic ->
Steps:

1. Follow the standard steps for Topological Sort.
2. Introduce a 'count' variable before entering the while loop.
3. Instead of printing or assigning data to an int[], increment the 'count' for each processed vertex.
4. Return 'count != vtx' to check if the topological order is valid (where 'vtx' represents the size of the graph).



```
boolean isCyclic(ArrayList<ArrayList<Integer>> graph){
    // creating indegree array
    int vtx = graph.size();
    int[] indegree = new int[vtx];
    for(int v=0; v<vtx; v++){
        for(int nbr: graph.get(v)){
            indegree[nbr]++;
        }
    }
    // Create a queue and add all vertex
    whose indegree is 0
    Queue<Integer> que = new ArrayDeque<>();
    for(int v=0; v<vtx; v++){
        if(indegree[v]==0){
            que.add(v);
        }
    }
    //process the que
    int count = 0;
    while(que.size()>0){
        //1. Remove from que,
        int rem = que.remove();
        //2. Increase count of processed elent
        count++;
        //3. Decrese indegree of nbr. of
        removed vertex, if indegree become
        0, then add it in que.
        for(int nbr: graph.get(rem)){
            indegree[nbr]--;
            if(indegree[nbr]==0){
                que.add(nbr);
            }
        }
    }
    if(count != vtx){
        return true;
    }else{ return false;
```

→ return !(count == vtx);
OR
return count != vtx

8:10 - 8:25
Break time

Single Source shotest path algo
https://www.interviewbit.com/snippet/4b7e5fea71adb5824fe7/

Dijkstra's Algorithm -> It is a widely-used algorithm for finding the shortest path between nodes in a weighted graph, particularly in graphs with non-negative edge weights. It was conceived by computer scientist Edsger W. Dijkstra in 1956. This algorithm is primarily used in scenarios such as network routing and navigation systems.

Note: Dijkstr's Algo is exactly same as BFS algorithm the only diffrence is we use priority queue instead of normal que.

Main logic ->
Step 1: Define a class called "Pair" with two integer fields: "nbr" (neighbor) and "wt" (weight).
Step 2: Create an undirected graph using the "Pair" type. To represent this graph, initialize an empty list called "graph," which will contain lists of pairs.
Step 3: Initialize the "graph" list with empty lists for each vertex. The number of vertices should be specified in advance.
Step 4: Populate the graph with edges using a 2D array called "edges." Iterate through each row of this array and extract three values: "u" (source), "v" (destination), and "wt" (weight). Add edges from "u" to "v" and from "v" to "u" in the "graph" list.
Step 5: Define another class called "DPair" with two fields: "vtx" (vertex) and "wsf" (weight so far).
Step 6: Implement a function named "shortestPath" that takes two parameters: the "graph" (a list of lists of pairs) and a "source" vertex.
Step 7: Inside the "shortestPath" function:

- Determine the number of vertices in the graph and create a priority queue (a data structure that automatically sorts elements) called "pq" to store "DPair" objects. Set up a custom comparator to sort "DPair" objects based on the "wsf" (weight so far).
- Create two arrays: "ans" to store the final shortest distances and "vis" to track visited vertices. Initialize "ans" with zeros and "vis" as an array of all "false" values.
- Initialize the priority queue "pq" with a "DPair" object representing the source vertex and a weight of 0.

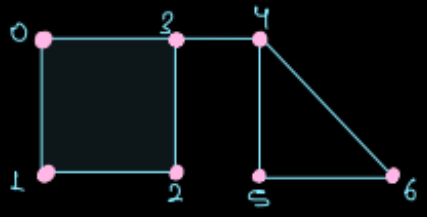
Step 8: Start a loop that continues as long as the priority queue "pq" is not empty:

- Remove the "DPair" with the minimum "wsf" from the queue and store it in a variable called "rem."
- Check if the vertex "rem.vtx" has already been visited (as indicated by "vis[rem.vtx] == true"). If it has, skip this iteration of the loop.
- If the vertex hasn't been visited, mark it as visited ("vis[rem.vtx] = true") and update the shortest distance to it in the "ans" array.
- Iterate through the neighbors of the current vertex "rem.vtx" by looping through the corresponding list in the "graph."
- For each neighbor "np, add a new "DPair" to the priority queue "pq" with the neighbor's vertex and an updated weight ("np.wt + rem.wsf").

