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: Two Pointers - 6 - June 2023

Pair some
Pair difference
Pair Count
Container with most water

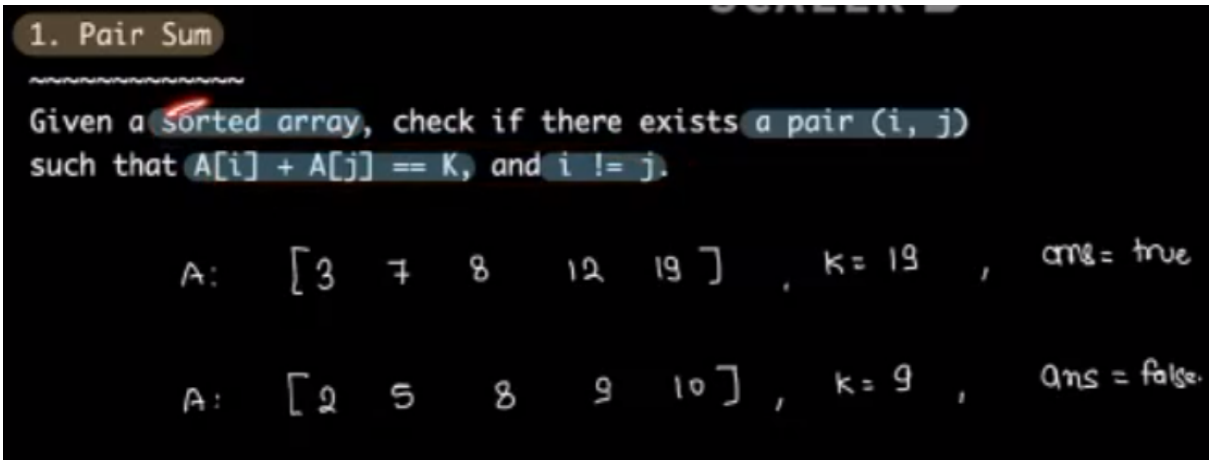
Given a sorted array, check if there exists a pair (i, j) such that A[i] + A[j] == k, and i != j.
A = [3,7,8,12,18] , k = 19
https://www.interviewbit.com/snippet/0bb1b08e93610f659671/

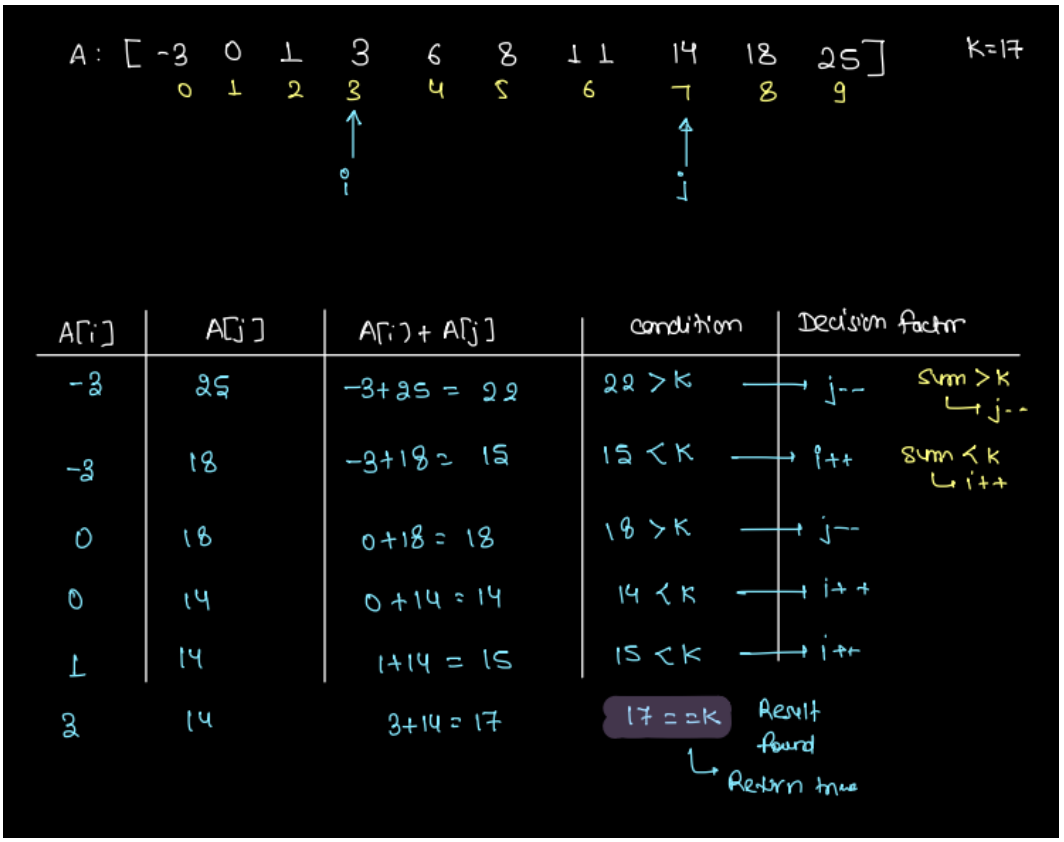
Main Logic ->

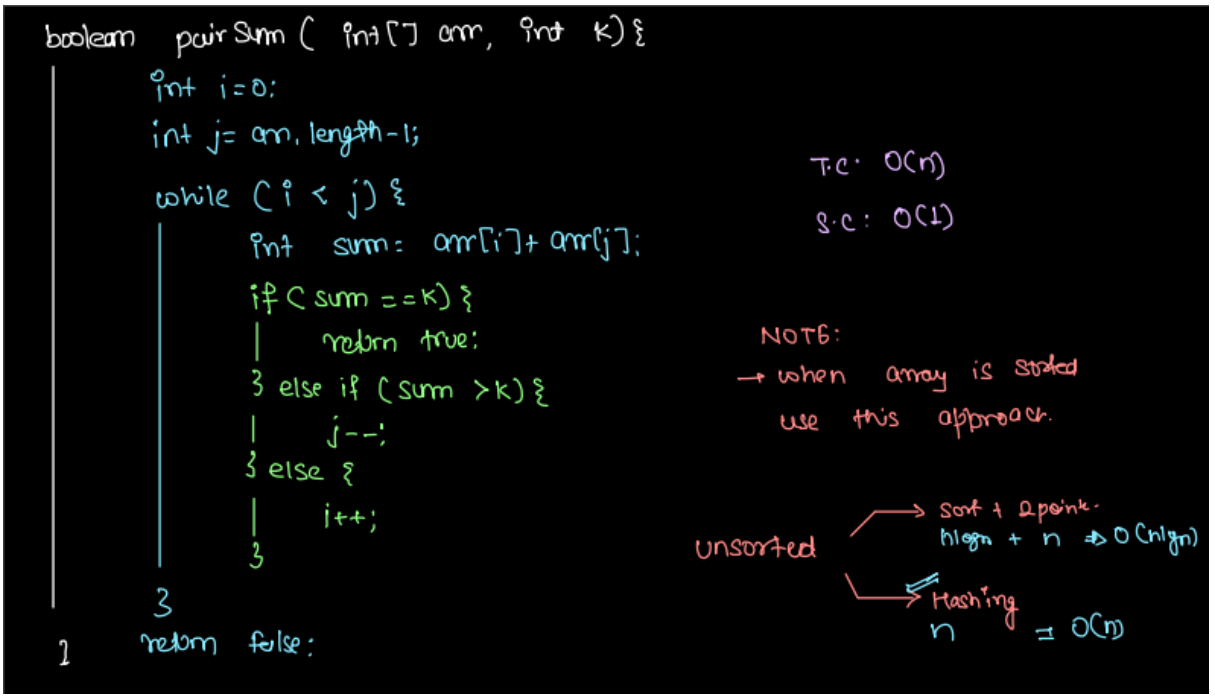
Initialization: Start with i **as the first index (0) and** j **as the last index (**array.length - 1**).**

1. Loop condition: Continue the loop as long as i is less than j.
2. Check the sum of elements: Calculate the sum of the elements at indices i and k. If the sum A[i] + A[k] is equal to k, return true as you have found a pair that satisfies the condition.
3. Update pointers: If the sum A[i] + A[k] is greater than k, decrement j by 1 (move the pointer towards the smaller value). Otherwise, decrement i by 1 (move the pointer towards the larger value).
4. Repeat steps 3-4 until i is no longer less than j.

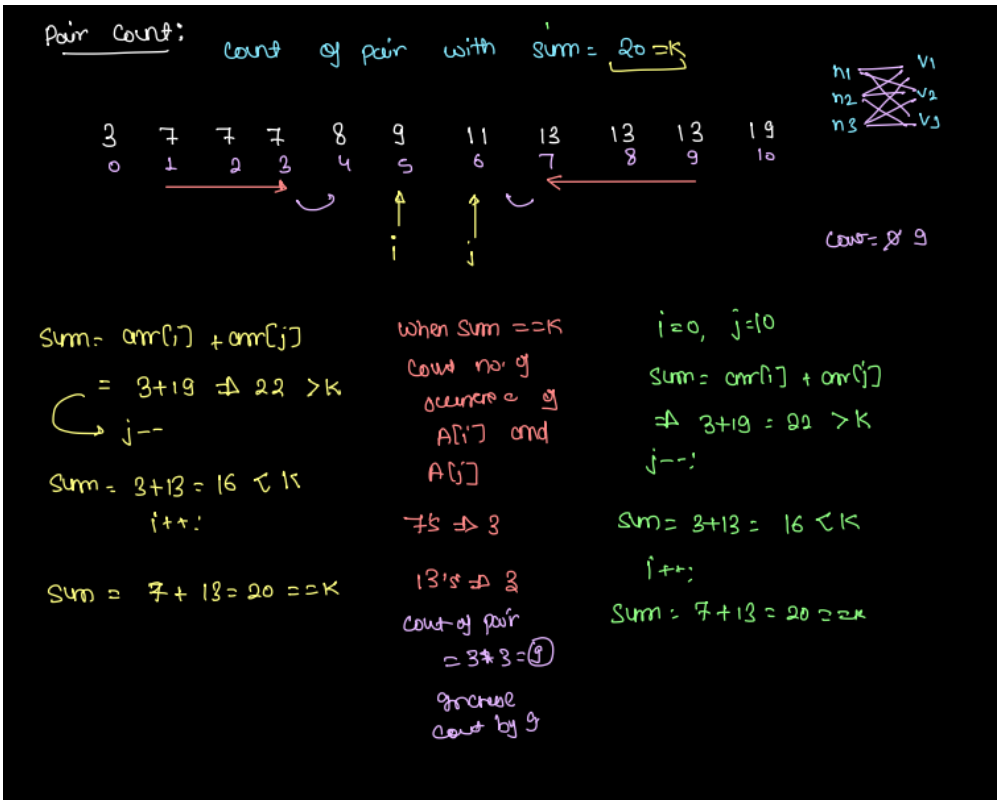
In summary, this code snippet compares the sum of elements at indices i and k with the target value k. If the sum is equal to k, it returns true. If the sum is greater than k, it decrements the j index; otherwise, it decrements the i index. The loop continues until i is no longer less than j.

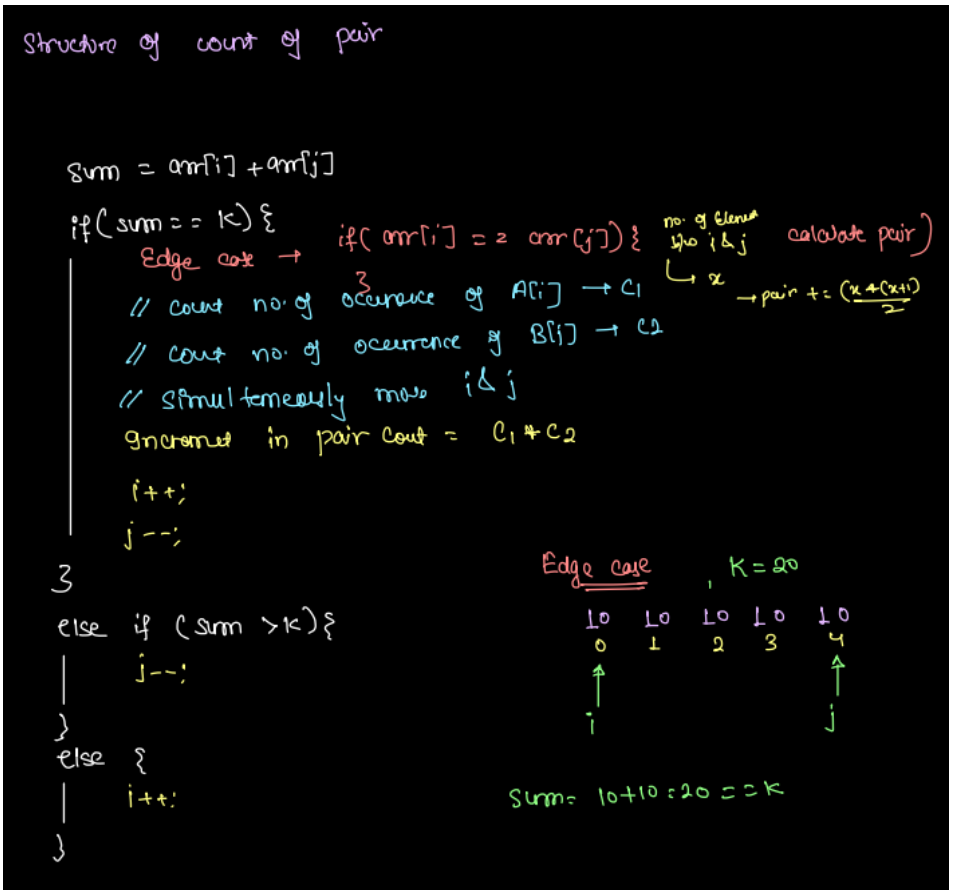






Pair of count —----------
https://www.interviewbit.com/snippet/9f78ff704fca643e1569/





Pair difference—----------

Given a sorted array, check if there exists a pair(i, j) such that A[j] - A[i] == k, k > 0, and i!=j.

A= [-3,0,1,3,6,8,11,14,18,25]. K = 5

https://www.interviewbit.com/snippet/1946b036c3851f401e8f/

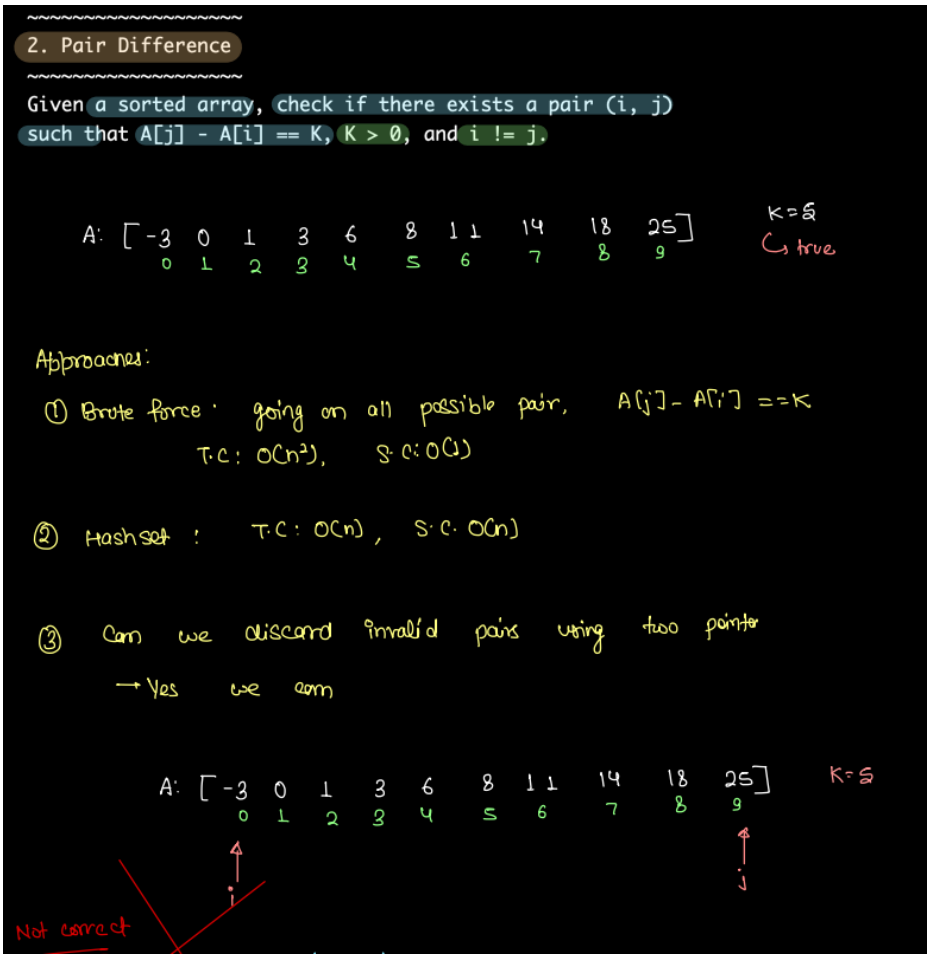
Main logic -

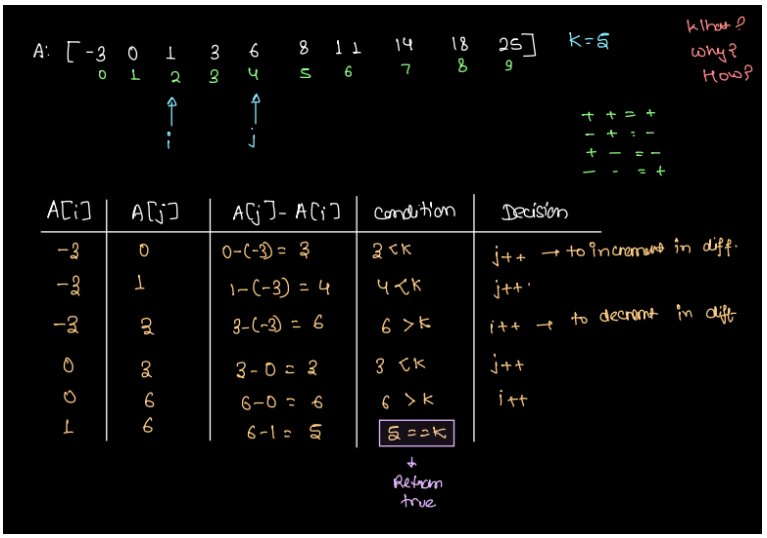
In this approach, we deviate from the previous logic where i started from 0 and j started from n-1 because the difference between the indices was decreasing. Instead, we will modify the logic to address this situation.

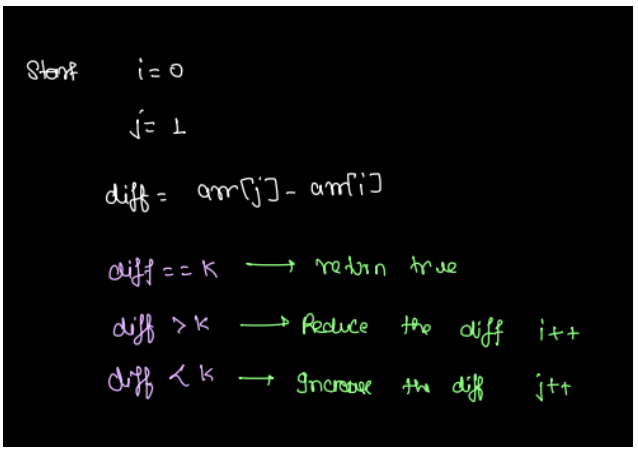
Initialization: Start with i as the first index (0) and j as the second index (1).

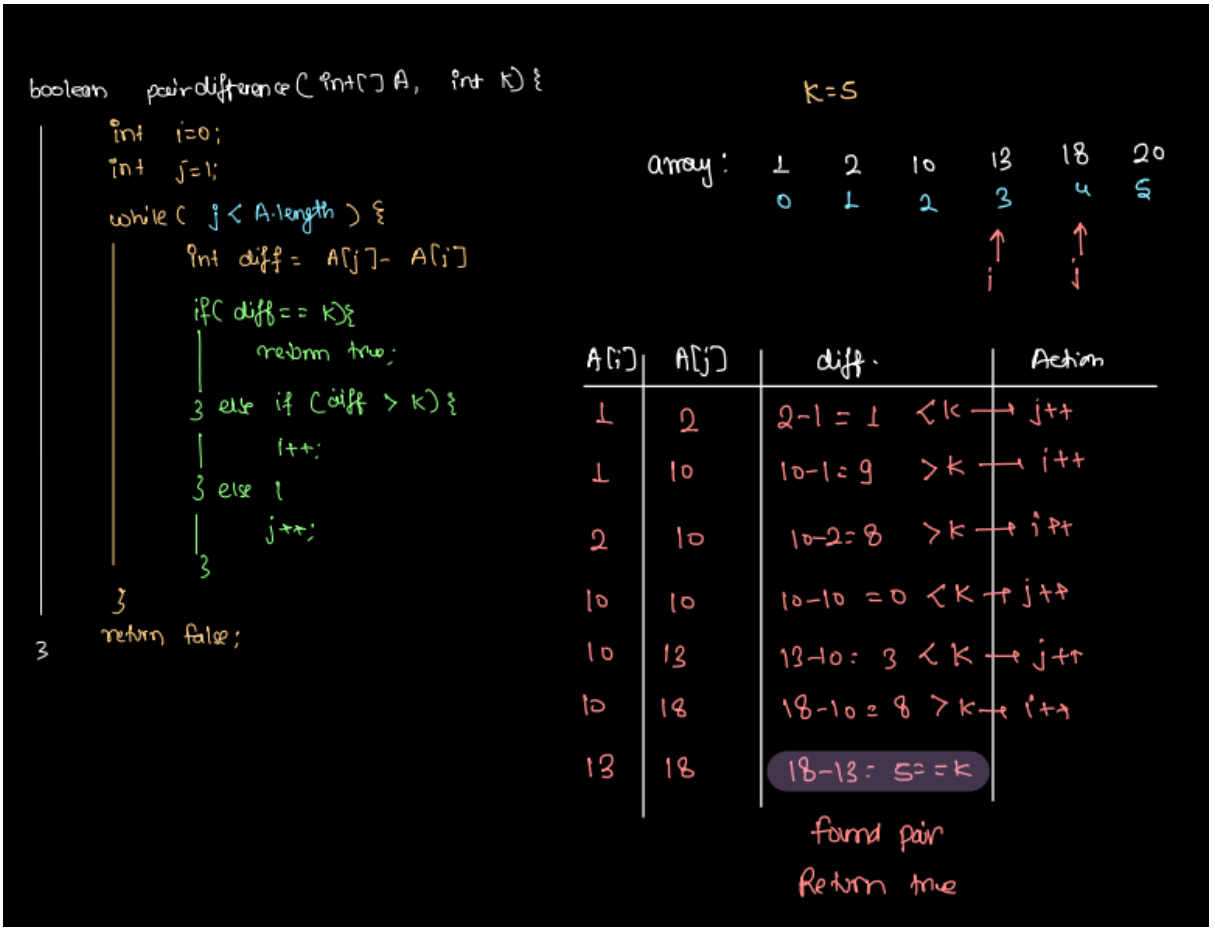
1. Loop condition: Continue the loop as long as i is less than j.
2. Calculate the difference: Calculate the difference between the elements at indices i and j.
3. Check the difference A[i]-A[j]: If the difference is less than the target value k, increment j by 1 (move the pointer to increase the difference count).
4. If the difference is greater than the target value k, increment i by 1 (move the pointer to reduce the difference).
5. If the difference is equal to the target value k, return true as you have found a pair that satisfies the condition.
6. Repeat steps 3-6 until i is no longer less than j.

In summary, this code snippet starts with i at index 0 and j at index 1. It calculates the difference between the elements at i and j and checks if it matches the target value k. Depending on the comparison result, it adjusts the pointers i and j accordingly. The loop continues until i is no longer less than j.









Container with most water —---------

Container with most water

Given an array where A[i] represents the height of each wall, select any two walls to maximize the accumulated water between them. return maximum amt of water that can be store here.

A = [3,6,4,5,2]

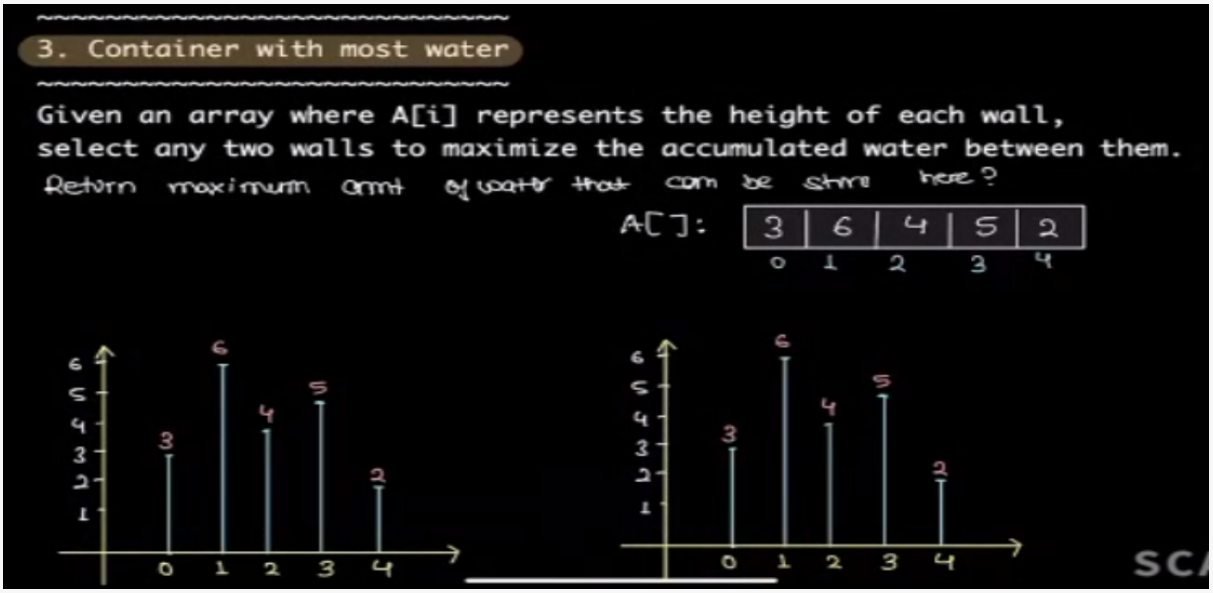
Here's a breakdown of the logic:

1. Initialize variables: Set water to 0 to store the maximum amount of water.
2. Set up a while loop: The loop condition i < j ensures that we continue iterating until the pointers i and j meet.

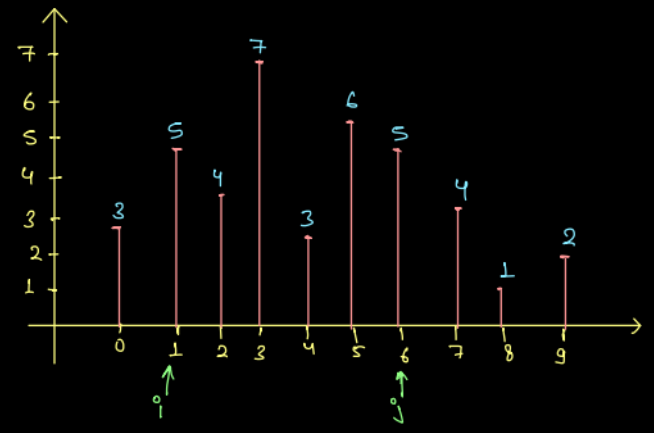
3. Calculate the amount of water: The formula `(j - i) * Math.min(A[i], A[j])` calculates the amount of water trapped between buildings at indices `i` and `j`. Here, `j - i` represents the building counts(length), and `Math.min(A[i], A[j])` represents the height of the smaller building.
4. Update the maximum water: Compare the calculated amount of water with the current maximum (`water`) and update `water` if the calculated amount is greater.
5. Move the pointers: Check the heights of buildings at indices `i` and `j`. If `A[i]` is greater than `A[j]`, decrement `j` by 1 (move the pointer towards the bigger building). Otherwise, increment `i` by 1 (move the pointer towards finding a bigger building).
6. Repeat steps 3-5 until the pointers `i` and `j` meet.

By following this logic, you can find the maximum amount of water that can be stored between the buildings in the given array.

https://www.interviewbit.com/snippet/67ac352ffb75e6e3b983/



A: [3 5 4 7 3 6 5 4 1 2]
0 1 2 3 4 5 6 7 8 9



What of Algo ?

A[i]	A[j]	ht	length	Amt	Action.
3	2	2	9	18	ht[i] > ht[j] → j--;
3	1	1	8	8	→ j--;
3	4	3	7	21	→ ht[i] < ht[j] → i++;
5	4	4	6	24	→ j--
5	5	5	5	25	→ i++ or j++;

```
int containerWithMostWater ( int[] ht) {
    int i= 0;
    int j= ht.length-1;
    int water=0;
    while (i < j) {
        int amt = (j-i) * Math.min( ht[i], ht[j]);
        water = Math.max(water, amt);
        //discard same pair of wall
        if (ht[i] > ht[j]) {
            j--;
        } else {
            i++;
        }
    }
    return water;
}
```

T.C : O(n)

S.C : O(1)

Pairs with given sum II

Problem Description

Given a sorted array of integers (not necessarily distinct) A and an integer B, find and return how many pair of integers (A[i], A[j]) such that i != j have sum equal to B.

Since the number of such pairs can be very large, return number of such pairs modulo (109 + 7).

https://www.interviewbit.com/snippet/aab7f095f5cd186f0d68/

DSA: Strings 1 - 24 - June 2023

Introduction of String

Sequence of character knows as string

What is character

Whatever we can type from keyboard is character
like
alphabets -> lowercase, upercase
numeric digits -> 0 - 9
Special Character -> other then this two is called special character even space.
ASSCII Range:
[97 - 122] -> a-z - lowercase

[65 - 90] -> A-Z - uppercase
[48 - 57] -> '0' - '9'
32 -> space

Behaviur of String in java

It is immutable means we can can not insert, removal and update of any character is not pssobile in string that behavier of string is known as immutability. Any operation with string is take o(n) complexity because on every operation it will create new string and append the new charecter like conctination.

Problem1: Given a string. toggle charecter and return ans string. Given string have lowecase + uppercase character (aBdHklmOp)
https://www.interviewbit.com/snippet/5903ebc3e9773722be81/

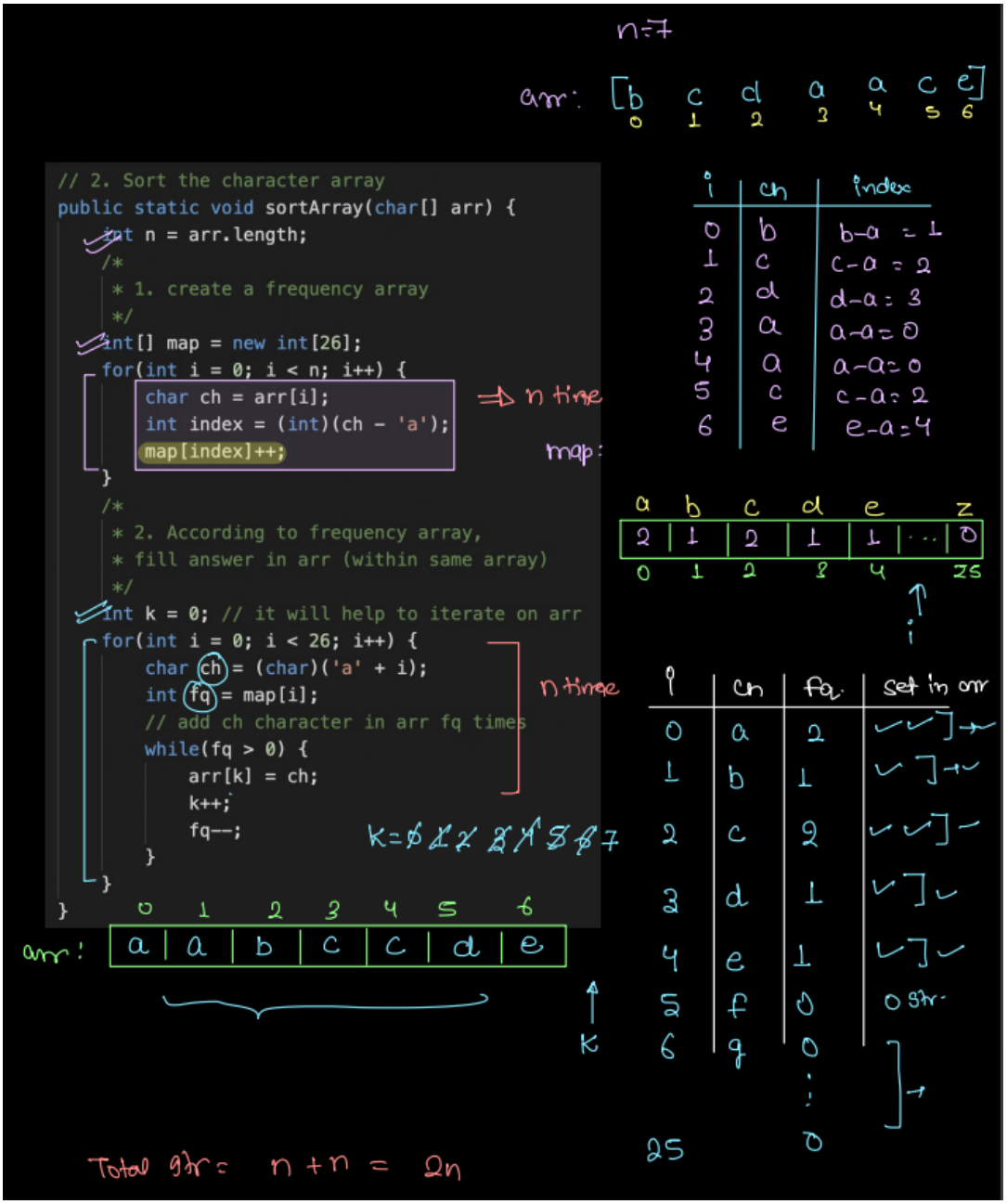
Problem: Given a character array ch[]. Sort it within same array.

```
char[] arr = {'d','a','b','a','d','a','c'};
```

https://www.interviewbit.com/snippet/f8b1896db3a3da851782/

Main logic ->

- Create an array called countArr with a length of 26 to store the count of each character.
- Iterate over the char array until its length, and convert each character to an index by subtracting 97 or the ASCII value of 'a'. Since 'a' has an ASCII value of 97, subtracting 97 from it will yield 0. Increase the count at the corresponding index in the countArr by 1.
- Iterate over countArr until 26 and retrieve the frequency count for each character stored in countArr[i]. Convert this frequency count and use a while loop to iterate as long as the count is greater than 0.
- Assign the value of countArr[i], after converting it to a character, to the corresponding index in the existing char array. Maintain an index variable to track the index for the new array and increase index by 1 on every iteration of the while loop.
- Assign the updated char array back to the original array, arr, using the expression arr[idx] = (char) countArr[i] + 'a'.



Given a string, find the length of longest palindromic
https://www.interviewbit.com/snippet/4d91475b4c3ddaaee1df/
String str = "xbdyzzydbrdyzydp"
Main Logic
We can solve this problem using pointers. There are two types of palindromic strings: even length and odd length. We need to iterate through two for loops, one for even-length palindromic strings and the other for odd-length palindromic strings.

For even-length palindromic strings, we check if the right-side value is equal. If it is true, we iterate from `i = 0` **to** `i < n - 1`**. To avoid going out of bounds of the array, we set** `n-2` **as the upper limit. Here, we define** `p1 = i` **and** `p2 = i + 1`**, and calculate the maximum value between** `ans` **and** `isPalindromic(string, p1, p2)` **using** `Math.max(ans, isPalindromic(string, p1, p2))`**.**

For odd-length palindromic strings, we iterate from `i = 1` **to** `i < n - 1`**. We define** `p1 = i - 1` **and** `p2 = i + 1` **and calculate the maximum value between** `ans` **and** `isPalindromic(string, p1, p2)` **using** `Math.max(ans, isPalindromic(string, p1, p2))`**. Finally, we return** `ans` **as the count.**

To clarify, `isPalindromic(string, p1, p2)` **is a function that checks if the substring from index** `p1` **to index** `p2` **in the string** `string` **is a palindrome.**

In summary, the code follows these steps to find the maximum count of palindromic substrings:

Iterate for even-length palindromic strings:
- **Check if the right-side value is equal.**
- **and add this condition in above p1 => 0 && p2 < n && above condition to protect indexOutof bound error.**
- **Iterate from** `i = 0` **to** `i < n - 1` **(**`n-2` **as the upper limit).**
- **Set** `p1 = i` **and** `p2 = i + 1`**.**
- **Calculate the maximum between** `ans` **and** `isPalindromic(string, p1, p2)` **using** `Math.max(ans, isPalindromic(string, p1, p2))`**.**

Iterate for odd-length palindromic strings:
- **Iterate from** `i = 1` **to** `i < n - 1`**.**
- **Set** `p1 = i - 1` **and** `p2 = i + 1`**.**
- **Calculate the maximum between** `ans` **and** `isPalindromic(string, p1, p2)` **using** `Math.max(ans, isPalindromic(string, p1, p2))`**.**

Return `ans` **as the count of palindromic substrings.**

To determine if a substring is palindromic, we utilize a `while` **loop. Within this loop, if** `str.charAt(p1)` **is equal to** `str.charAt(p2)`**, we decrement** `p1` **and increment** `p2`**. Ultimately, the function returns** `count = p2 - p1 - 1` **as the character count of the palindromic substring.**

Why (p2 - p1) - 1; because we are finding length and hear p2-p1 is 0 base of index and we are just adding 1 to it for making in length because as we know we use n-1 for iterate thatsby we have to add 1.

Idea 1: Using bruteforce

Go on every possible substring and check if it is palindromic or not, if it is palindromic maximise length.

```
int solve (String str){
    int n= str.length();
    int ans=0;
    for(int s=0; s<n; s++){
        for(int e=s; e<n; e++){ → O(n²)
            // all possibility of starting
               and ending index.
            if( isPalindromic (str, s, e) == true){
                ans = Math.max(ans, e-s+1);
            }                    → O(n)
        }
    }
    return ans;
}
```

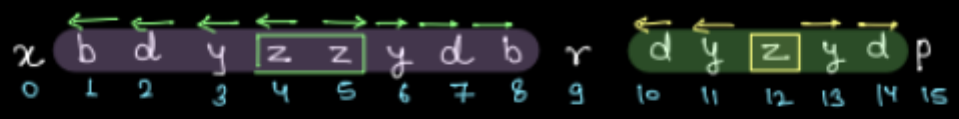
n=4

	e→
s=0	0, 1, 2, 3, 4
1	1, 2, 3, 4
2	2, 3, 4
3	3, 4
4	4

⇒ overall T.C = $O(n^2) * O(n)$

$= O(n^3)$

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



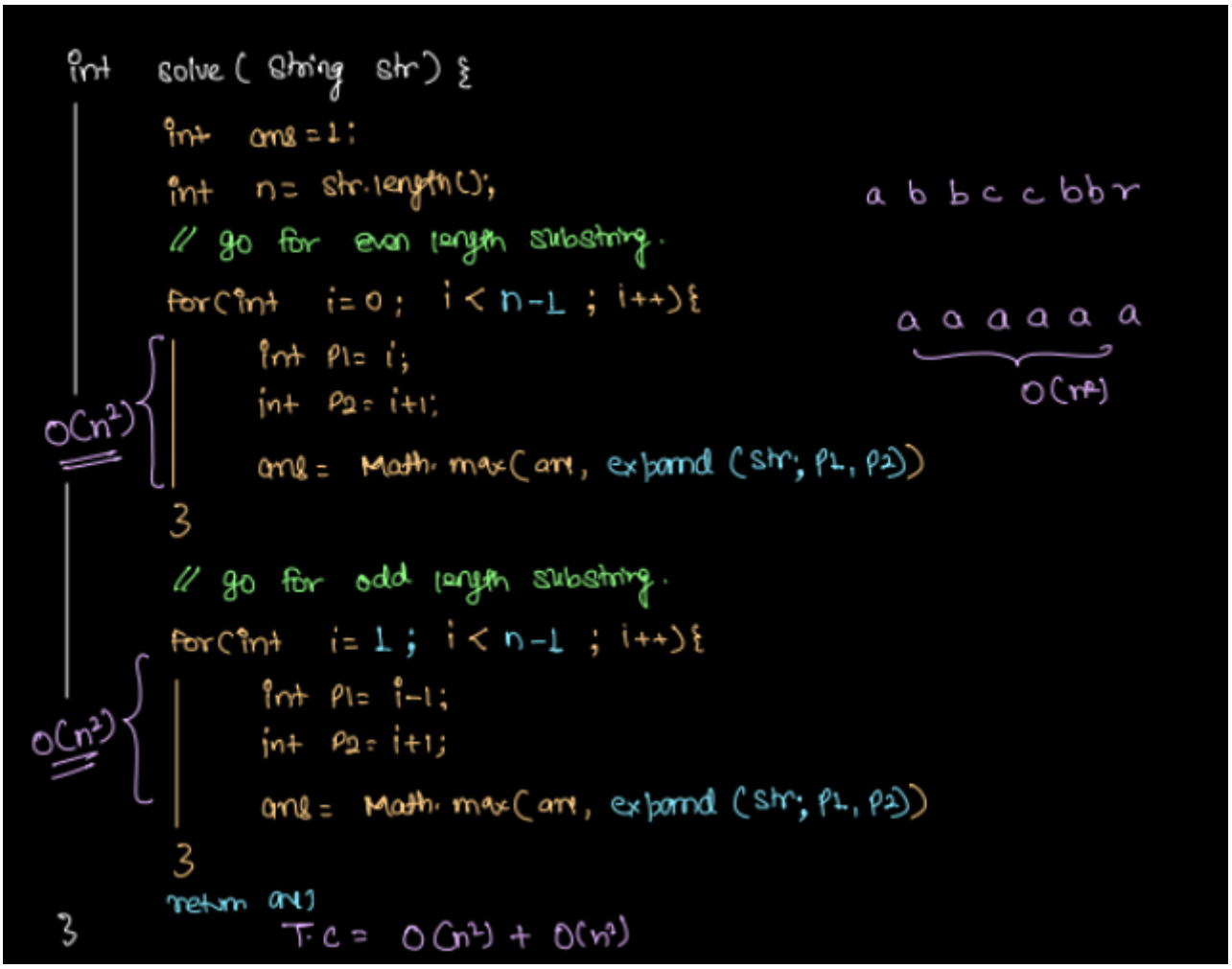
Observation:

① Even length palindromic substring

zz → yzzy → dyzzyd → bdyzzydb

② odd length palindromic substring

z → yzy → dyzyd



StringBuilder -> StringBulder have efficient time complexity over string and help us to concatenate and update the char in efficient time.

Disadvantage of string

- It is immutable
- Concatination is not efficient. it will take o(n)

StringBuilder can help us to manage both disadvantageor issue. Not technically, but for understanding StringBulider is kind of arraylist of character. StringBuilder is not immutable.

Syntax of StringBuilder

StringBuilder sb = new StringBuilder(“Hello”);
it will convert this hellow to [‘H’, ‘e’, ‘l’, ‘l’, ‘o’]

// Concatinate -> sb.append(character) // it will take o(1)

// How to iterate on StringBulider
for(int i = 0; i < sb.length(); i++){
char ch = sb.charAt(ch + “”);
}

// Update in O(1)
sb.setCharAt(5, ‘B’)
// Delete
- sb.deleteCharAt(index);

- **It will not return deleted character**
- **Delete last character will take O(1)**
- **if we will delete for first or from middle it will take O(n) because it will shift all character**

// How to convert StringBuilder to String.
it will take O(n) TC
sb.toString();

// Insert character on ith index
If we are inserting at last O(1) but sothing in middle it is O(n)
sb.insert(index, character);

Length of longest palindromic substring

StringBuilder

Problem Description
Akash likes playing with strings. One day he thought of applying following operations on the string in the given order:

Concatenate the string with itself.
Delete all the uppercase letters.
Replace each vowel with '#'.
You are given a string A of size N consisting of lowercase and uppercase alphabets. Return the resultant string after applying the above operations.