Step 9: After the loop finishes, return the "ans" array, which contains the shortest distances from the source vertex to all other vertices.
Step 10: To find the shortest paths, call the "shortestPath" function and provide the "graph" and the "source" vertex as arguments. The function will return an array containing the shortest distances

-

* Single source shortest path (in terms of Edge)



source = 0 , dst = 6

All path:

0-1-2-3-4-5-6

0-1-2-3-4-6

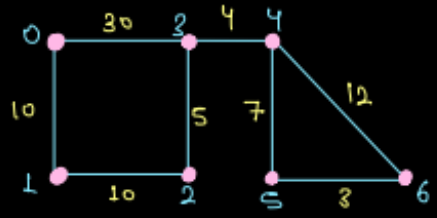
0-3-4-5-6

0-3-4-6

→ using BFS, we can calculate shortest path in terms of Edge

Dijkstra's Algorithm:

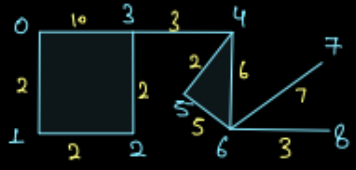
Single source, shortest path in terms of weighted.



NOTE: Dijkstra's Algo is ex
same as BFS algorith
the only difference is w
use priority Queue inste
of normal que

src = 0

0	10	20	25	29	36	39



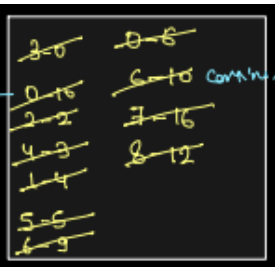
src = 3

answer array:

vis:

T	T	T	T	T	T	T	T	T
6	4	2	0	3	5	9	16	12
0	1	2	3	4	5	6	7	8

continue.



PQ<Pair> - min

Remove
vtx
wsf
add unvisited nbr.

class Pair {
    int vtx;
    int wsf;
}

[0] -> 3@30, 1@10, .

[1] -> 0@10, 2@10, .

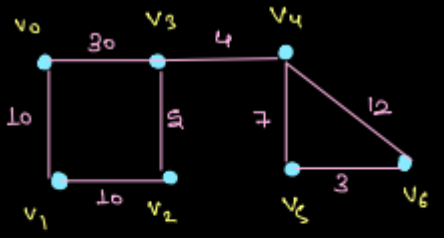
[2] -> 3@5, 1@10, .

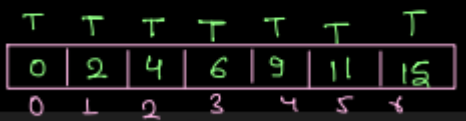
[3] -> 0@30, 2@5, 4@4, .

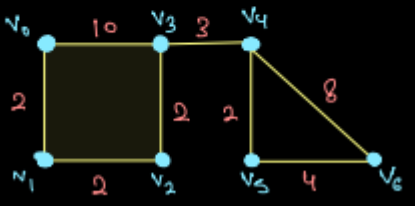
[4] -> 3@4, 5@7, 6@12, .

[5] -> 4@7, 6@3, .

[6] -> 4@12, 5@3, .








```java
while(pq.size() > 0) {
    // rem
    DPair p = pq.remove();

    // work -> mark visited and store answer
    if(vis[p.v] == true) {
        continue;
    }
    vis[p.v] = true;
    ans[p.v] = p.wsf;

    // add unvisited neighbour from p.v
    // np -> neighbour pair
    for(Pair np : graph.get(p.v)) {
        if(vis[np.nbr] == false) {
            pq.add(new DPair(np.nbr, np.wt + p.wsf));
        }
    }
}
```

Src., wsf

0, 0

3, 10 — continue.

1, 2

2, 4

3, 6

4, 9

5, 11

6, 17 — Contin.

6, 15

TODO: make directed Graph.

Apply Dijkstra's.

DRY Run.

60 days window

Short Realism

→ PSP ⇒ 80-90 Good Refund

→ M8E ⇒

Good candidate profile