NOTE: 'a' , 'e' , 'i' , 'o' , 'u' are defined as vowels.

https://www.interviewbit.com/snippet/8c08c326ef5d6a25e3ef/

Count of occurrence

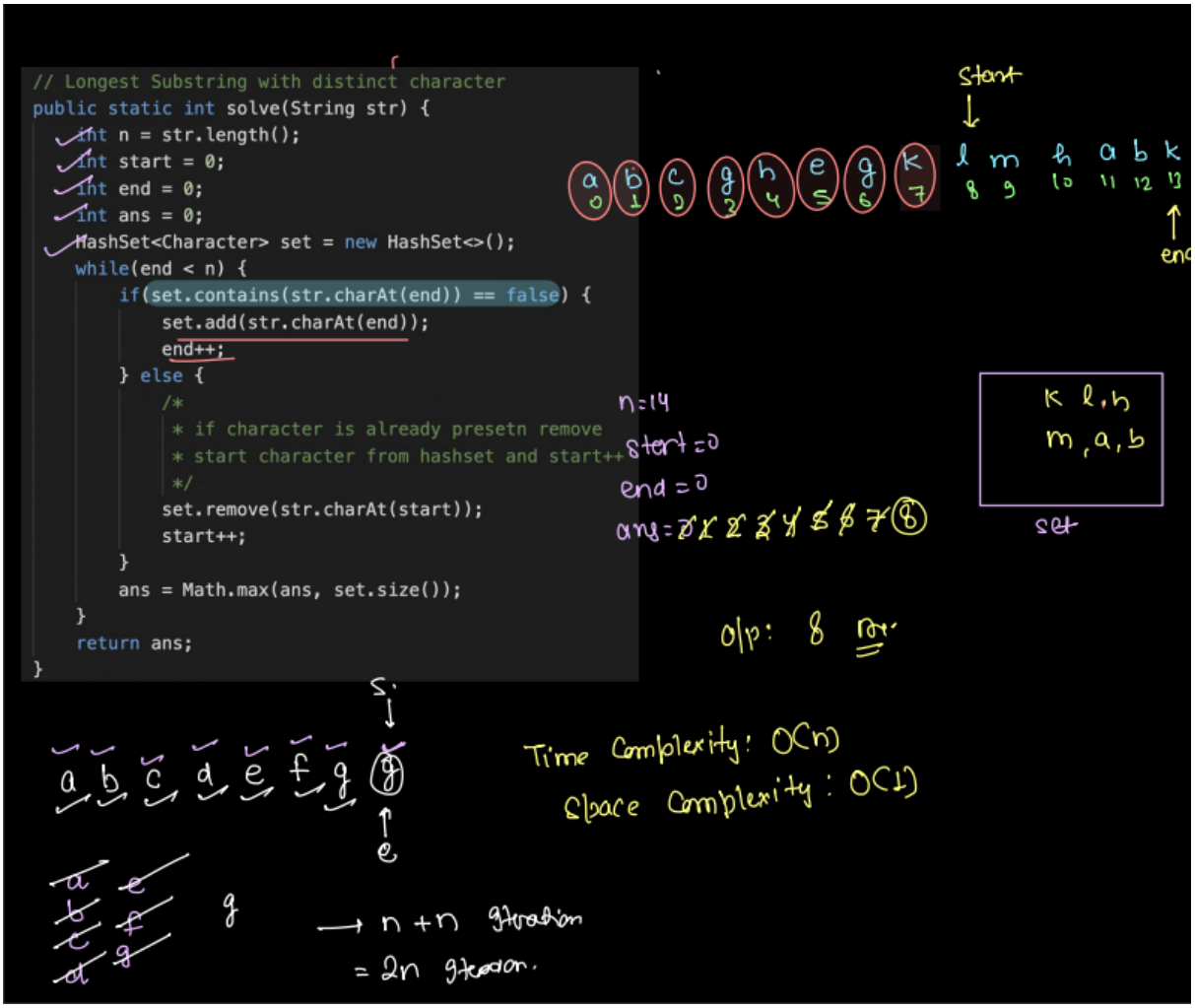
Problem Description
Find the number of occurrences of bob in string A consisting of lowercase English alphabets.

https://www.interviewbit.com/snippet/516668e89a05f6a6c0c2/

DSA: Strings 2 - 27 - June 2023

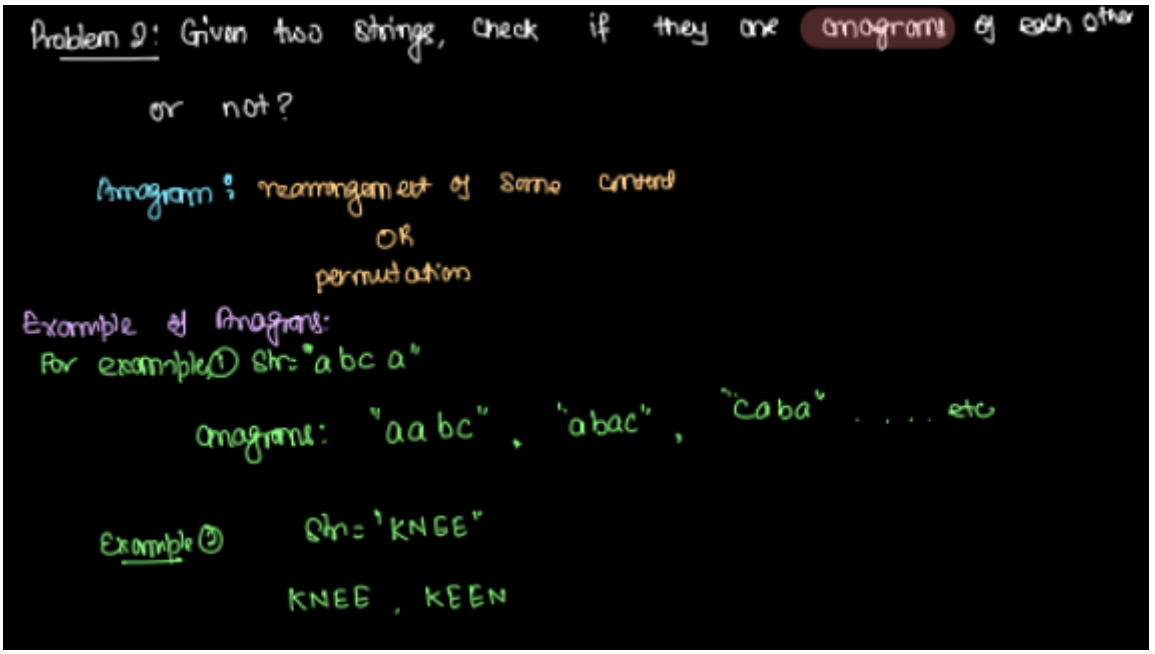
Longest Substring With Distinct Element

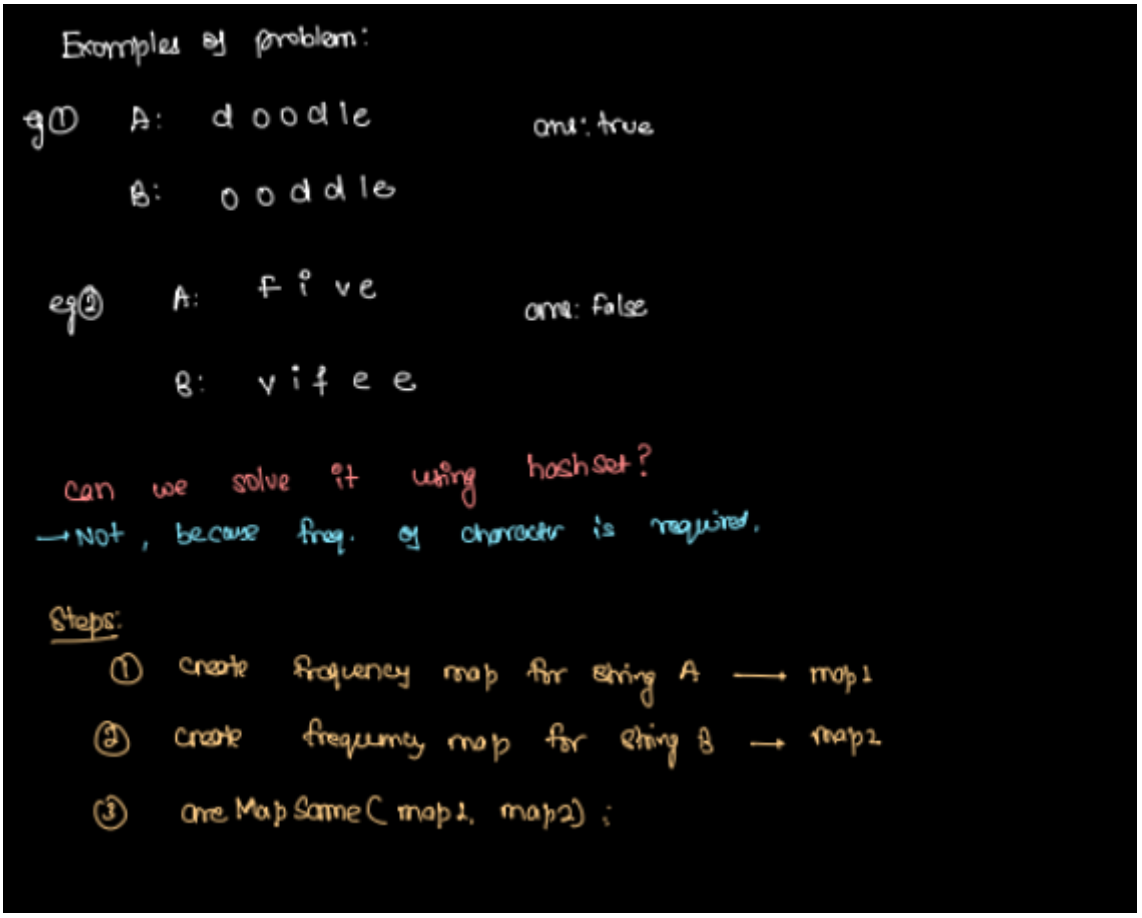
https://www.interviewbit.com/snippet/ddbc6bee46eeefe61b65/



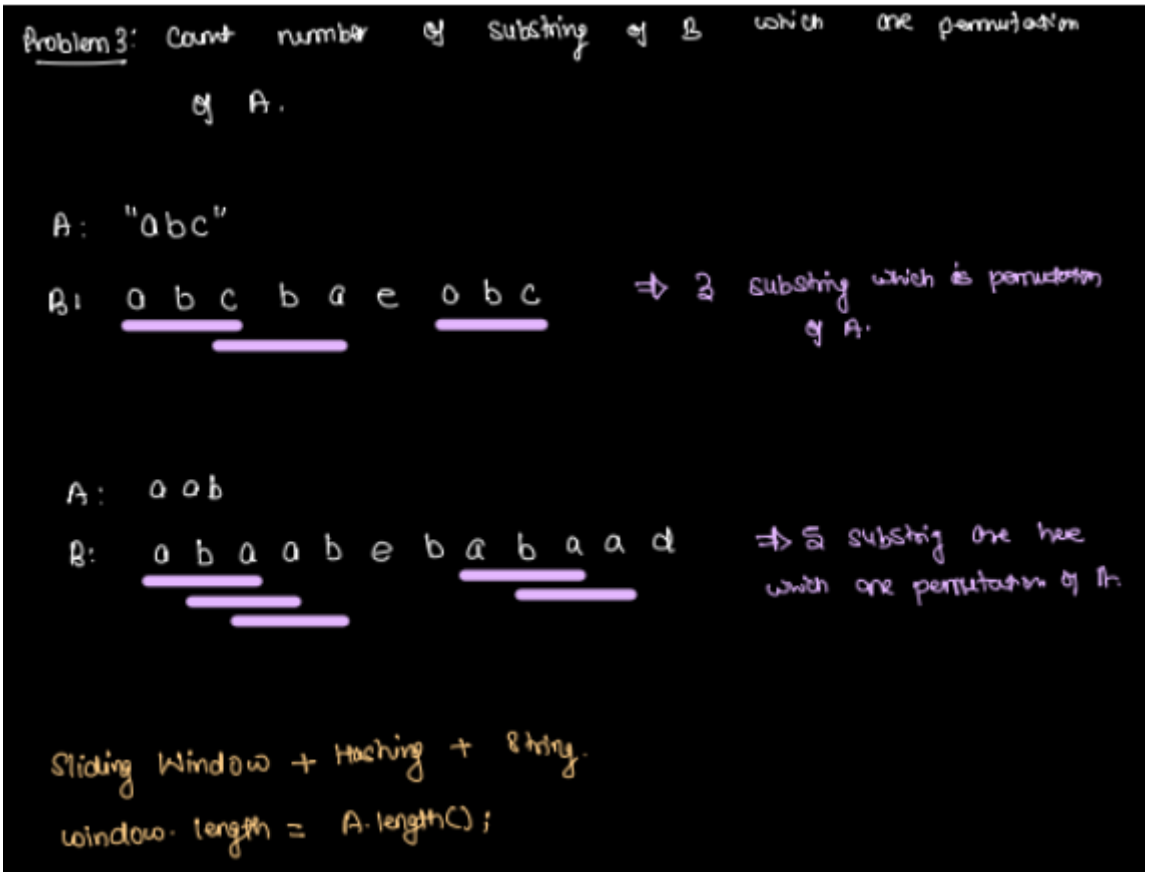
Anagrams Problem

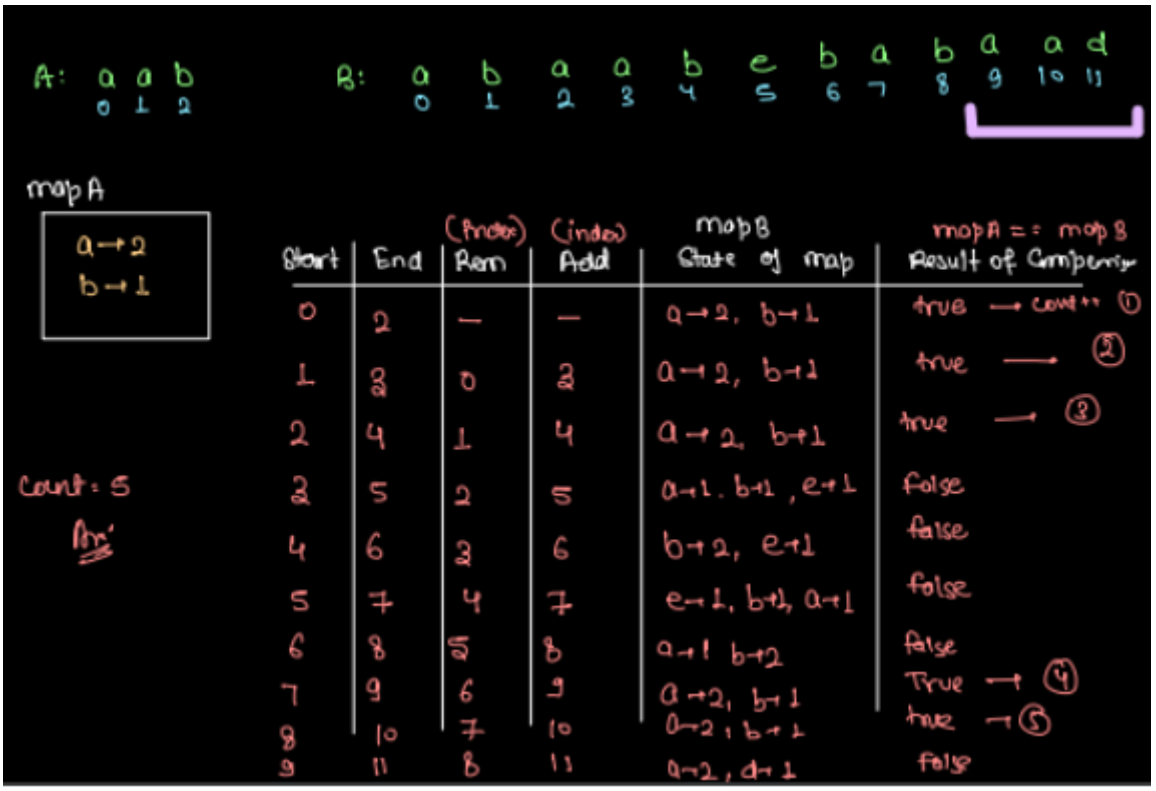
https://www.interviewbit.com/snippet/ea1d516ea21b69807661/





Count of Substring of B which is permutation of A
https://www.interviewbit.com/snippet/f0d306db93d8e8ad9316/
Optimized
https://www.interviewbit.com/snippet/65cc2383db5f5eb704bb/





Check Palindrome - 2

Problem Description

Given a string A consisting of lowercase characters.

Check if characters of the given string can be rearranged to form a palindrome.

Return 1 if it is possible to rearrange the characters of the string A such that it becomes a palindrome else return 0.

Main Logic ->

- Create a frequency map to count the occurrences of each character in the string A. This map will store each character as a key and its count as the corresponding value.
- Count the number of characters in A that have an odd count. If there are more than one character with an odd count, it is not possible to rearrange the characters to form a palindrome. Return 0 in this case.
- If there is at most one character with an odd count, it is possible to rearrange the characters to form a palindrome. Return 1 in this case.

  https://www.interviewbit.com/snippet/9fe693e66172bfa88238/

Check anagrams

Problem Description

You are given two lowercase strings A and B each of length N. Return 1 if they are anagrams to each other and 0 if not.

Note : Two strings A and B are called anagrams to each other if A can be formed after rearranging the letters of B.

A = "secure"
B = "rescue"

Main logic ->

1. Initialize two frequency arrays, freqArrA and freqArrB, both with a length of 26, to store the character frequencies for the given strings.
2. Iterate through the characters of the first string using a for loop with an index variable i ranging from 0 to the length of the string.
3. Inside the loop, calculate the index of the current character by subtracting the ASCII value of 'a' (97) from the character's ASCII value.
4. Increment the count at the calculated index in freqArrA.
5. Repeat steps 3 and 4 for the second string to populate freqArrB.
6. After creating both frequency arrays, start a for loop with an index variable i ranging from 0 to 25.
7. Inside the loop, check if the frequency value at index i in freqArrA is not equal to the frequency value at index i in freqArrB.
8. If the above condition is true, return 0 to indicate that the frequencies are not equal.
9. After the loop, if all the frequency values are equal, return 1 to indicate that the frequencies are equal.
10. End the function.

https://www.interviewbit.com/snippet/9003730c4d2b8830348c/

DSA: Pattern Matching - 1 - july 2023

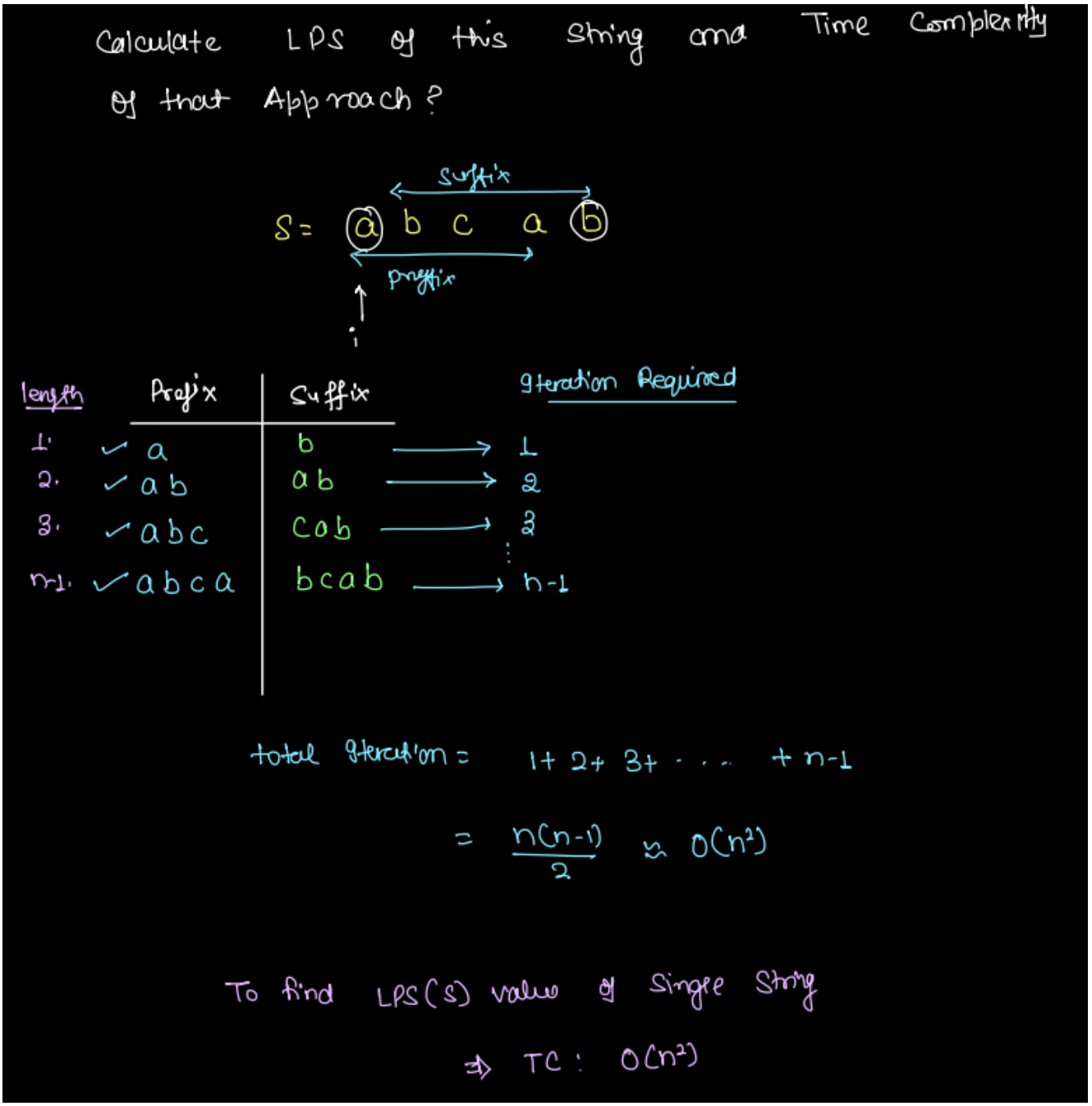
Prefix and suffic string (str = abab)

- Substring starting from index 0 is calling prefix strings. like **a, ab, aba, abab**
- Substring ending at index n-1 is calling suffix strings. like **b, ab, bab, abab**

LPS of string ->

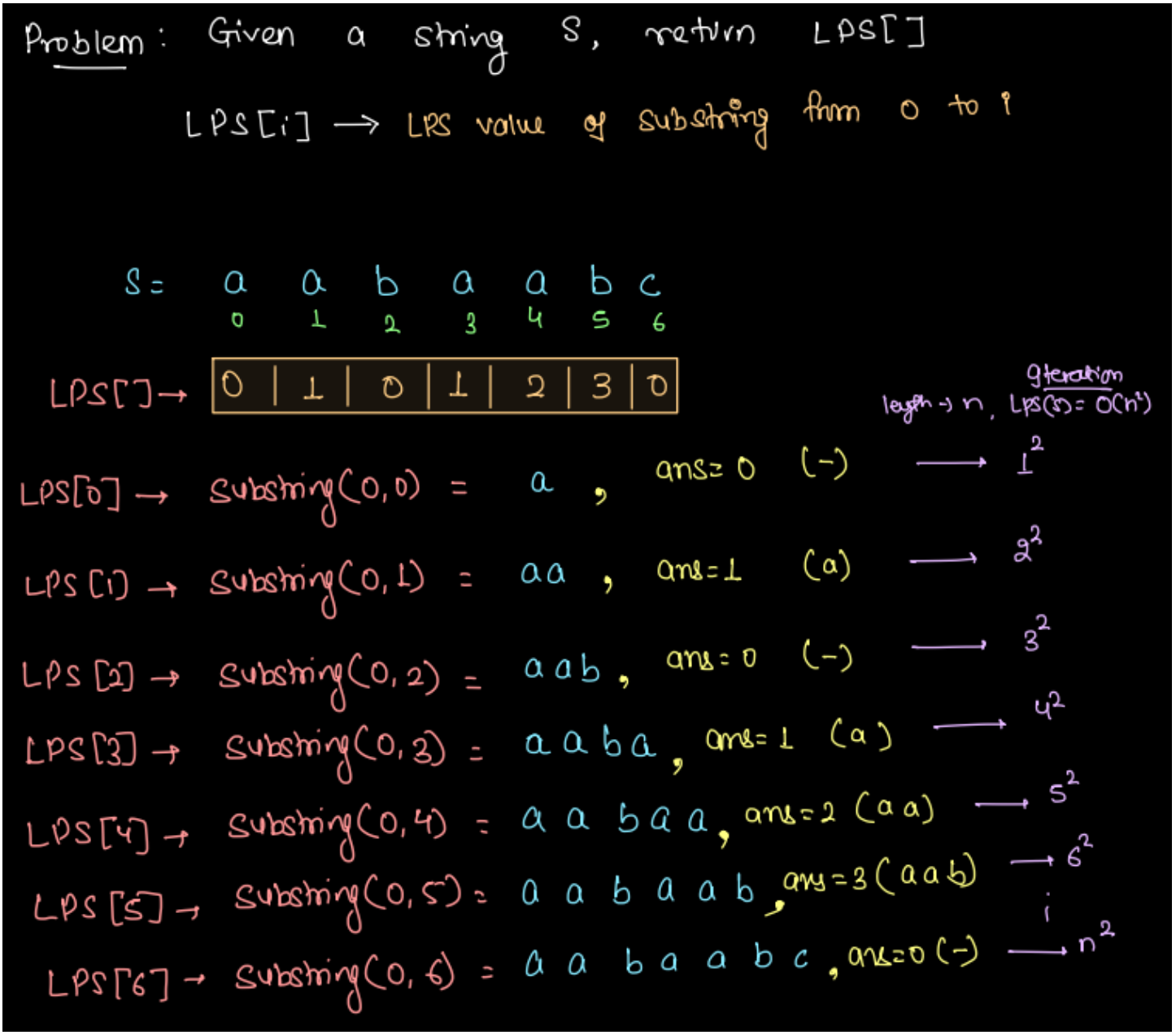
prefix string which also match with suffix is called LPS string

Prifix string -> substring starting from 0
Sufix string -> substring ending at index n-1;



LPS string -> Length of longest prefix which is also a suffix is LPS of string excluding complete string.

LPS Array is the array of substring which logest and suffix and prfix is equals.



Given a string S = “aabaabc”, return LPS[] of string

LPS[i] -> LPS value of substring from 0 to i;

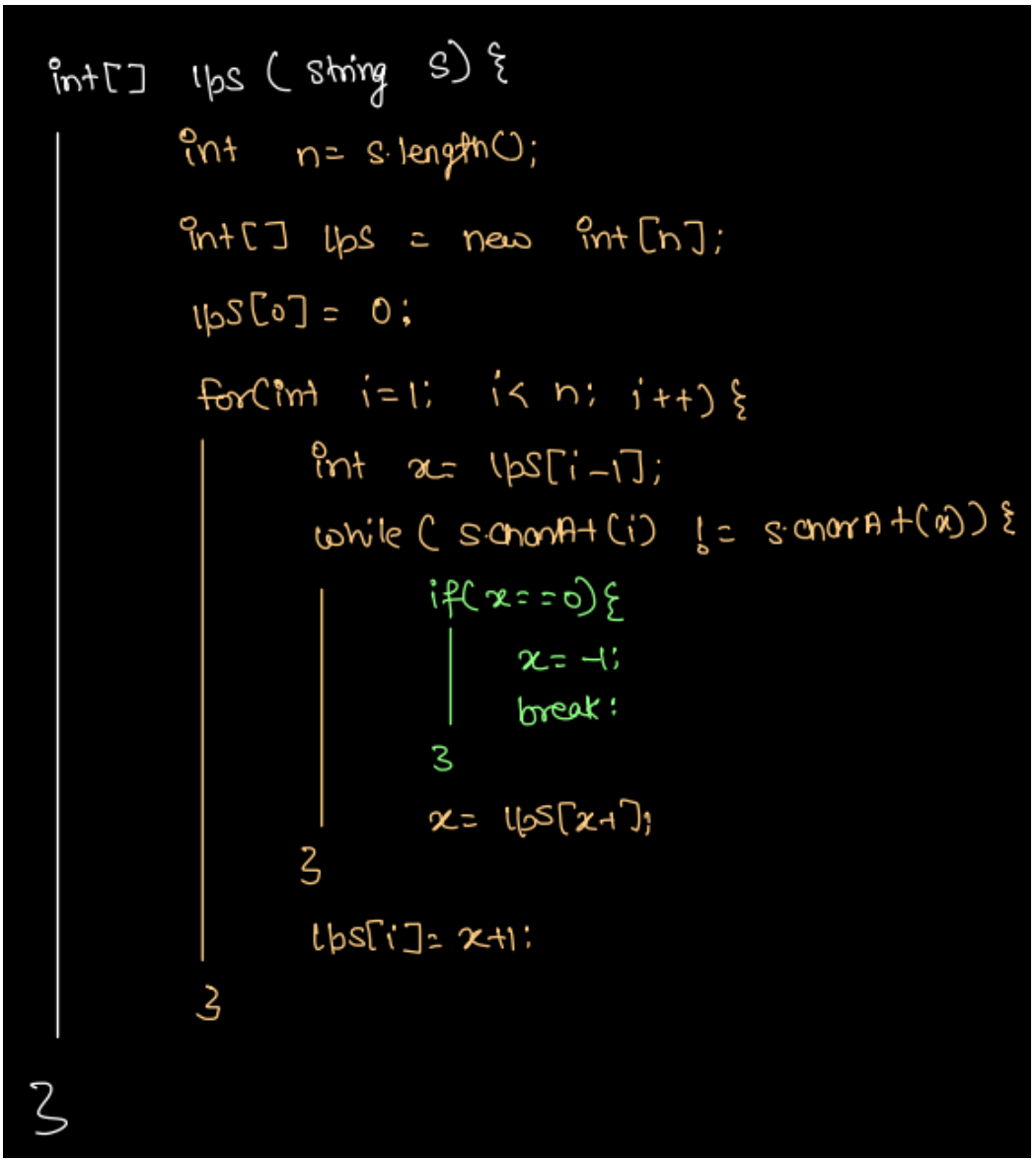
https://www.interviewbit.com/snippet/e52dbf10c3bcab37596b/

LPS -> get lps array of string

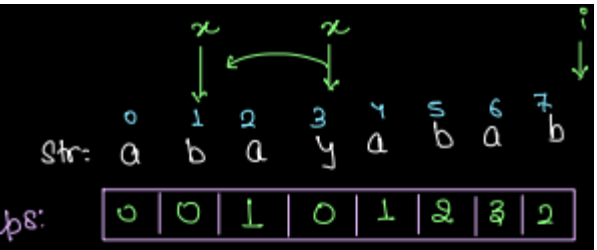
To find the Longest Proper Prefix which is also a Suffix (LPS) of a string array, follow these steps:

- Create an LPS array of length n and initialize it with 0 values.
- Iterate through the array from i = 1 to i < n using a for loop.
- Inside the loop, assign x = LPS[i-1].
- Use a while loop with the condition str.charAt(i) != s.charAt(x).
- Inside the while loop, add a conditional check: if x == 0, set x = -1 and break the loop.
- After the conditional check, write x = LPS[x-1].
- Close the while loop.
- Set LPS[i] = x + 1.
- Close the for loop.

- Return the LPS array.



```
int[] lps (String s) {
    int n = s.length();
    int[] lps = new int[n];
    lps[0] = 0;
    for(int i=1; i<n; i++) {
        int x = lps[i-1];
        while (s.charAt(i) != s.charAt(x)) {
            if(x==0){
                x=-1;
                break;
            }
            x = lps[x-1];
        }
        lps[i] = x+1;
    }
}
```



T.C.: O(n)

n = 8

i = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7

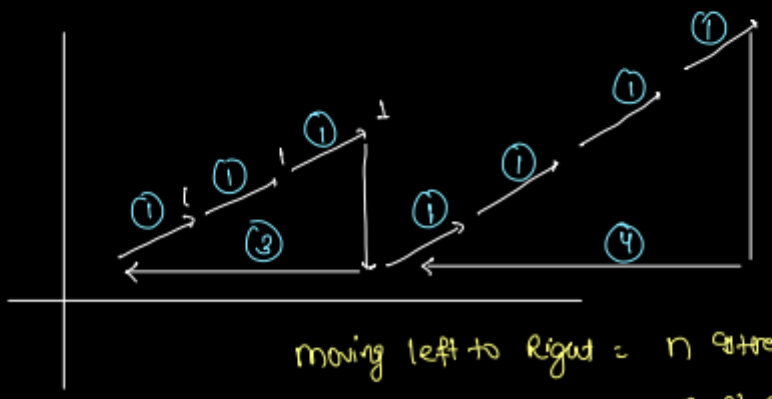
x = ~~3~~ 1

8:32 — 8:45 Am (break time)

① Time Complexity

② KMP (Pattern matching)

Time Complexity:



moving left to Right = n operation.

moving Right to left = n operation.

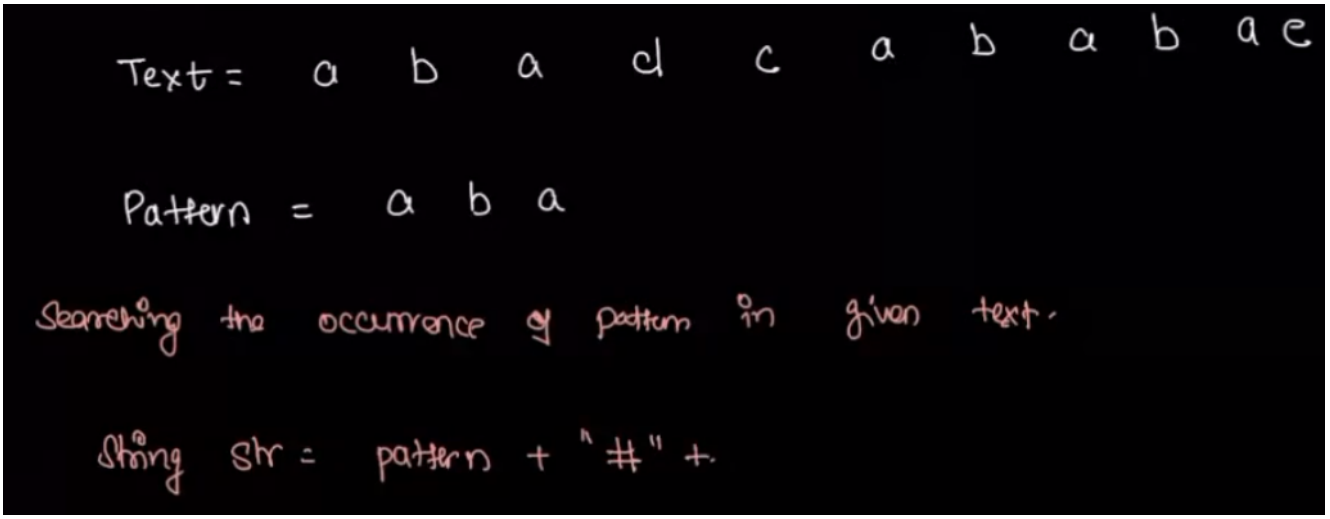
Total opr = n + n = 2n

= O(n)

Count occurrence of Pattern P in text T Matching KMP
(Knurth-Moris-pratt) Algo - This algo is use to Search the occurrence of pattern in given text.
https://www.interviewbit.com/snippet/e1835e09d8ad879e9f27/
Main logic -> To achieve the desired outcome,

- we need to form a single string by combining the pattern, a delimiter (such as $, %, or &), and the text.
- we calculate the LPS array (Longest Proper Prefix which is also a Suffix) of the created string.
- We then iterate through the LPS until its length, checking if LPS[i] is equal to the length of the pattern. If it is, we increment the count by 1 and return the answer.

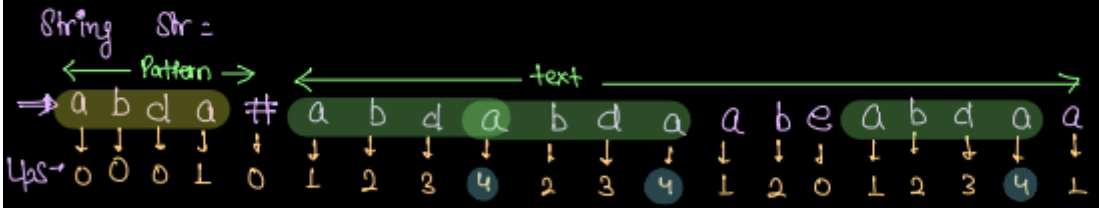
Given the text "abadcababac" and the pattern "aba", we can represent the string as follows: str = pattern + "$" + text; We can use any delimiter instead of "$" like "%", "&".



T = a b d a b d a a b e a b d a a

P = a b d a

String Str = p + "#" T




```
if( lps[i] == p.length()){
|     count++;
}
```

```
int patternMatching (String T, String P){
|   if( p.length() > T.length() ) {
|   |   return 0;
|   }
|   String str = p + "#" + T;
|   int[] lps = lps (Str);          ⟶ n+m
|   int ans = 0;
|   for( int i=0; i< lps.length; i++){ ⟶ n+m
|   |       if( lps[i] == p.length()) {
|   |       |     ans++;
|   |       }
|   }
|   return ans;
}
```

T.C : O(n+m)

S.P ⇒ O(n+m)

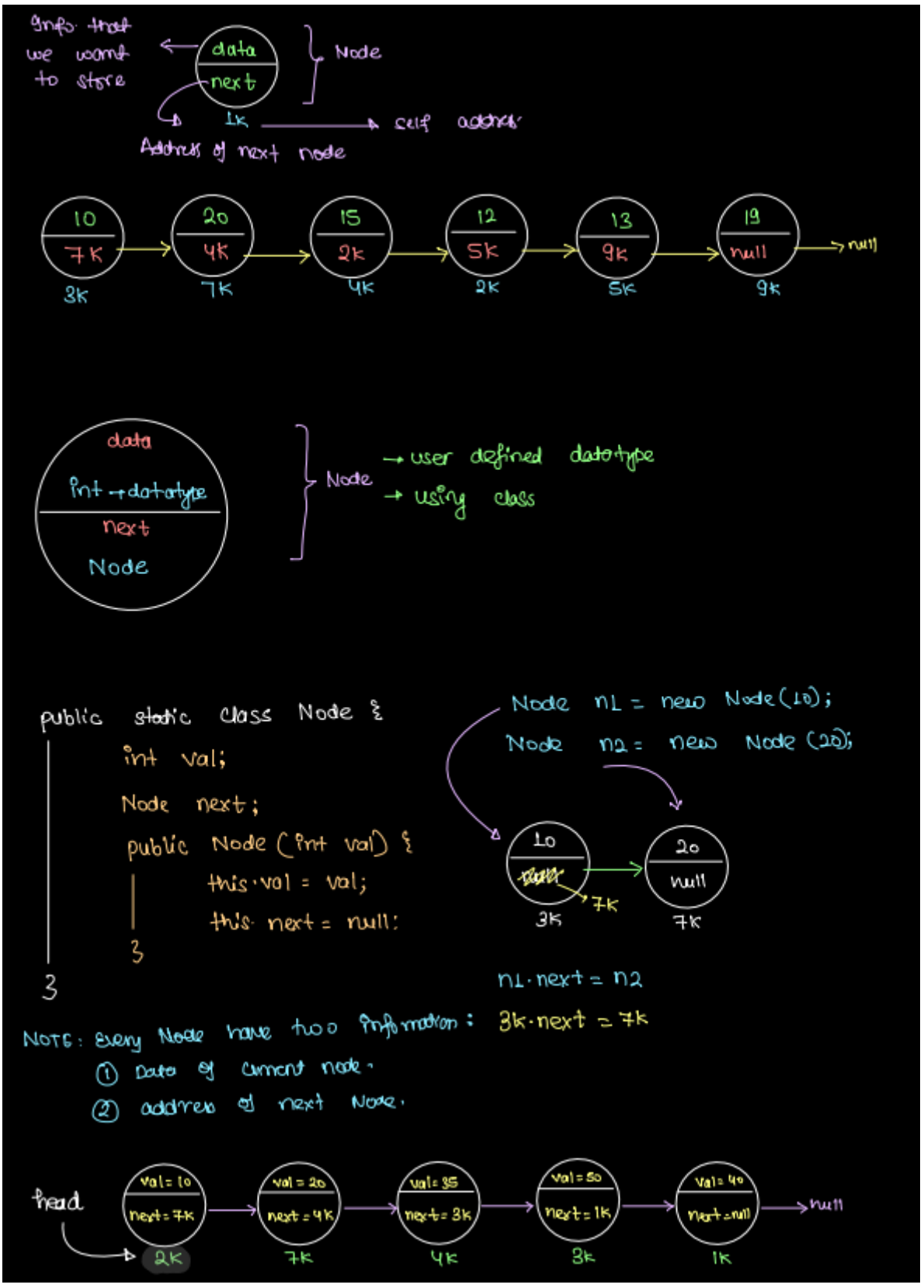
DSA: Linked List 1 - 4 - July 2023

Introduction of linkedlist

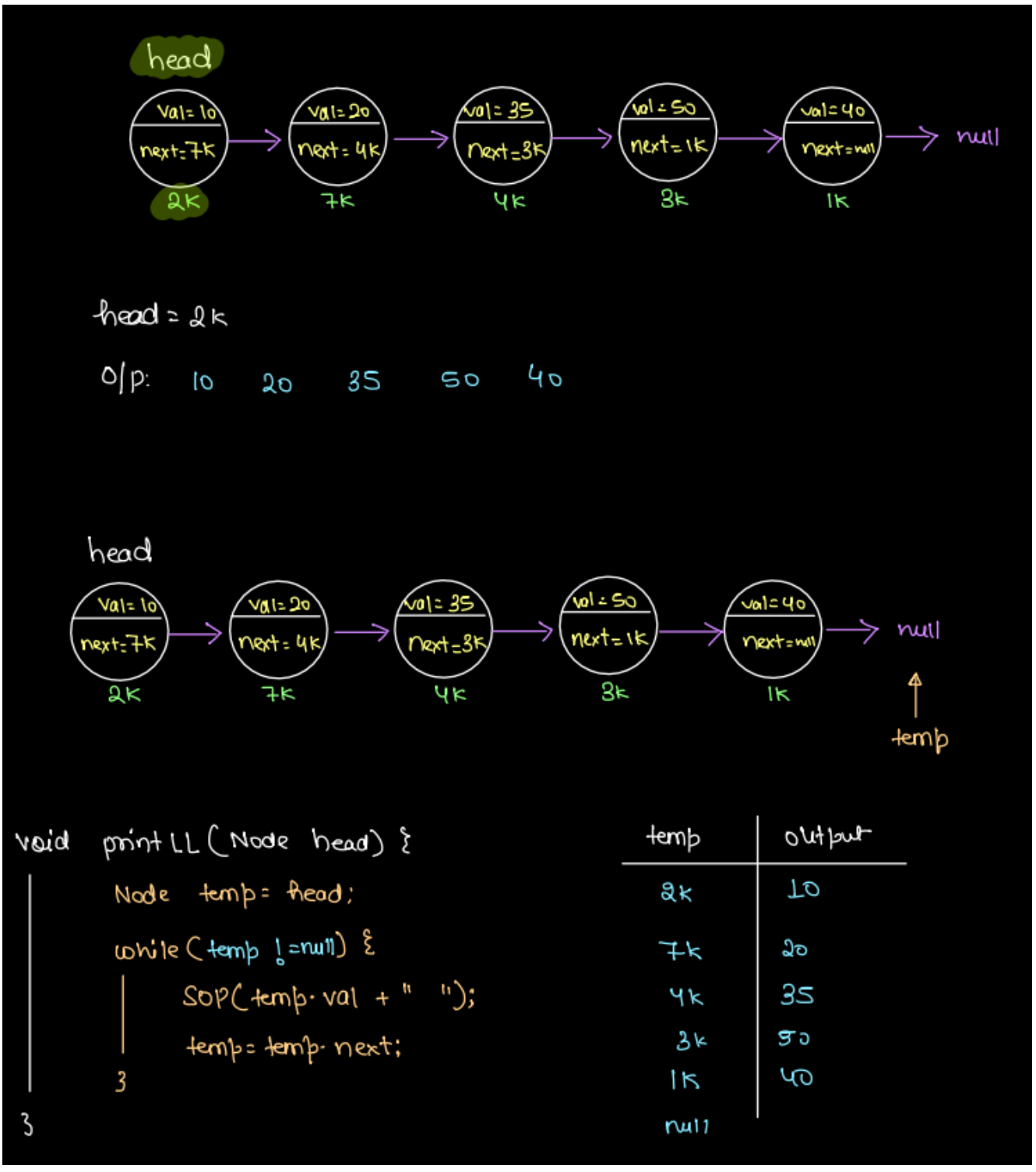
What is linked lsit and why it is required?

Whenever we store data in an array, continuous memory allocation is necessary. Arrays can accommodate up to 10^5 data elements, but when dealing with larger data sets and a lack of continuous memory space, linked lists come to our aid. Linked lists do not require continuous memory allocation; instead, they store nodes wherever memory is available and connect them to one another. Each node contains the address of the next node and a value. The first node is referred to as the head of the list.

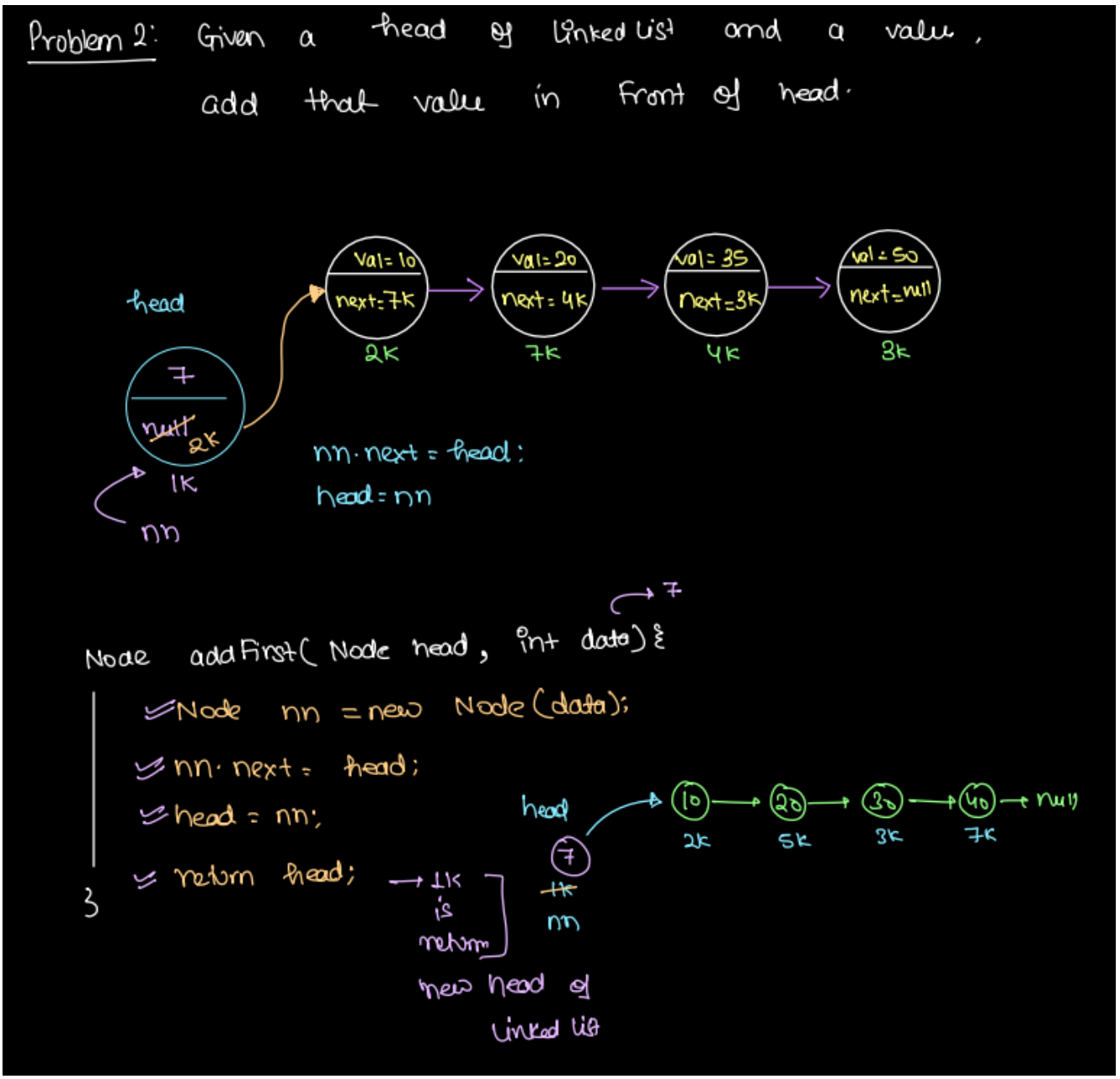
Structure of Node



Given head of the LinkedList print its content



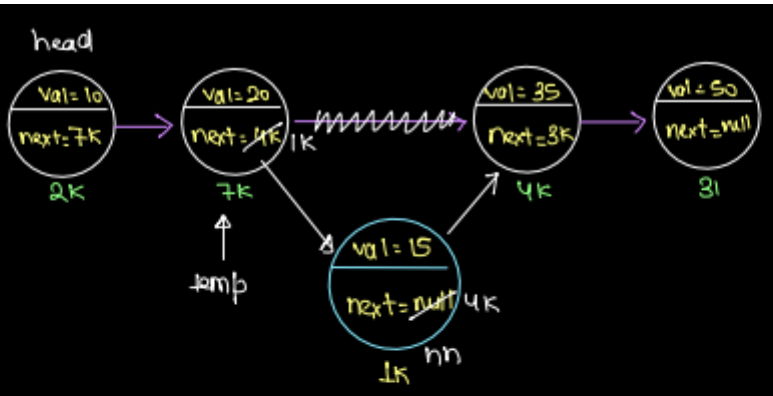
Given a head of LinkedList and a value, add that value in front of head



Given head of LinkedList, a data and an index. Add node at particular index in LinkedList

Steps

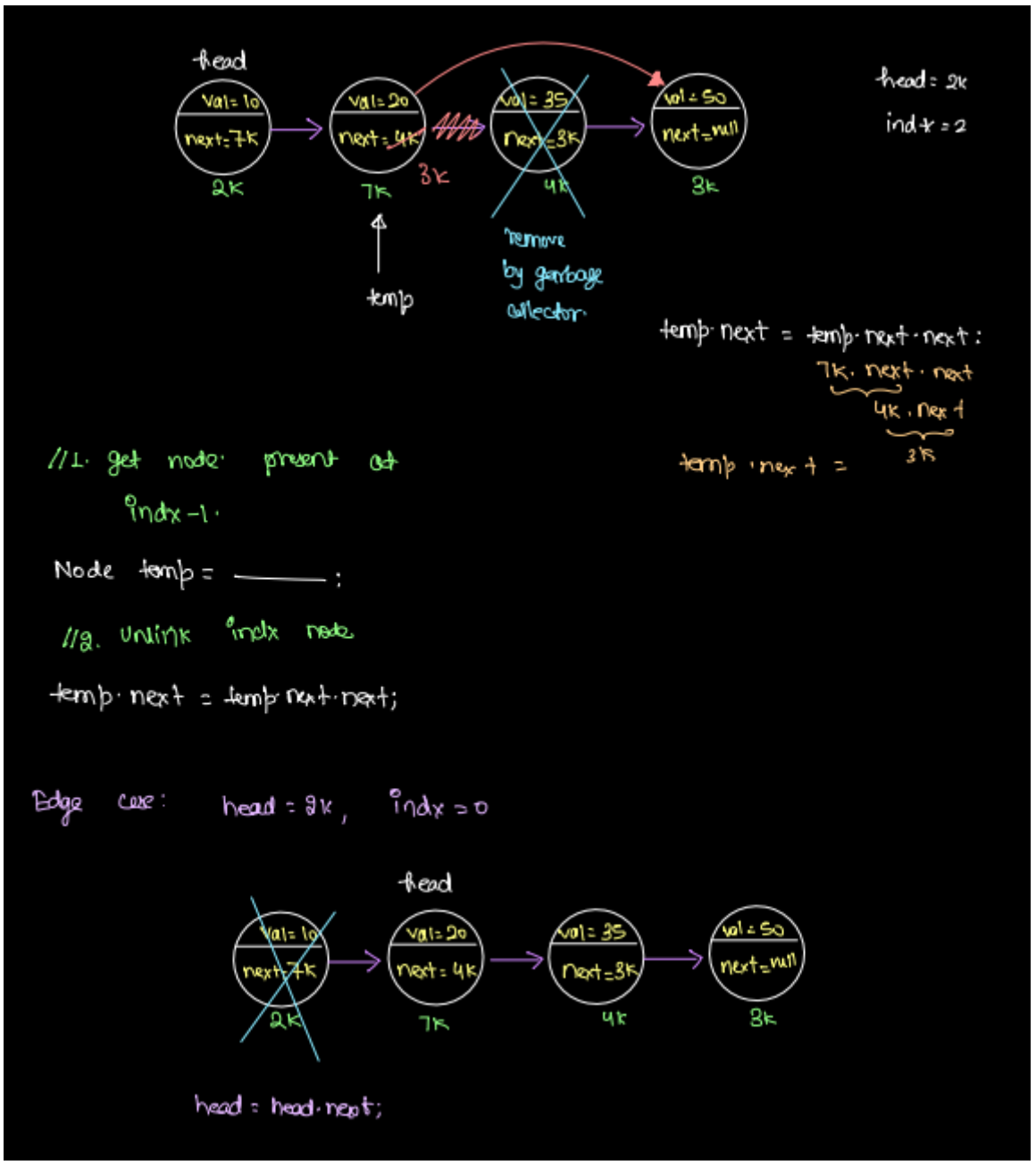
- First we will find index-1 node using for loop (int i = 1; i <=index-1; i++)
- Create new node with given data
- First link node with index nod like nn.next = temp1.next (temp is previous node here)
- Unlink temp1 node with assigned nod like index node temp1.next = nn;
    - There is 1 edge case if index=0 then we do not need to find node just add node at front of LinkedList like nn.next = head and head = nn;




```
// 1. Create a new Node with given data.
Node nn = new Node(data);

// 2. get Node from linked list which is
   available at index -1.
Node temp = ------.

// 2. Make a proper connection b/w temp & nn.
nn.next = temp.next;
temp.next = nn;
```

```
Node addAt( Node head, int data, int indx) {
    Node nn = new Node(data);
    if(indx == 0) {
        nn.next = head;
        head = nn;
    } else {
        // get Node at indx-1;
        Node temp = head;
        for(int k=1; k <= indx-1; k++) {
            temp = temp.next;
        }
        // link the nodes
        nn.next = temp.next;   } → order of linking step matter alot
        temp.next = nn;
    }
    return head;
}
```

Delete Node from a particuler index




```
Node removeAt(Node head, int indx){
    if(indx == 0){
        head = head.next;
    } else {
        // get node at indx-1;
        Node temp = head;
        for(int k=1; k<=indx-1; k++){
            temp = temp.next;
        }
        // unlink node present at indx
        temp.next = temp.next.next;
    }
    return head;
}
```

Given a head of LinkedList. Reverse the entire LinkedList and return new head

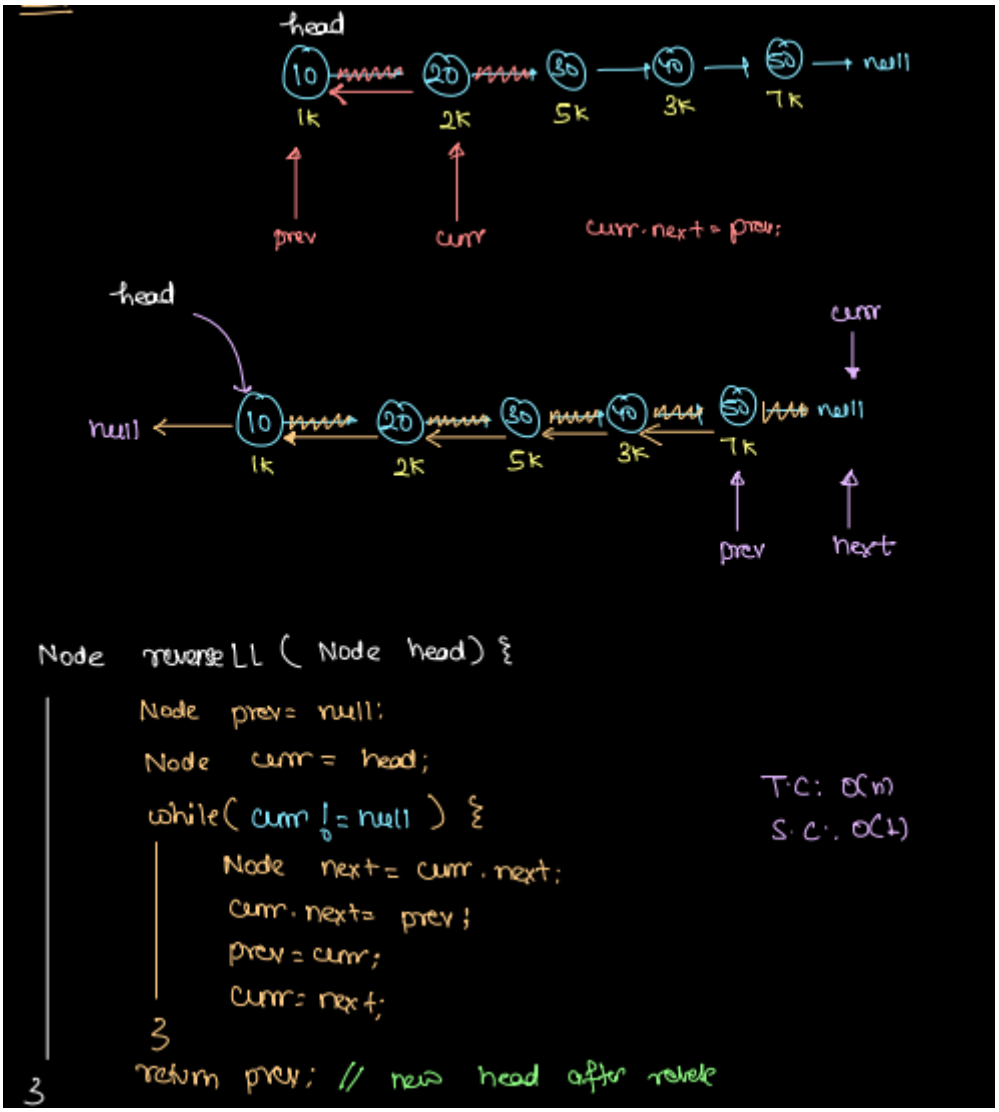
Main Logic ->

Step 1

first we will create prev node and assign it to null

- **Step 2**
  We will create curr node and assign it to head.
- **Step 3**
  Apply while loop till current != null. Inside while loop
  - **Create next node and assign it to current.next**
  - **Assign prev to current.next we are disconnecting current node and join with prev node**
  - **Now connect prev with current prev = curr**
  - **Now connect current with next curr = next close the loop**

- **Step 4**
  return prev.

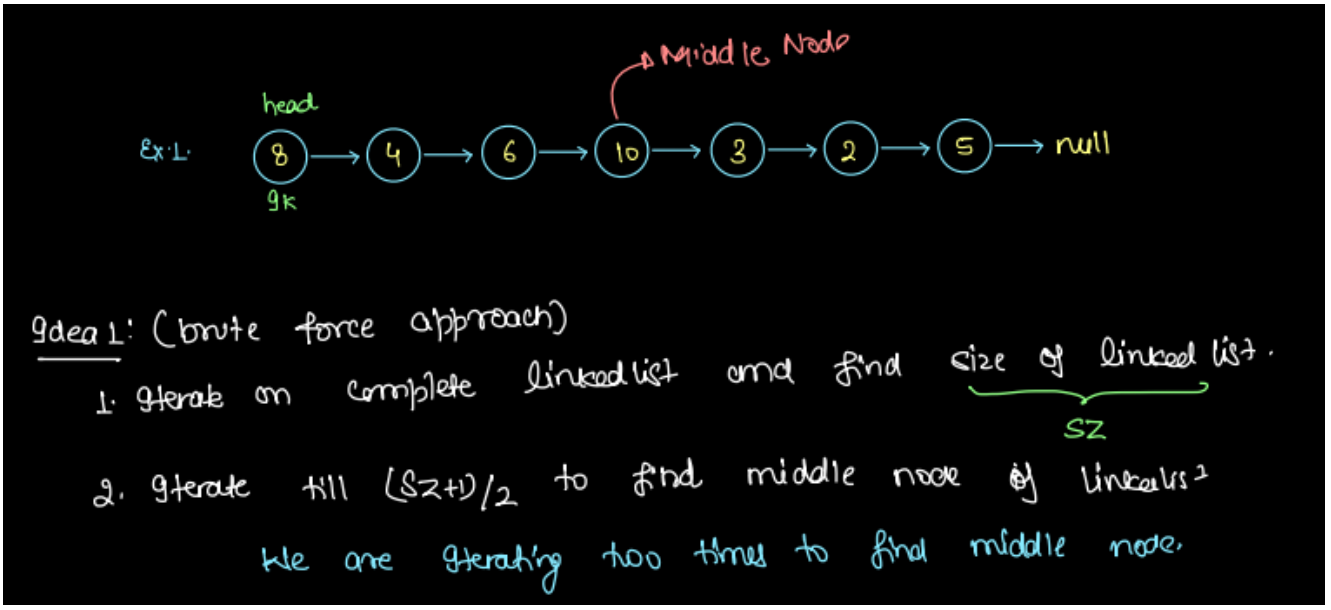


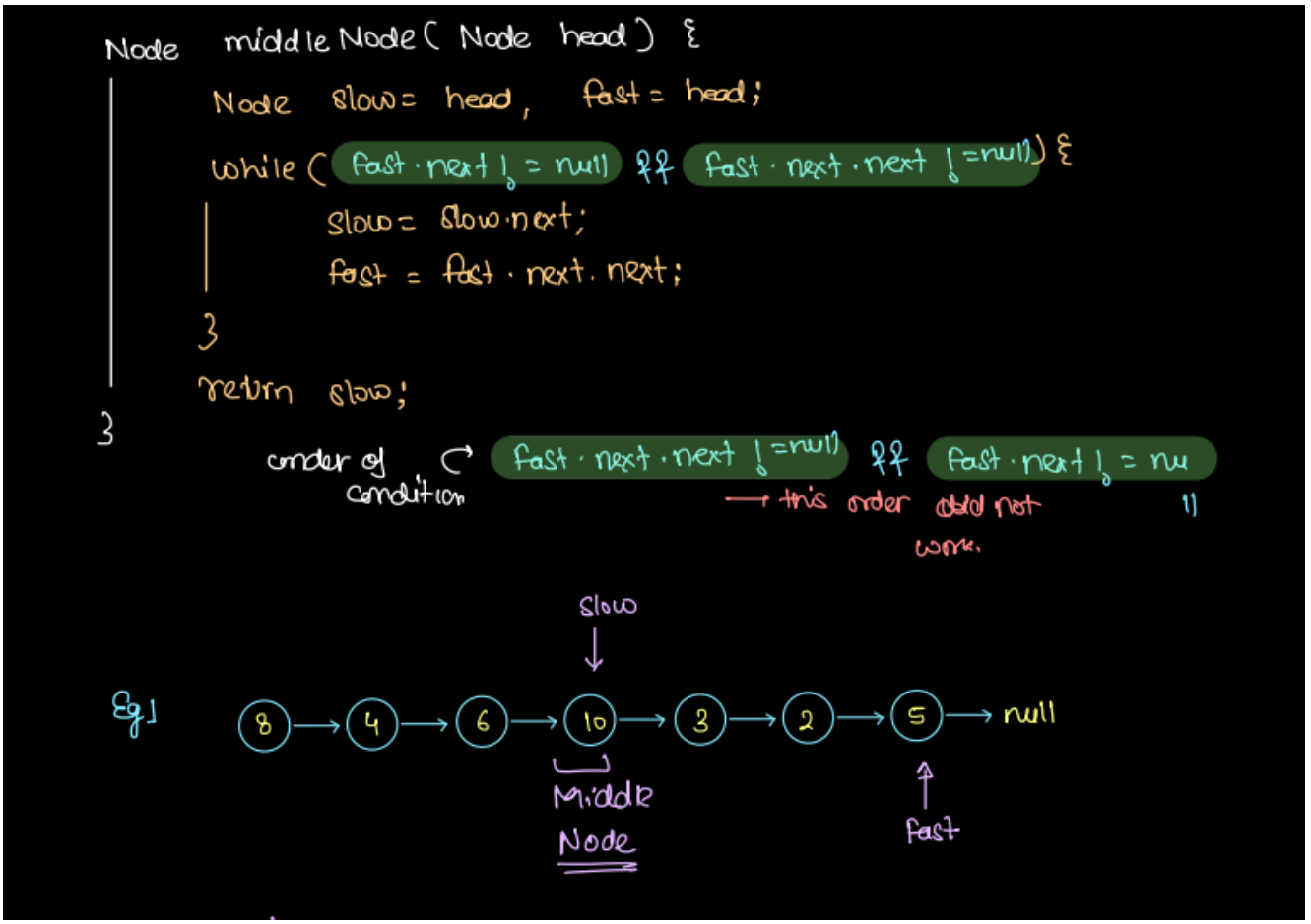
DSA: Linked List 2 - 6 July 2023

Given a LinkedList. Find and return mid Node

Main Logic:

To solve the problem, we will employ the Slow and Fast pointer technique. Initially, we create two pointers, slow and fast, both starting from the head node. We then enter a while loop with the condition (fast.next != null && fast.next.next != null). Within this loop, we update the slow pointer as slow = slow.next and the fast pointer as fast = fast.next.next. After the while loop concludes, we return the slow pointer.
https://www.interviewbit.com/snippet/b644bcfd278a627dc139/





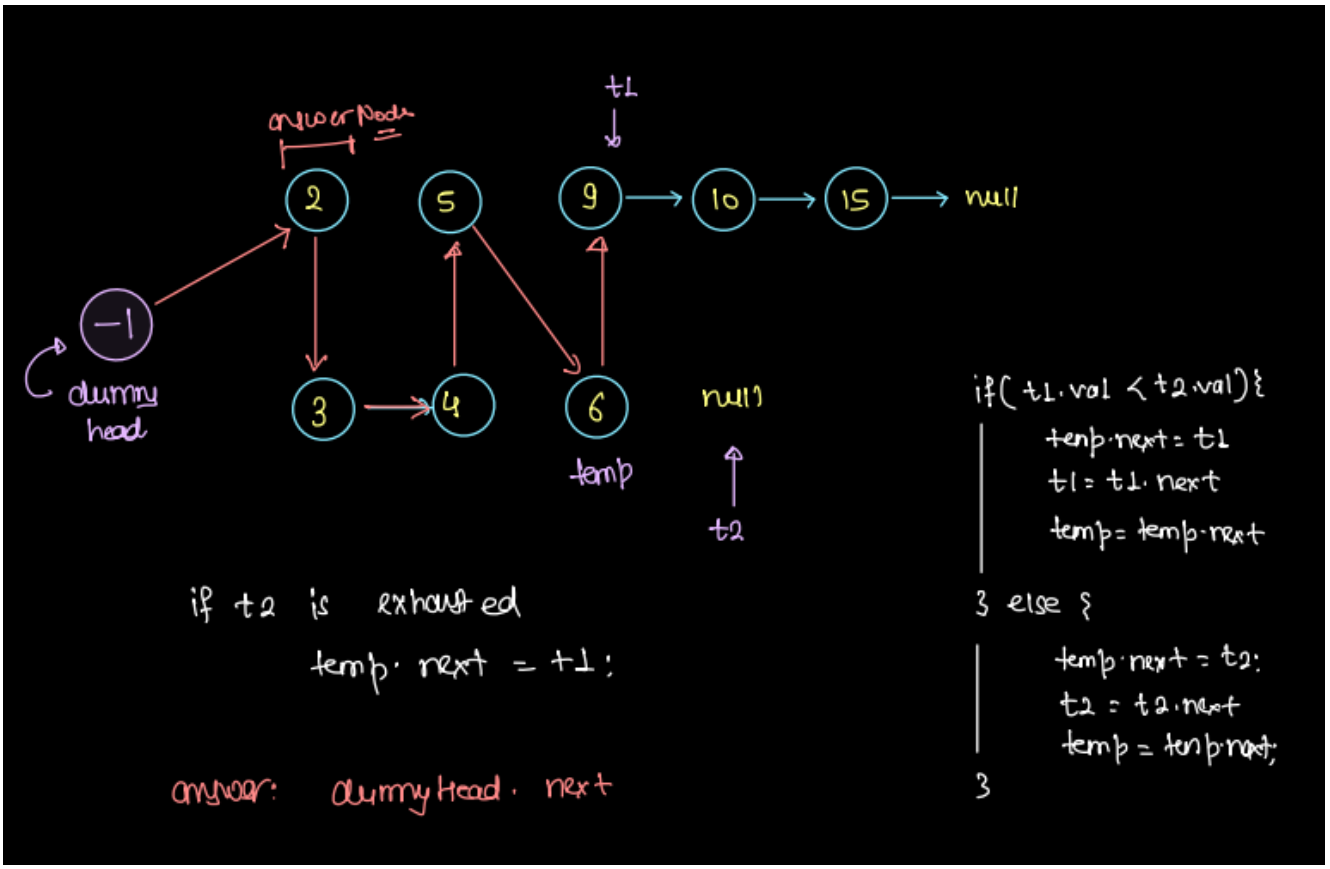
Given two sorted LinkedList, merge and get final sorted list. Note no extra space allowed.
https://www.interviewbit.com/snippet/3277120fb9e10d542ae7/

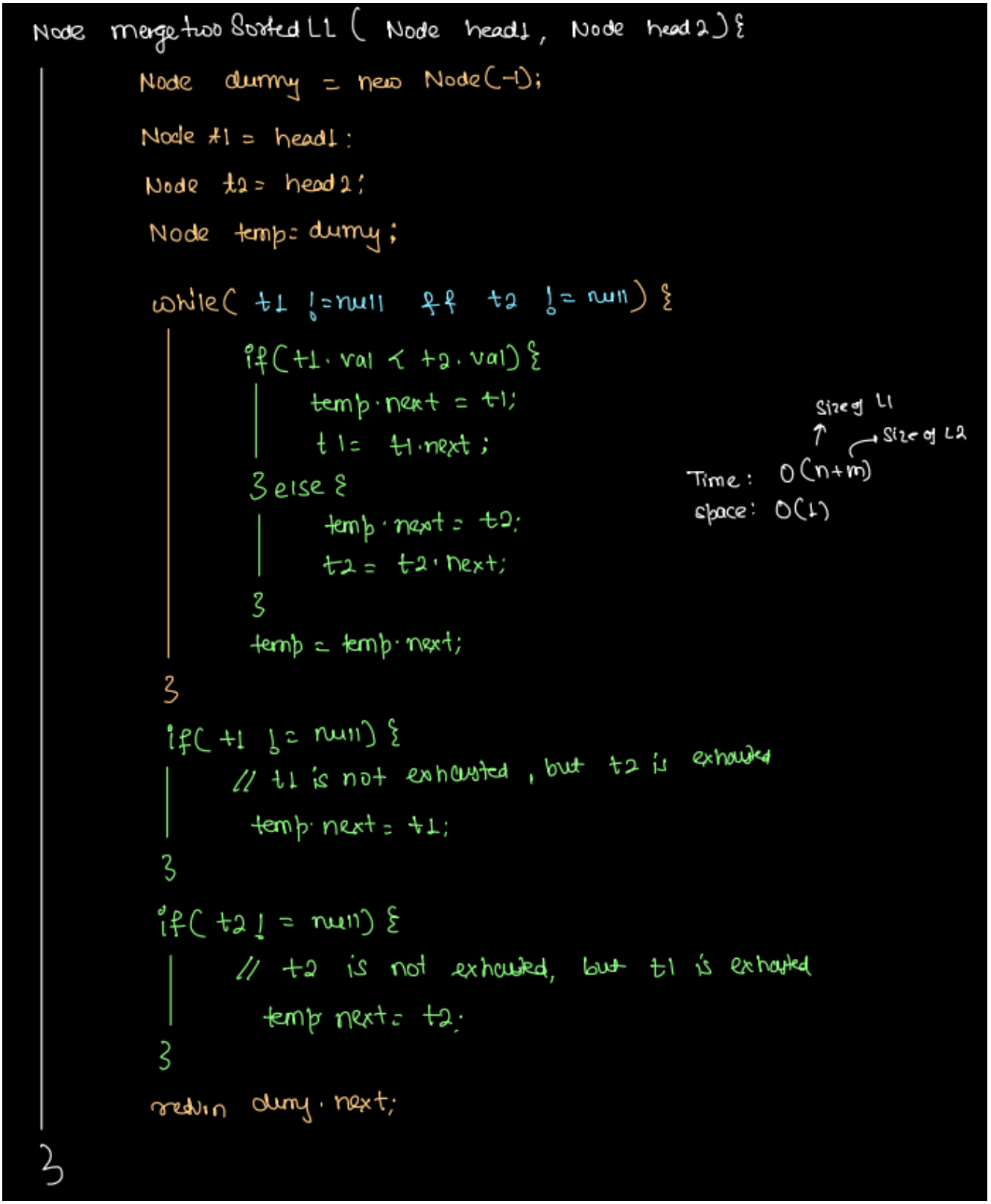
Main Logic ->

To solve this problem, we will utilize a dummy node concept which will avoid lot of conditions, initialized with a value of -1, along with two nodes, t1 and t2, representing the heads of the respective linked lists. We will traverse the linked lists using the while loop with the condition (t1 != null && t2 != null). Inside the loop, we will implement the following logic:

- If t1 is less than t2, assign t1 to dummy.next and update t1 to t1.next.
- If t2 is less than t1, assign t2 to dummy.next and update t2 to t2.next.
- After each assignment, reassign dummy to dummy.next.

After the while loop finishes, we will check one more condition: if t1.next != null, assign temp.next as t1; otherwise, assign temp.next as t2. Finally, we will return dummy.next as the new head of the merged linked list.





Reorder List - Rearranging given LinkedList in following manner L0 -> Ln -> L1 -> Ln-1 -> L2 ....
list 1 => 1-2-3-4-5-6-null
example => 1 => 6 => 2 => 6 => 3 => 4 => null

https://www.interviewbit.com/snippet/b3a051d7b426e55272fa/

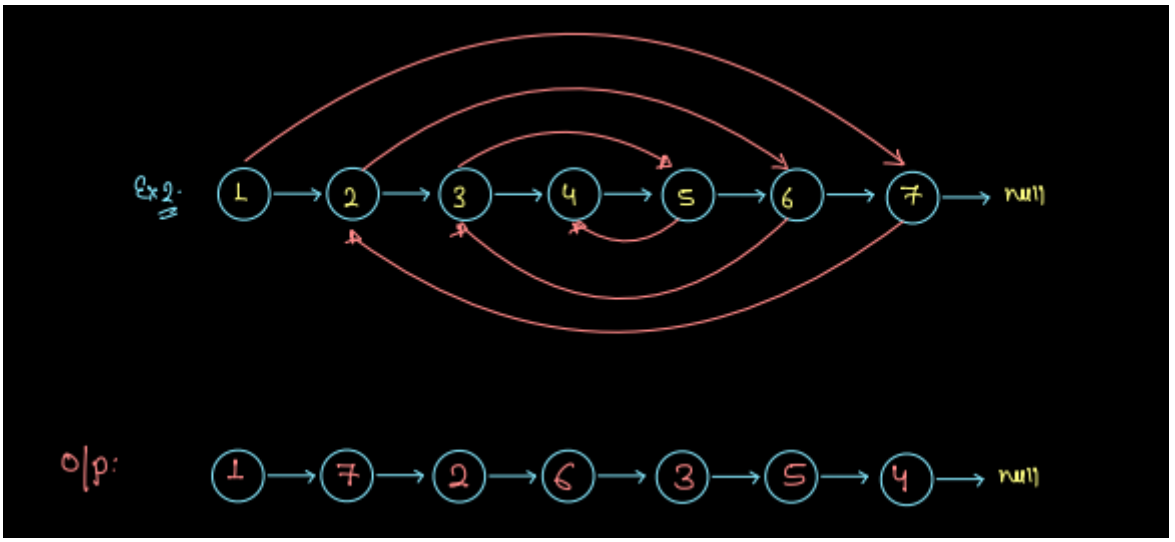
Main Logic =>
Steps
Divide the LinkedList into two parts: Find the middle node of the LinkedList. Create a new node called head2 and assign mid.next to it. Unlink the midpoint from the LinkedList by setting mid.next = null.

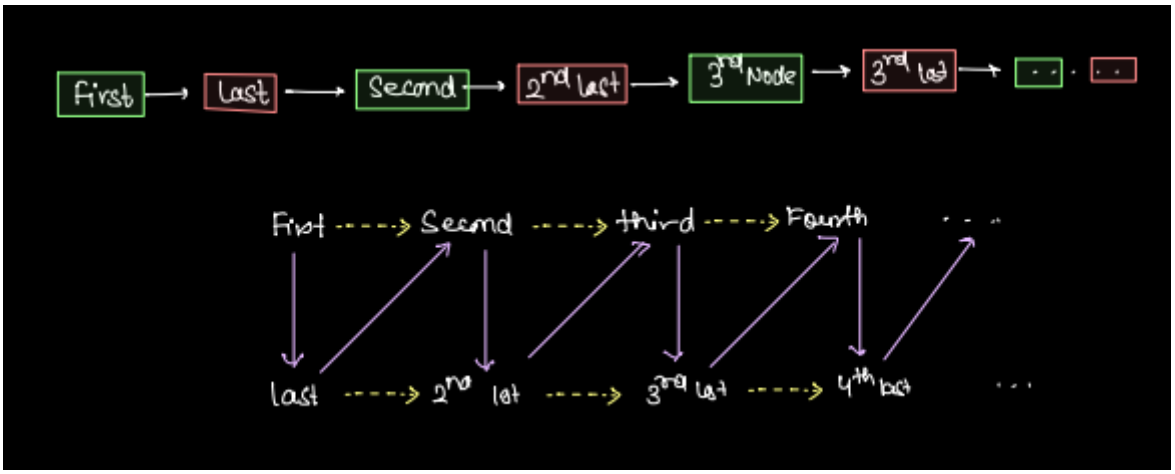
Reverse the second half of the LinkedList.

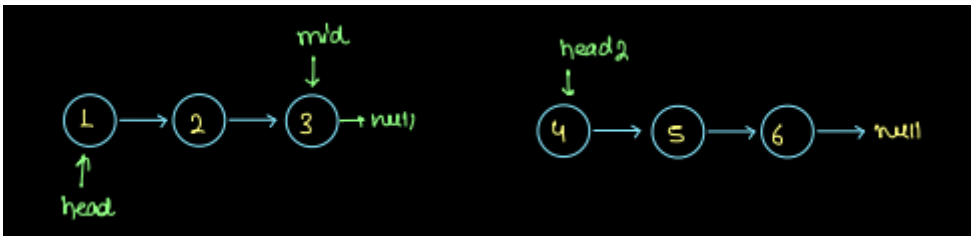
To create the final answer, link the two parts in such a way that they form the reordered list. Start by using the slow and fast pointer logic to find the midpoint and break the LinkedList into two parts. Then,
create the a dummy node initialized with a value of -1, and a temp node initialized with head
enter a while loop with the condition t2 != null.. Inside the loop, update temp.next = t1 and reassign t1 to t1.next. Then, update temp = temp.next. Repeat the same process for t2: reassign dummy.next = t2, t2 = t2.next, and temp = temp.next and after while loop write condition if t1!== null temp.next = t1. Finally, return the head dummy.next as the merged LinkedList.

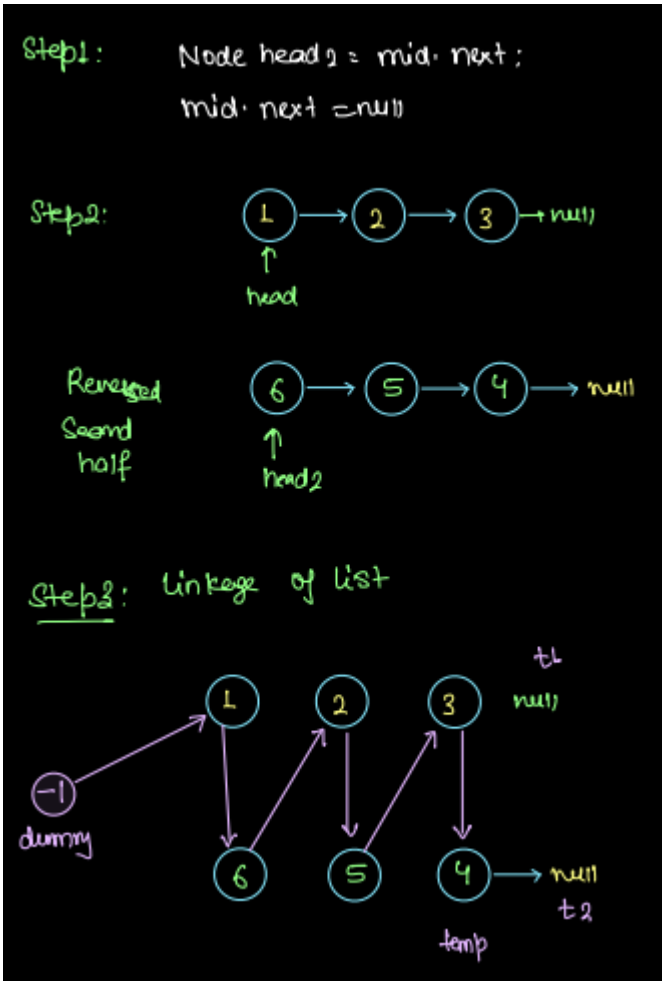
Note: After reordering the list by alternating nodes from `headA` and `headB`, it's possible that the length of `headA` is greater than the length of `headB` if the original list length is odd. In this case, there will be remaining nodes in `headA` that haven't been processed yet.

To handle this situation, the condition `if(headA != null)` checks if there are any remaining nodes in `headA`.









There are two approaches to detect cycles in a linked list: a brute-force approach using a HashSet, and an optimized approach using the Floyd cycle detection algorithm.

Brute-force approach:

- Create an empty HashSet.
- Traverse the linked list, and for each node:
    - Check if the address of the current node exists in the HashSet.

- If it does, then a cycle is present in the linked list.
- If it doesn't, add the address of the current node to the HashSet.

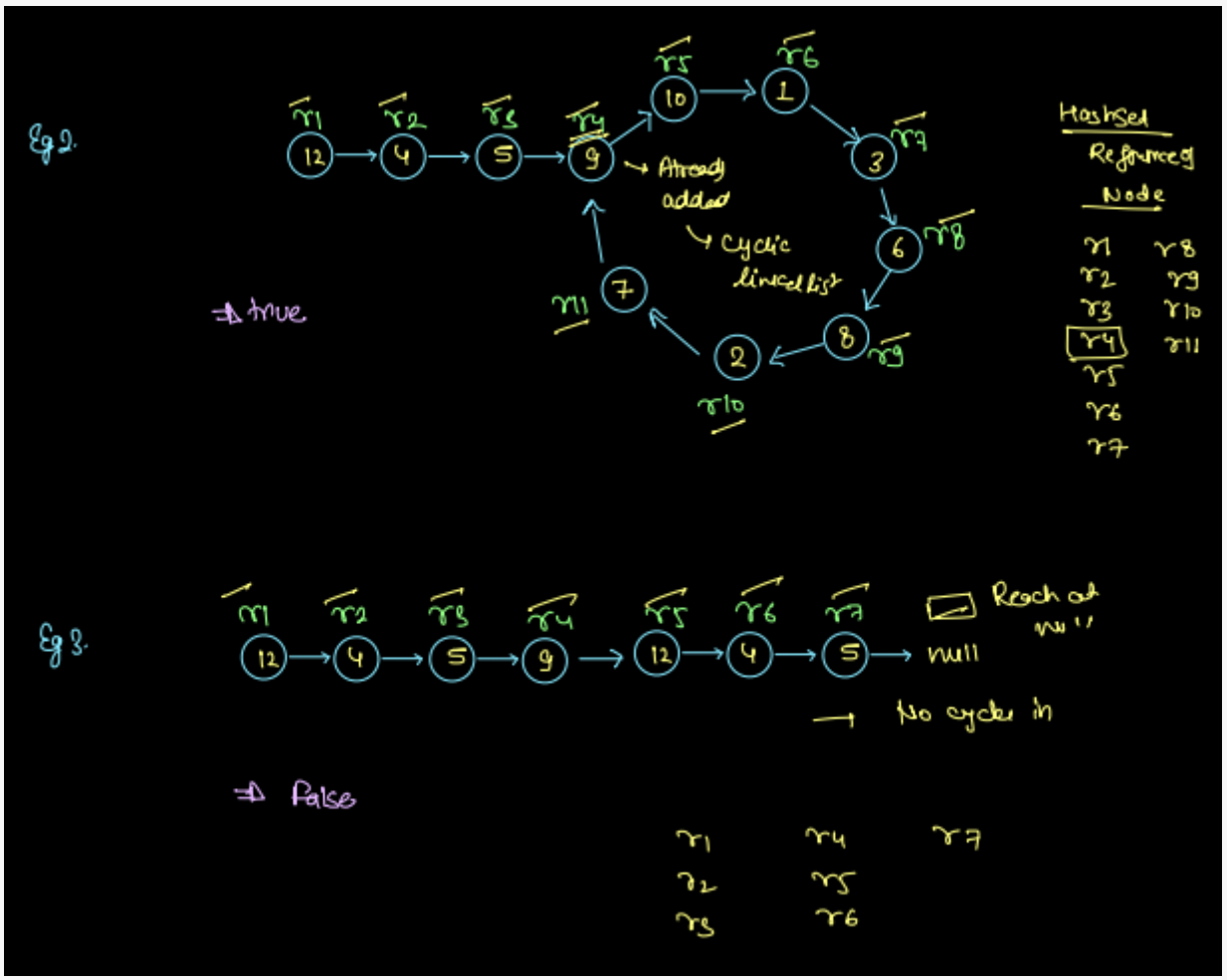
- If the traversal completes without finding a cycle, then no cycle is present.

This approach has a time complexity of O(n) and a space complexity of O(n) since we need to store the addresses of the visited nodes in the HashSet.

Optimized approach using Floyd cycle detection algorithm:

- Create two pointers, slow and fast, and initialize them with the head of the linked list.
- Initialize a boolean variable, isCycle, to false.
- Enter a while loop with the condition (fast.next != null && fast.next.next != null).
- Inside the loop, update slow as slow = slow.next and fast as fast.next.next.
- Check if fast is equal to slow.
  - If they are equal, it means a cycle is present in the linked list.
  - Set isCycle to true and break the loop.
- After breaking the loop, return the value of isCycle.

This approach detects cycles by using two pointers moving at different speeds through the linked list. It has a time complexity of O(n) and a space complexity of O(1) since no additional data structure is used to store node addresses.



Idea 1: Using hashSet to create, if reference in set is repeating or not.

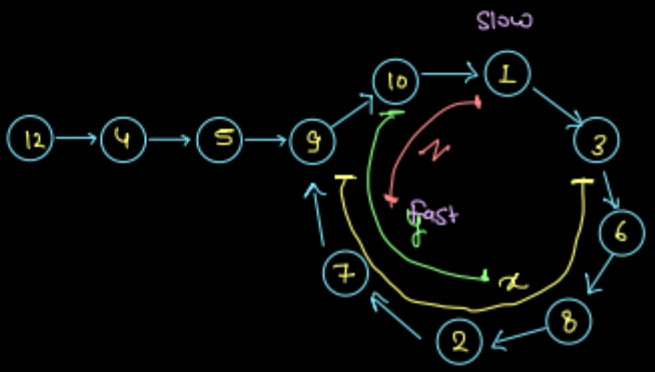
```
HashSet<Node> set = new HashSet<>();
```

T.C: O(n)

S.C: O(n)

Idea 2: without space

Using → Floyd cycle Detection Algorithm



d1 d2 d3
x > y > z

distance b/w Slow & fast is decreasing and after some time they meet.

once slow and fast are started

slow by 1 step ⇒ slow = slow.next

fast by 2 step ⇒ fast = fast.next.next

→ once If slow & fast both are inside of the cycle the distance b/w fast and slow is decreasing and after some iteration, eventually it will become 0.

Slow & fast meet at same point.

```
boolean isCyclic(Node head){
    Node slow = head;
    Node fast = head;
    boolean isCycle = false;
    while(fast.next != null && fast.next.next != null){
        slow = slow.next
        fast = fast.next.next
        if(slow == fast){
            isCycle = true;
            break;
        }
    }
    return isCycle;
}
```

Starting point of linkedlist

Main logic ->

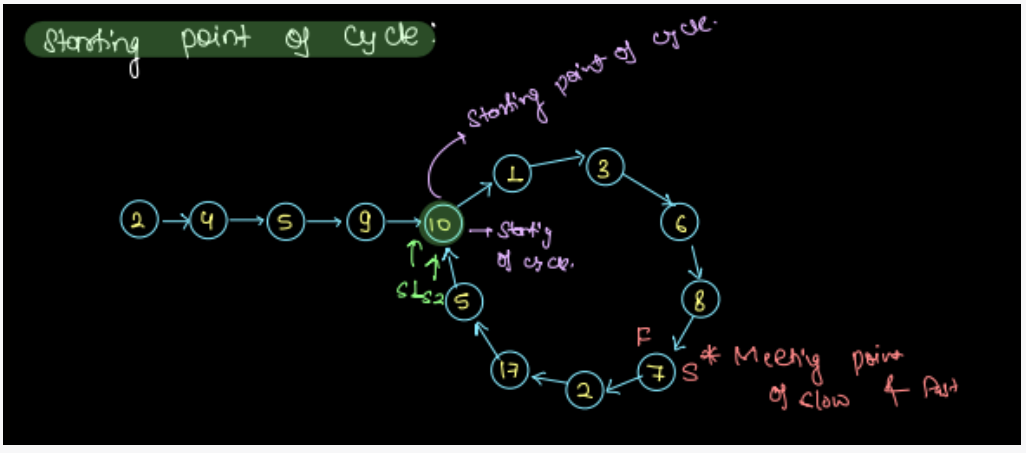
We will solve the problem using two pointers, fast and slow. Initialize two nodes: slow, and fast, all assigned to the head of the linked list. Also, declare a boolean variable, isCycle.

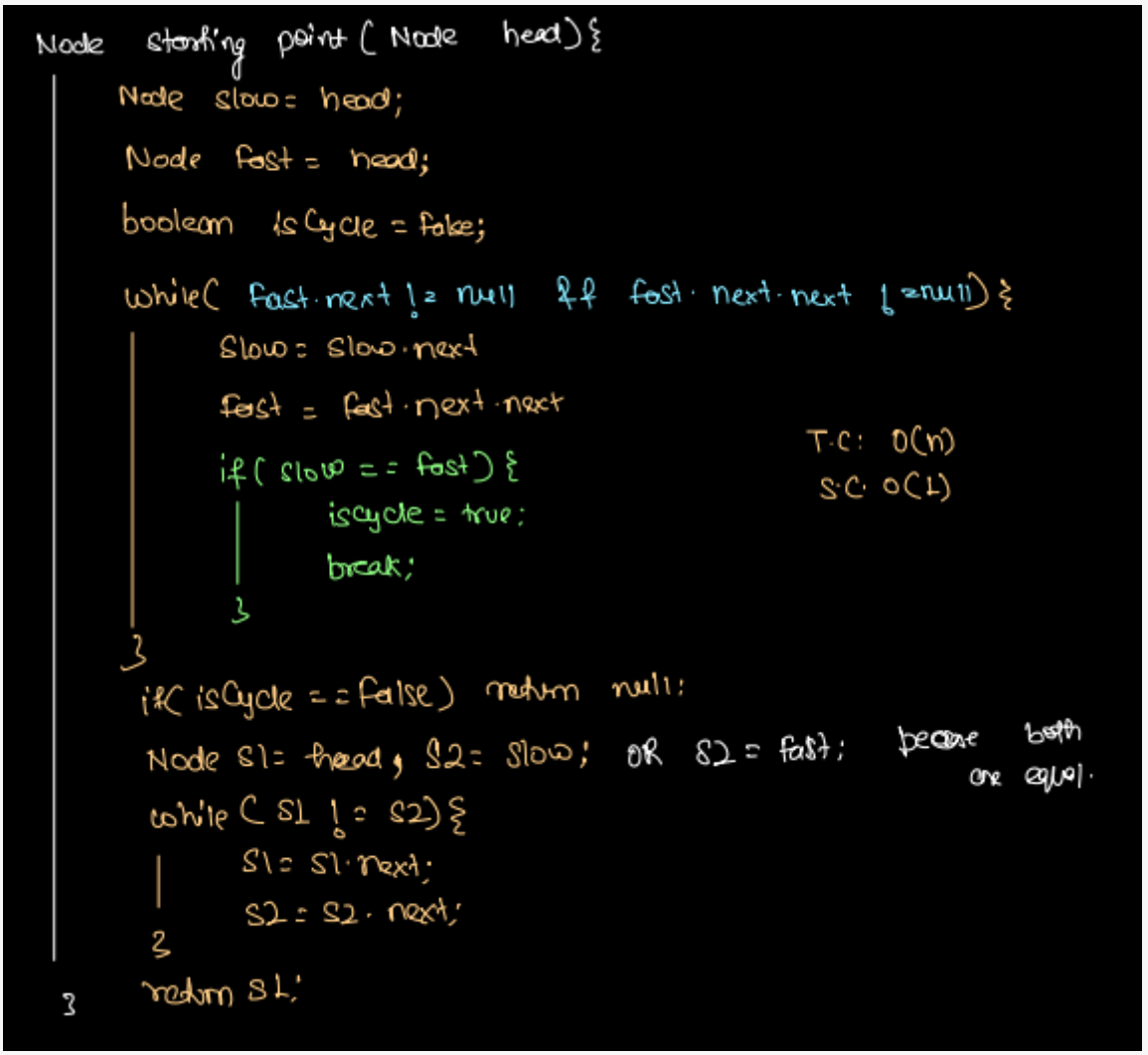
Iterate through a while loop with the condition (fast.next != null && fast.next.next != null). Inside the loop, update both slow and fast: slow = slow.next and fast = fast.next.next. If slow is equal to fast, set isCycle to true and break the loop.

After breaking the loop,
Write a condition if(isCycle == false) return null;

create two additional nodes, s1 and s2, with s1 assigned to the head and s2 assigned to either slow or fast. Enter another while loop and check if s1 is not equal to s2. Inside this loop, update s1 = s1.next and s2 = s2.next.

Once the second while loop ends, return either slow or fast as the answer, which represents the starting point of the linked list cycle.





Remove Cycle from LinkedList and return the head of LinkedList

Main logic ->

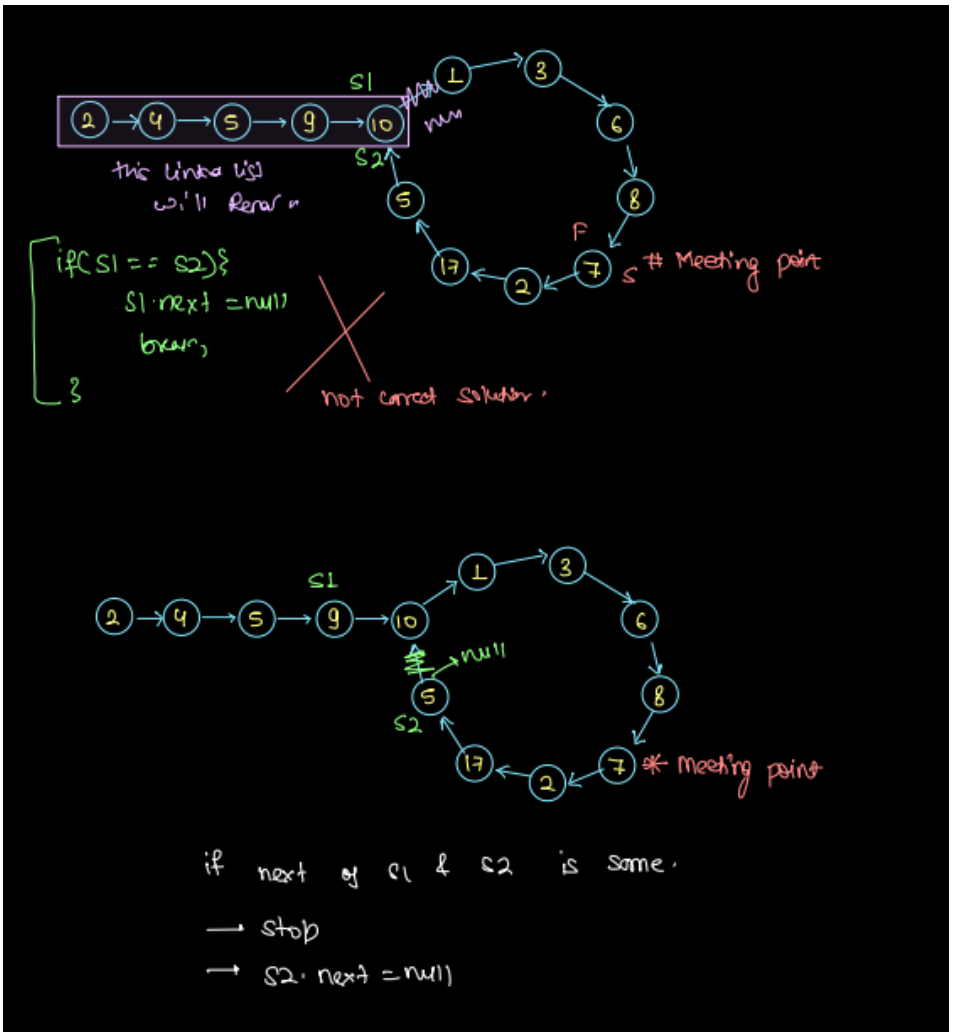
We will solve the problem using two pointers, fast and slow. Initialize two nodes: slow, and fast, all assigned to the head of the linked list. Also, declare a boolean variable, isCycle.

Iterate through a while loop with the condition (fast.next != null && fast.next.next != null). Inside the loop, update both slow and fast: slow = slow.next and fast = fast.next.next. If slow is equal to fast, set isCycle to true and break the loop.

After breaking the loop,
Write a condition if(isCycle == false) return null;

create two additional nodes, s1 and s2, with s1 assigned to the head and s2 assigned to either slow or fast. Enter another while loop and check if s1.next is not equal to s2.next. Inside this loop, update s1 = s1.next and s2 = s2.next.

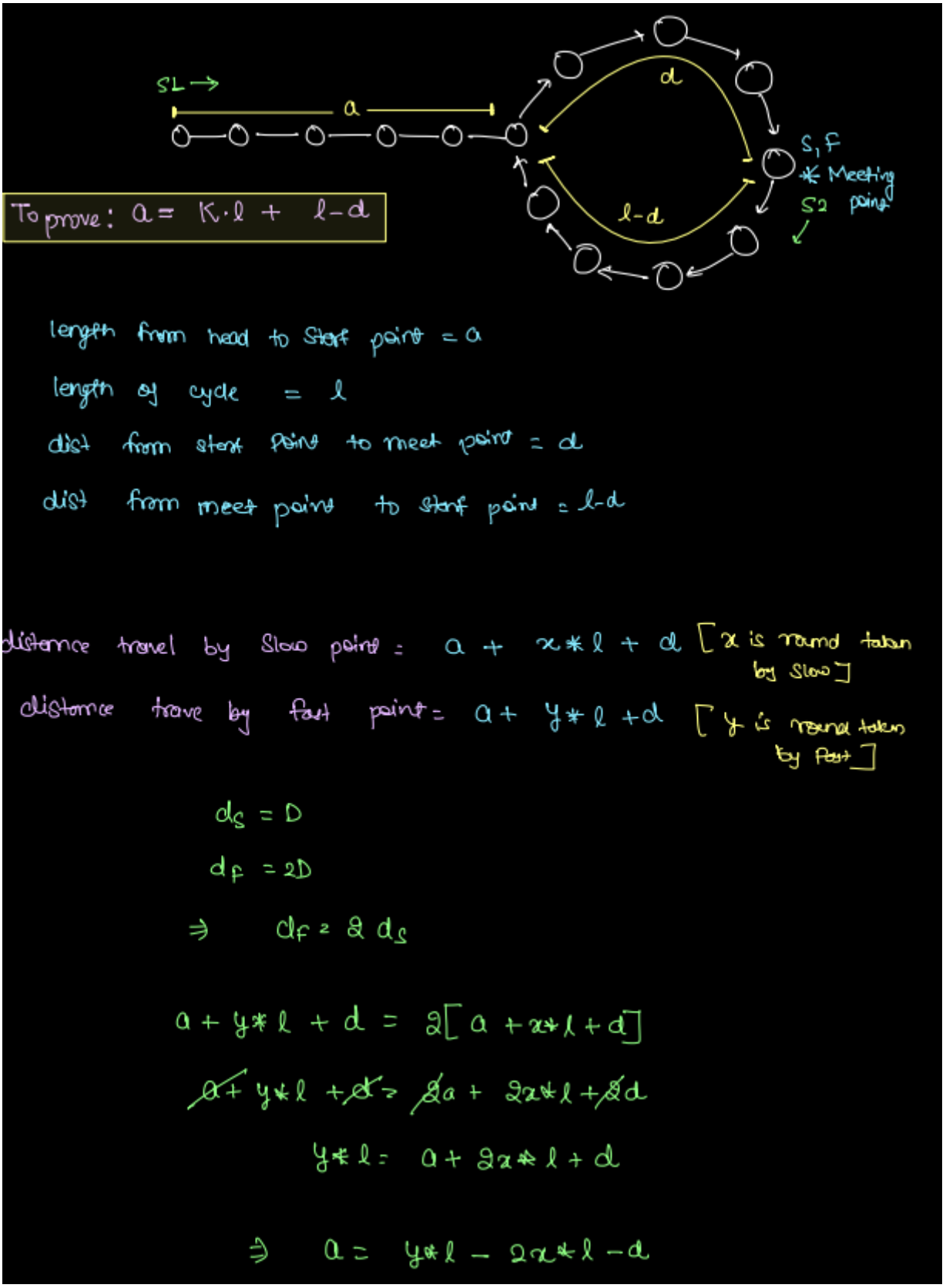
Once the second while loop ends, write null to s2.next = null and return **head**
https://www.interviewbit.com/snippet/52f299ff01f500d9744b/

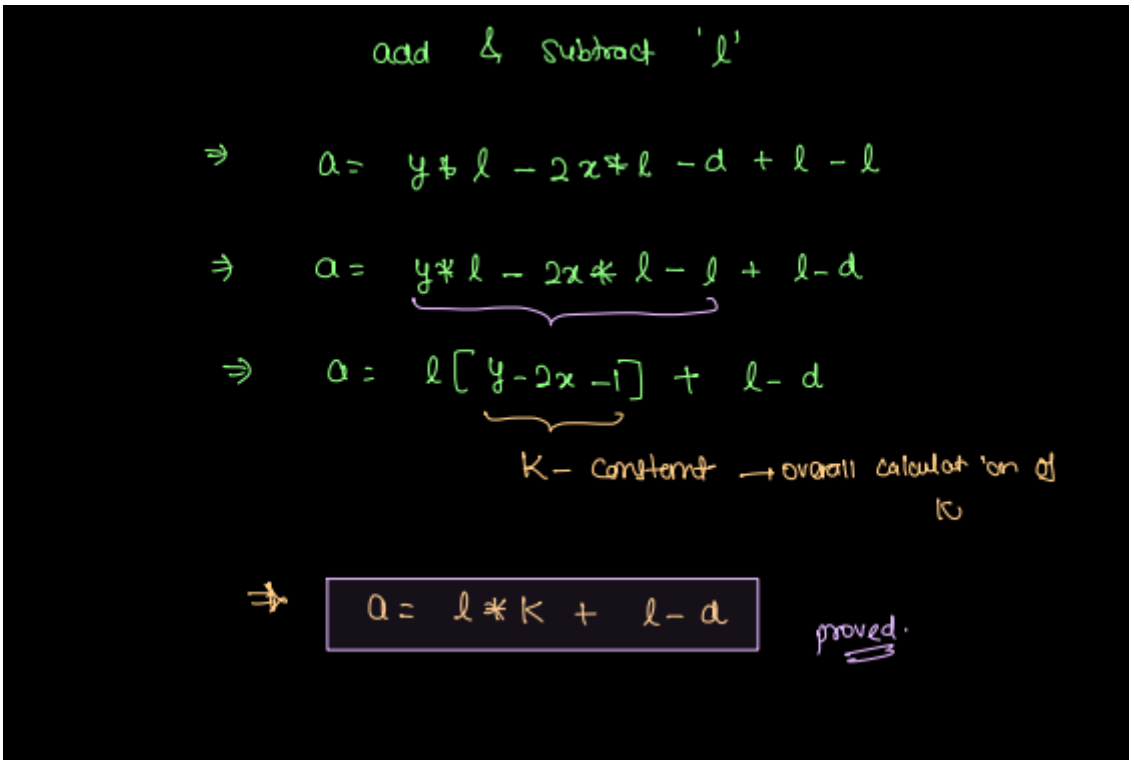



```
Node removeCycle (Node head) {
    Node slow = head;
    Node fast = head;
    boolean isCycle = false;
    while( fast.next != null && fast.next.next != null) {
        slow = slow.next
        fast = fast.next.next
        if ( slow == fast) {
            isCycle = true;
            break;
        }
    }
    if( isCycle == false) return head;
    Node S1 = head, S2 = slow;
    while (S1.next != S2.next) {
        S1 = S1.next;
        S2 = S2.next;
    }
    S2.next = null;
    return head;
}
```

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

Proof of starting point of cycle





DSA: Linked List 3 - 8 July 2023

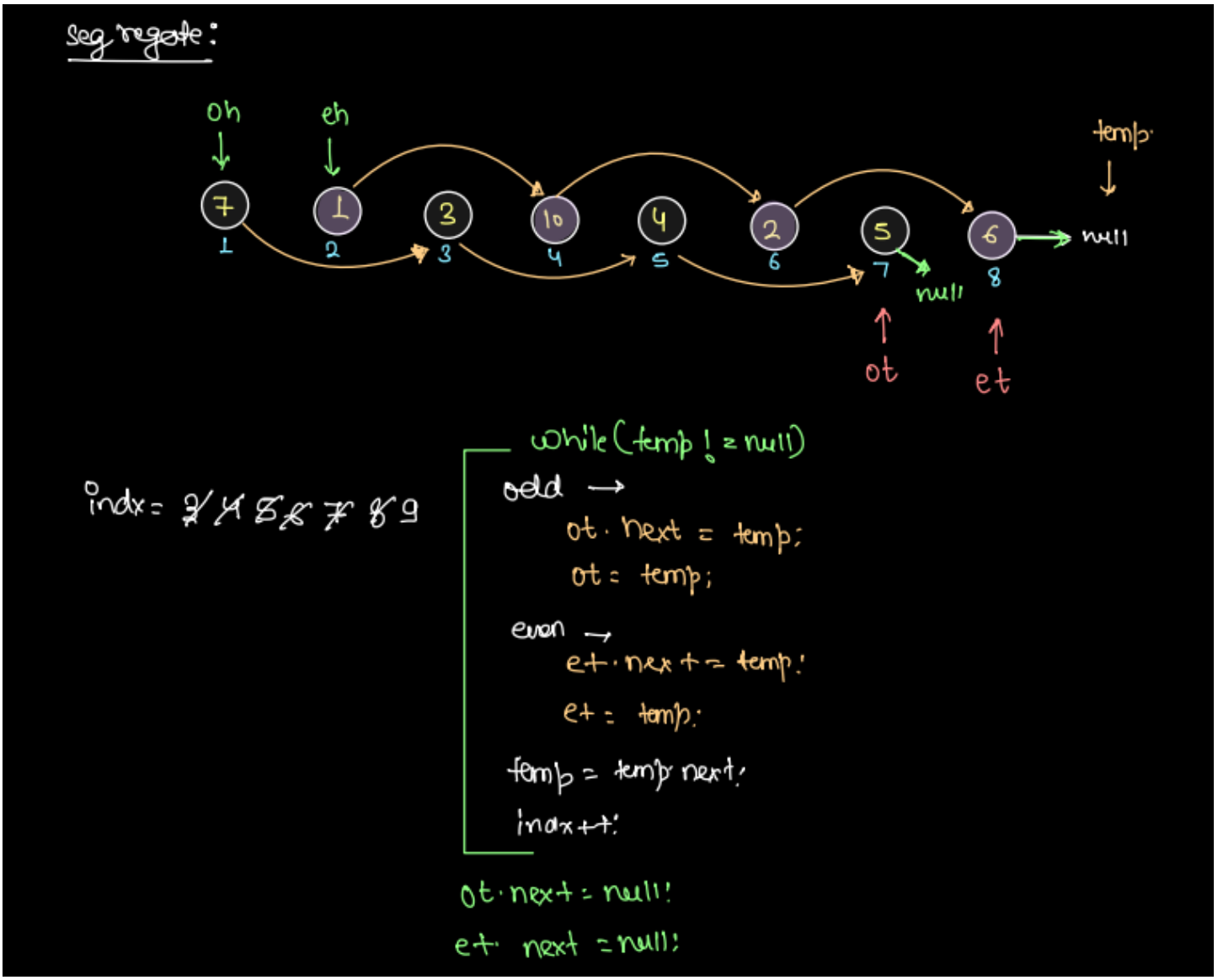
Basic of DLL
Remove Node from DLL
Add Before Tail
Introduction of LRU

DSA: Contest 5 discussion - 11 July 2023

Even Reverse
Given a linked list A, reverse the order of all nodes at even positions. (1 based positioning)
Main logic ->
There are 3 steps we will follow to solve this
1. Segregate List even and odd.
2. Reverse even list using function
3. Linkage both list starting from odd head.

Segregate list ->
- We will create two node oddHead and oddTale and assign it head.
- We will create two more variable evenHead and evenTale and assign it head.next.
- Now we will create one more node called temp and assign it to head.next.next. We will create 1 variable to tack index and assign it to 3red index because on first index oddTale and on 2 nd index evenTale thatsby we will start itarte index from 3.
- We will itarte over while loop condition eventTale != null and oddTale != null and inside while loop we will write a condition if index%2 != 0 then we will write assign oddTale.next = temp and oddTale = temp by doing we will move oddTale to next index. in even condition we will write evenTale.next = temp and oddTale = temp.
- After condition we will write temp = temp.next and index++;

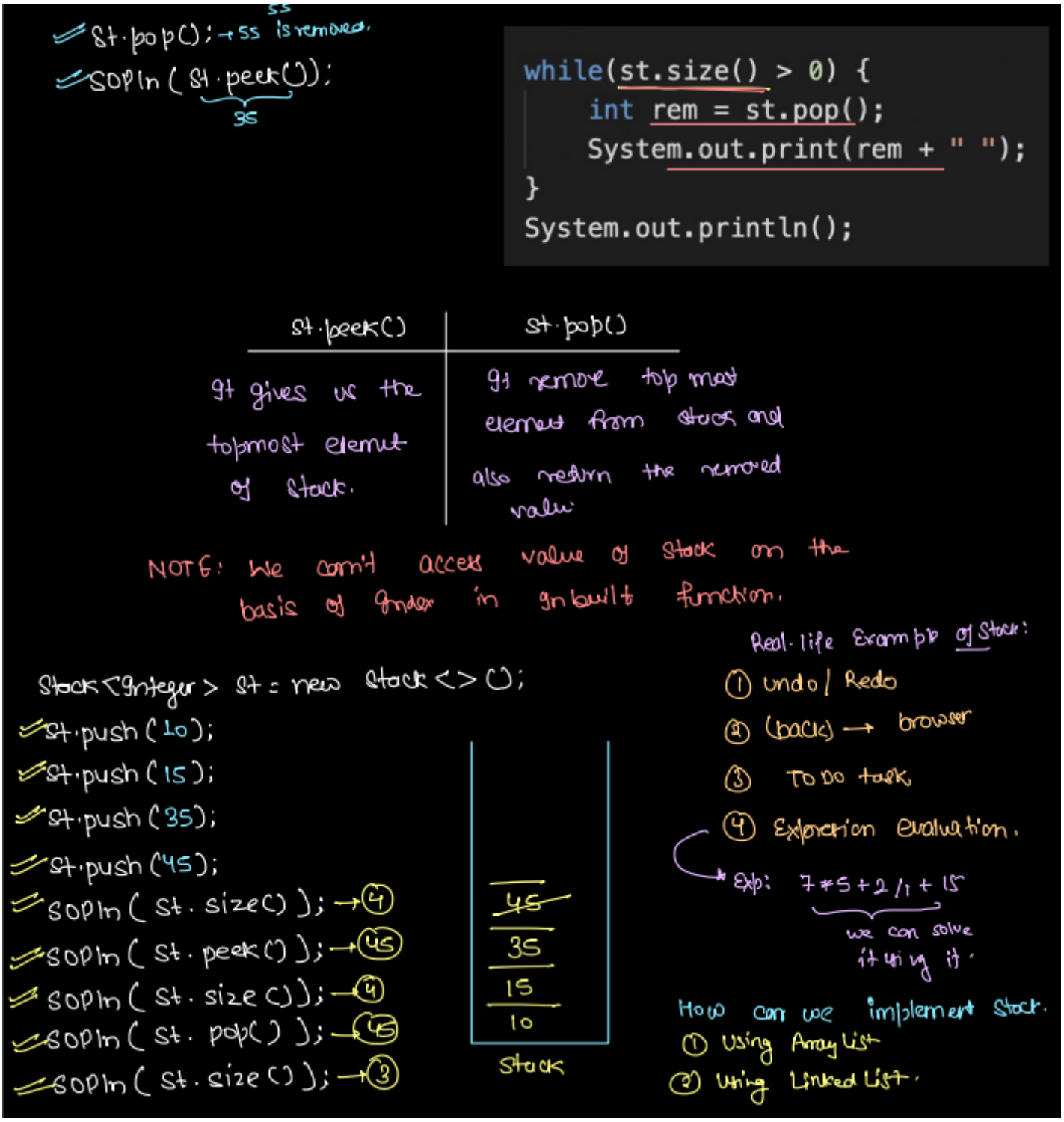
- After loop we will assign null to oddTale and evenTale.

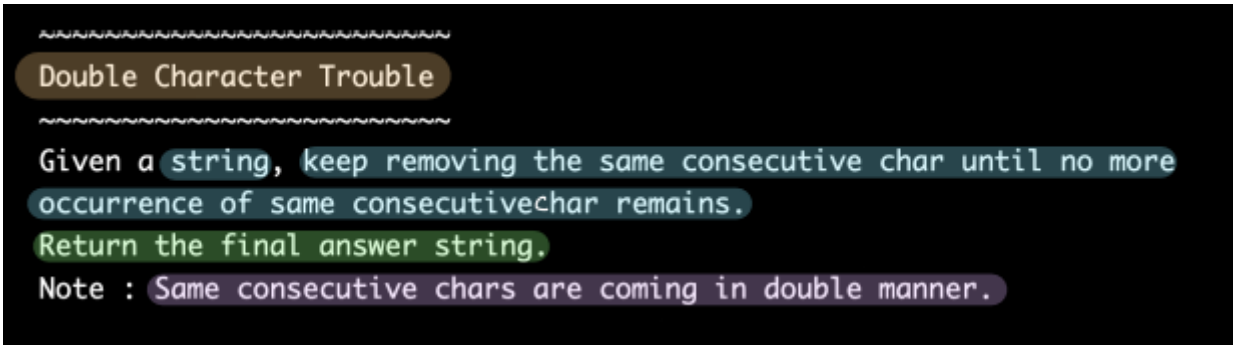


Reverse Even list ->
We will create a function to reverse string and pass evenHead to it.

DSA: Stack1 - 13 July 2023

Stack Introduction ->
Stack bahviour is LIFO symantic. **Last In First Out (LIFO)**.
Fucntion avaialle in stack
push() - Insert element in stack
pop() - For removal of top element.
peek() - for access of top element
size() - It will return the no of element available.
We can not access using index.

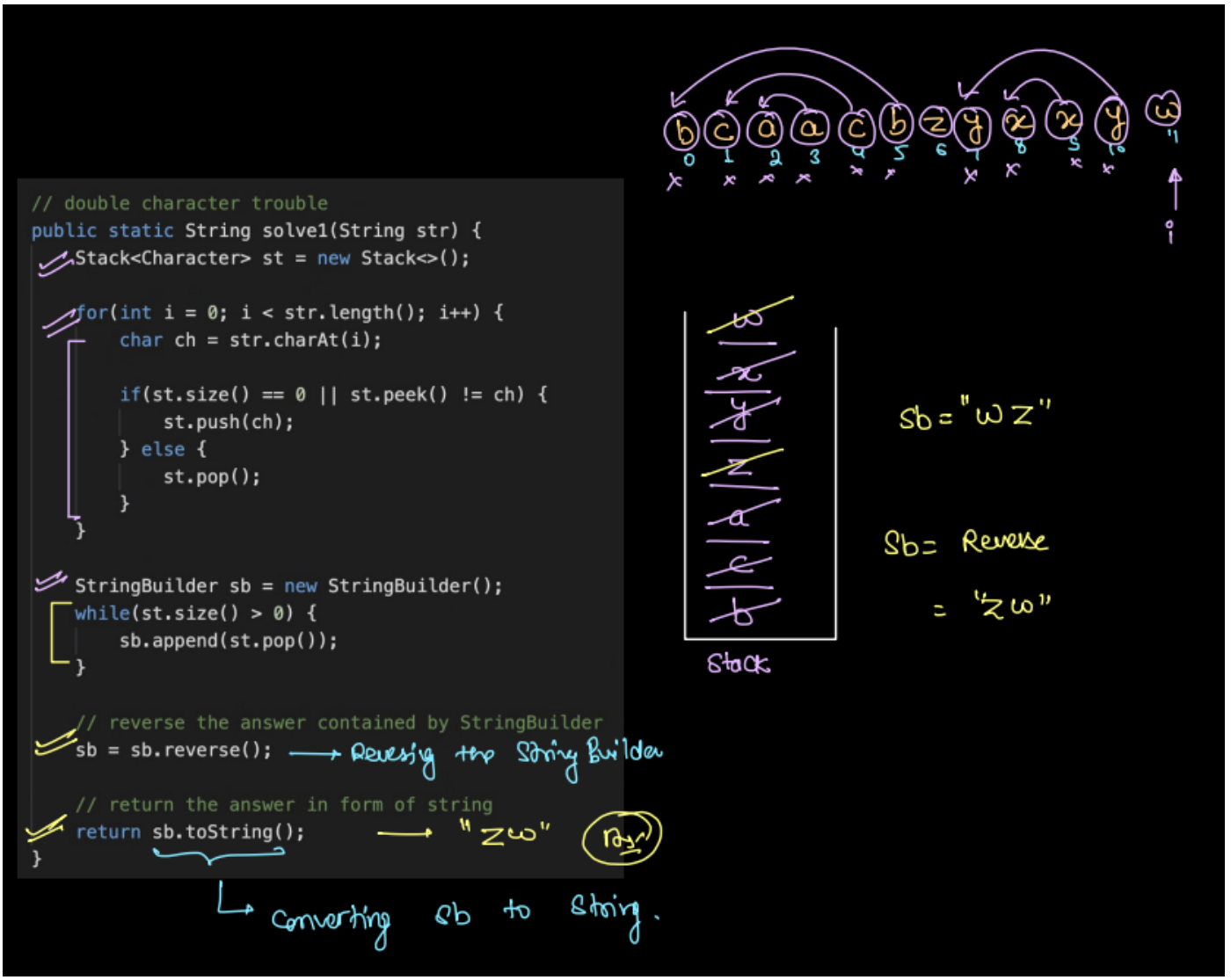


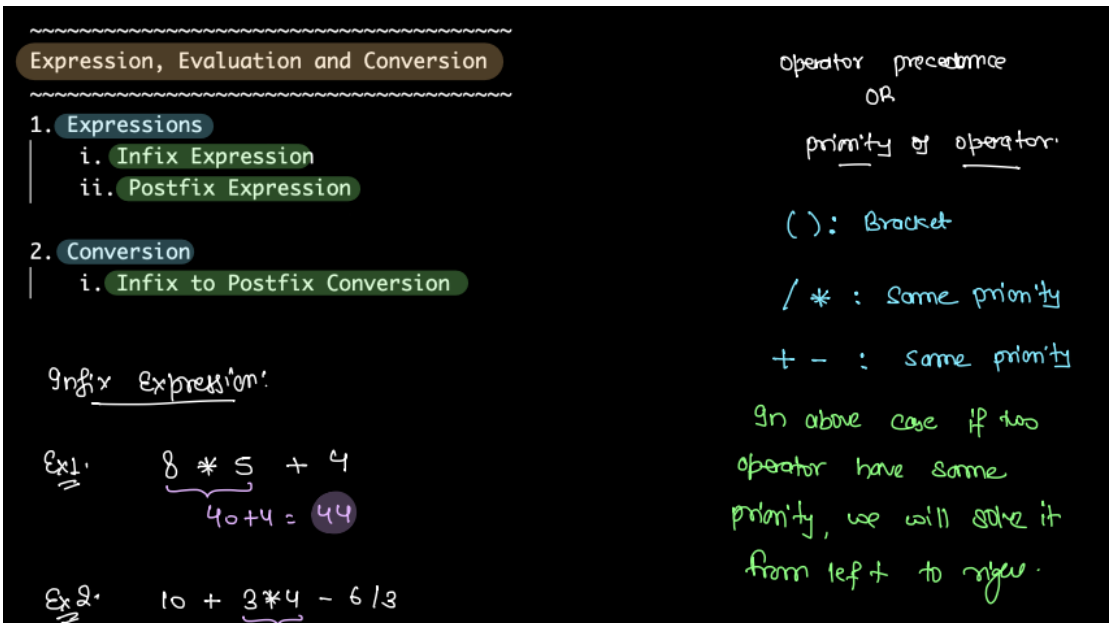


Main logic

First, we will establish a stack and iterate over a for loop based on the length of a string. Inside the loop, we'll add a condition: if the stack's size is 0 and the top element of the stack is not equal to the current character (ch), we'll push the character onto the stack; otherwise, we'll pop an element from the stack.

Next, we'll create a StringBuilder object and enter a while loop. As long as the stack's size is greater than 0, we'll append the popped elements from the stack to the StringBuilder. After the loop, we'll reverse the StringBuilder and convert it to a string using the toString method. Finally, we'll return the resulting string.





How a machine will perform any kind of calculation ?

Initial when we write any expression, it is in infix form. (B D M A S)

→ Compiler convert that infix expression into post fix expression.

→ Solve postfix expression and return answer.

What is post fix Expression:

Infix Expression ⟶ Postfix Expression.

a + b	⟶	a b +
a - b	⟶	a b -
a * b	⟶	a b *
a / b	⟶	a b /

Infix → a + b

Expression tree

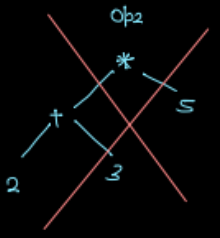
In one element → a + b → Infix Exp

postfix → a b +

Infix expression can be denoted in form of Expression tree :

Ex. → 2 + 3 * 5

Priority is maged.



Priority of operator is violated.

Correct option →

Infix: 2 + 3 * 5

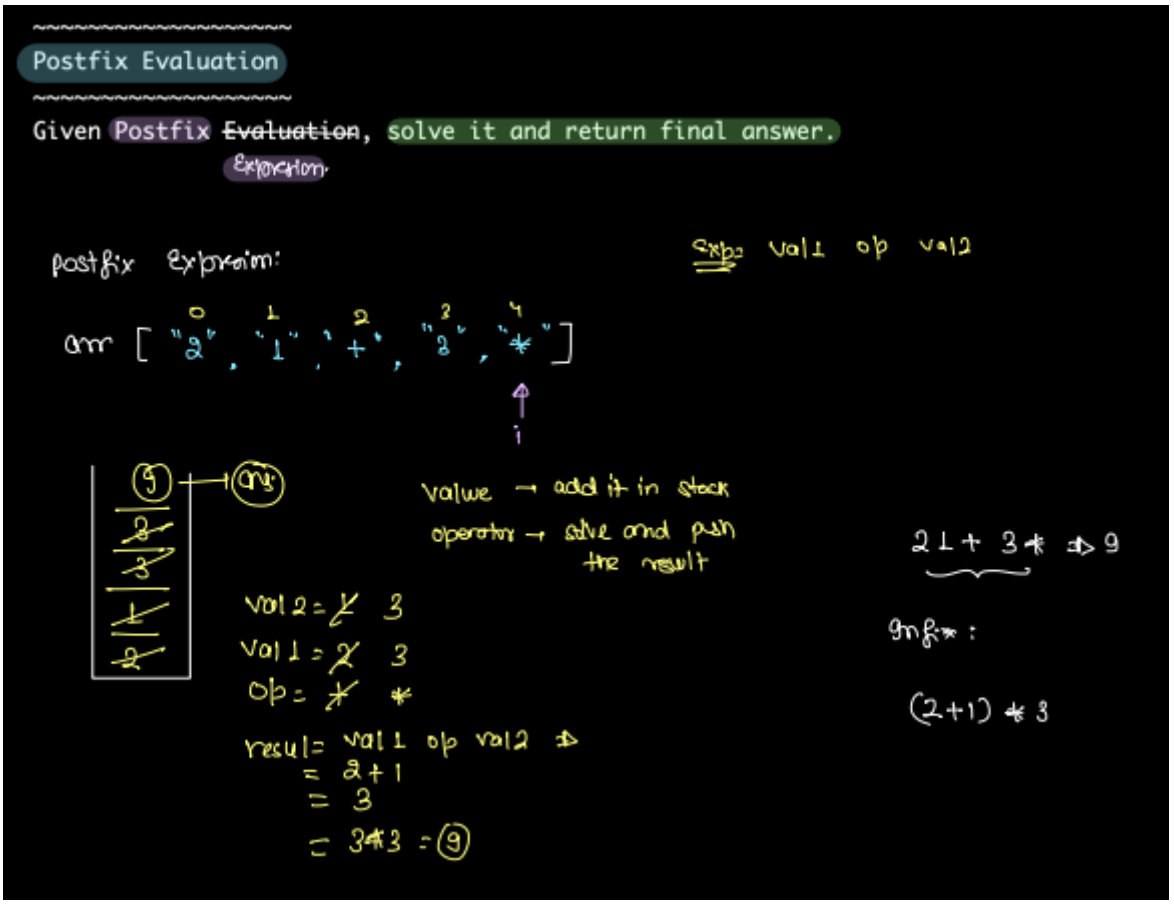
postfix: 2 3 5 * +

Advantage of postfix Expression:

① There is no bracket is postfix expression.

② In postfix exp, operator precedence already calculated and operators are arranged in priority order

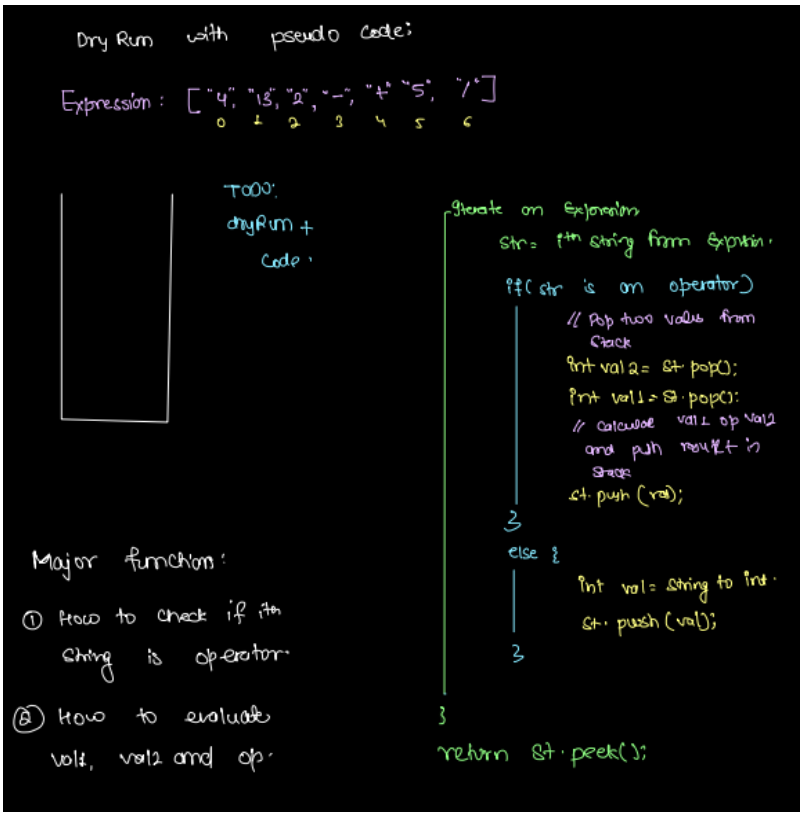
③ Postfix expression is used by compiler or machine to solve of Expression because of easy nature. Break [8:50 - 9:00 AM]



Main logic -> Solve Postfix

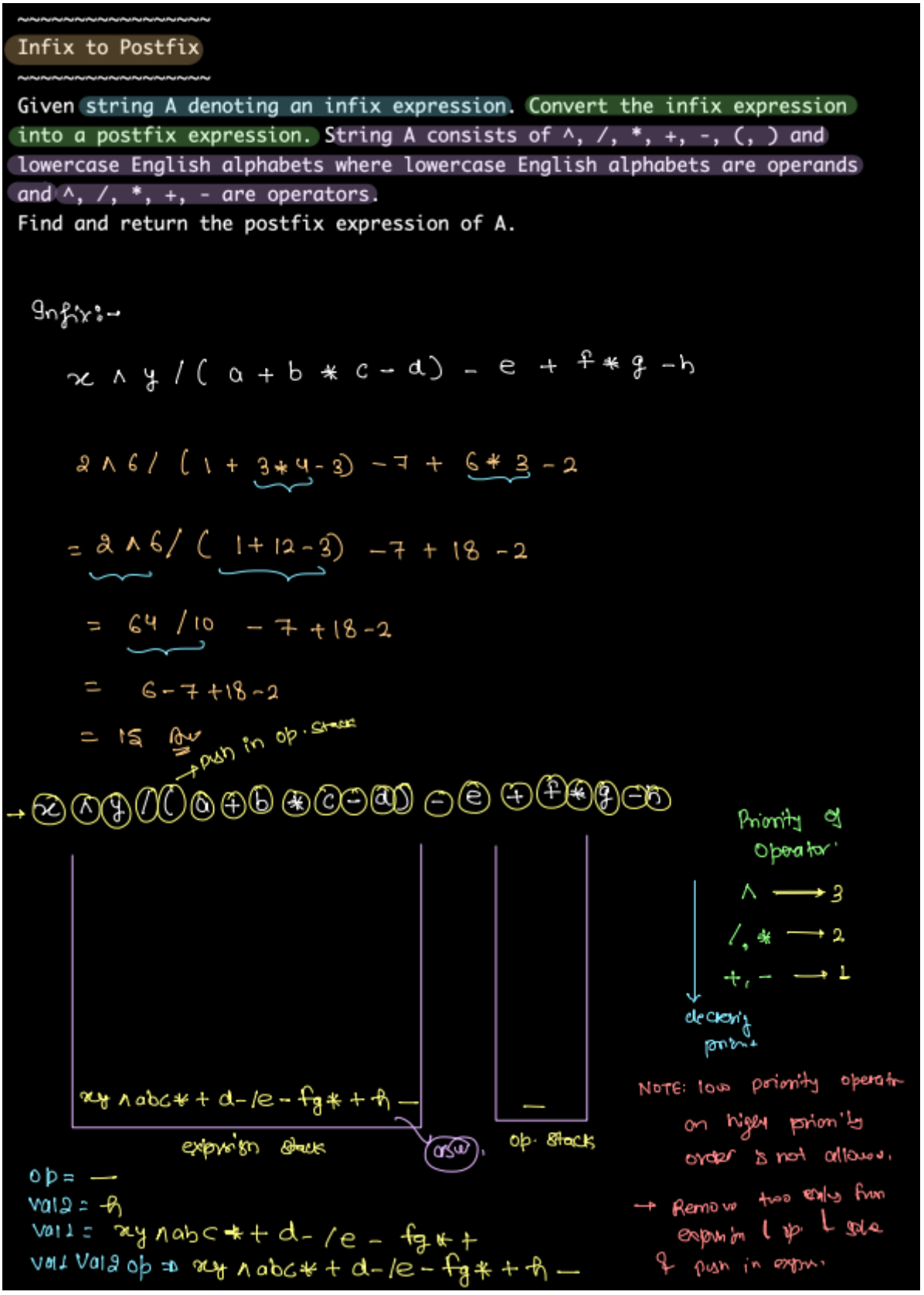
We will iterate using a while loop until the length of the array. If the current element is an operator, we will retrieve the top item from the stack as value2. Then, we will get the next item from the array as value1. We will solve the expression value2 operator value1 and push the result back into the stack. else If the current element is an operand string convert it to integer and, we will add it to the stack.

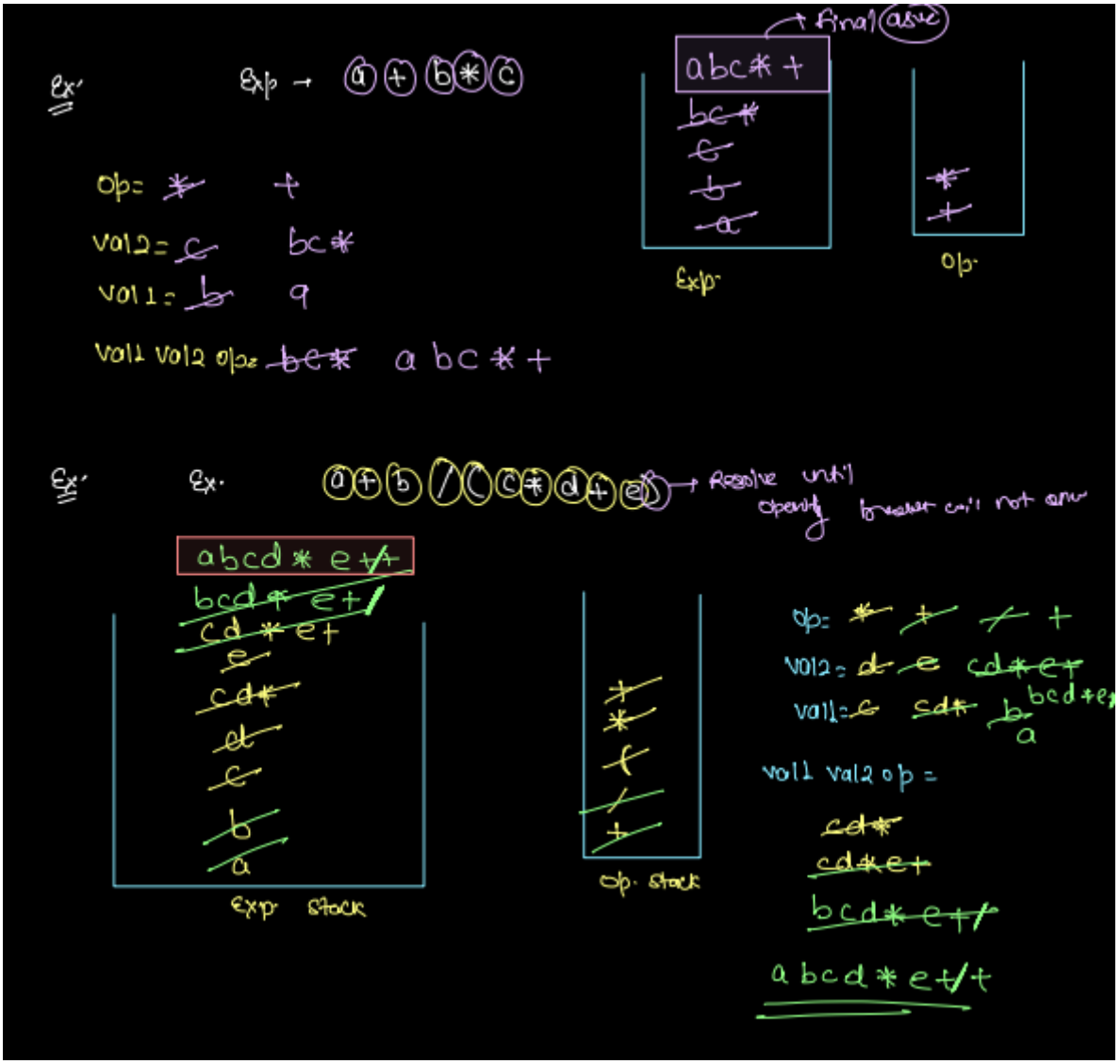
at end after iteration we will return stack.peek.



Infix to Postfix - >
https://www.interviewbit.com/snippet/478f49b6e6439476e461/
NOTE:

- **^ has the highest precedence.**
- **/ and * have equal precedence but greater than + and -.**
- **+ and - have equal precedence and lowest precedence among given operators.**





Assignment =>

Q1. Evaluate Expression

https://www.interviewbit.com/snippet/9b11356df078a2b27fe8/

Q2. Balanced Paranthesis

Given an expression string A, examine whether the pairs and the orders of “{“,”}”, ”(“,”)”, ”[“,”]” are correct in A.

Refer to the examples for more clarity.

https://www.interviewbit.com/snippet/e5dca27e27d26f1ad621/

Main logic ->

- Create a stack named st.
- Define a custom function isMatchingPair that checks for all possible pairs: start == '(' && end == ')' || start == '{' && end == '}' || start == '[' && end == ']'. If the pair matches, return true; otherwise, return false.

Iterate over the characters of the str string until string.length.

- Retrieve the character using charAt and store it in ch.
- If ch is '}', ']', or ')', enter the following condition; otherwise, push ch onto the stack.
- Inside the condition, check if the stack is empty; if yes, return 0.
- Check one more condition using the isMatchingPair custom function and pass the popped item and ch to this function. If it is not matching, return 0.

- At the end, write the condition and return st.isEmpty() ? 1 : 0.

Q3. Double Character Trouble ->
https://www.interviewbit.com/snippet/5f62286d3b2f96310802/

Main logic ->
Check if the length of the string is less than or equal to 1. If so, return the string as it is.

Create an empty stack of characters.
Iterate over the string using a for loop and convert it to a character array using str.toCharArray().

- Check if the stack is not empty and the current character is equal to the top element of the stack. If true, pop the top element from the stack; otherwise, push the current character onto the stack.

After iterating through the string, create a StringBuilder object.
Start a while loop and continue until the stack is empty. In each iteration, pop an element from the stack and append it to the StringBuilder.
Once the loop completes, reverse the StringBuilder using the reverse() method.
Convert the StringBuilder to a string using the toString() method and return it as the answer.

DSA: Stack2 - 15 July 2023

1. Design Min Stack
2. Nearest Smaller in left [value Based]
3. Nearest Smaller in left [Index Based]
4. Nearest Smaller in right [value + Index Based]
5. Largest Area Histogram

1. **Design a stack that supports push, pop, top and retrive the minimum element in constant time.**
https://www.interviewbit.com/snippet/d7bb9b7ea023a79ca8b7/

push(x) => Push element x onto stack
pop() => Removes the element on top
top() => Get the top element.
getMin() => Retrive the minimum element in the stack.

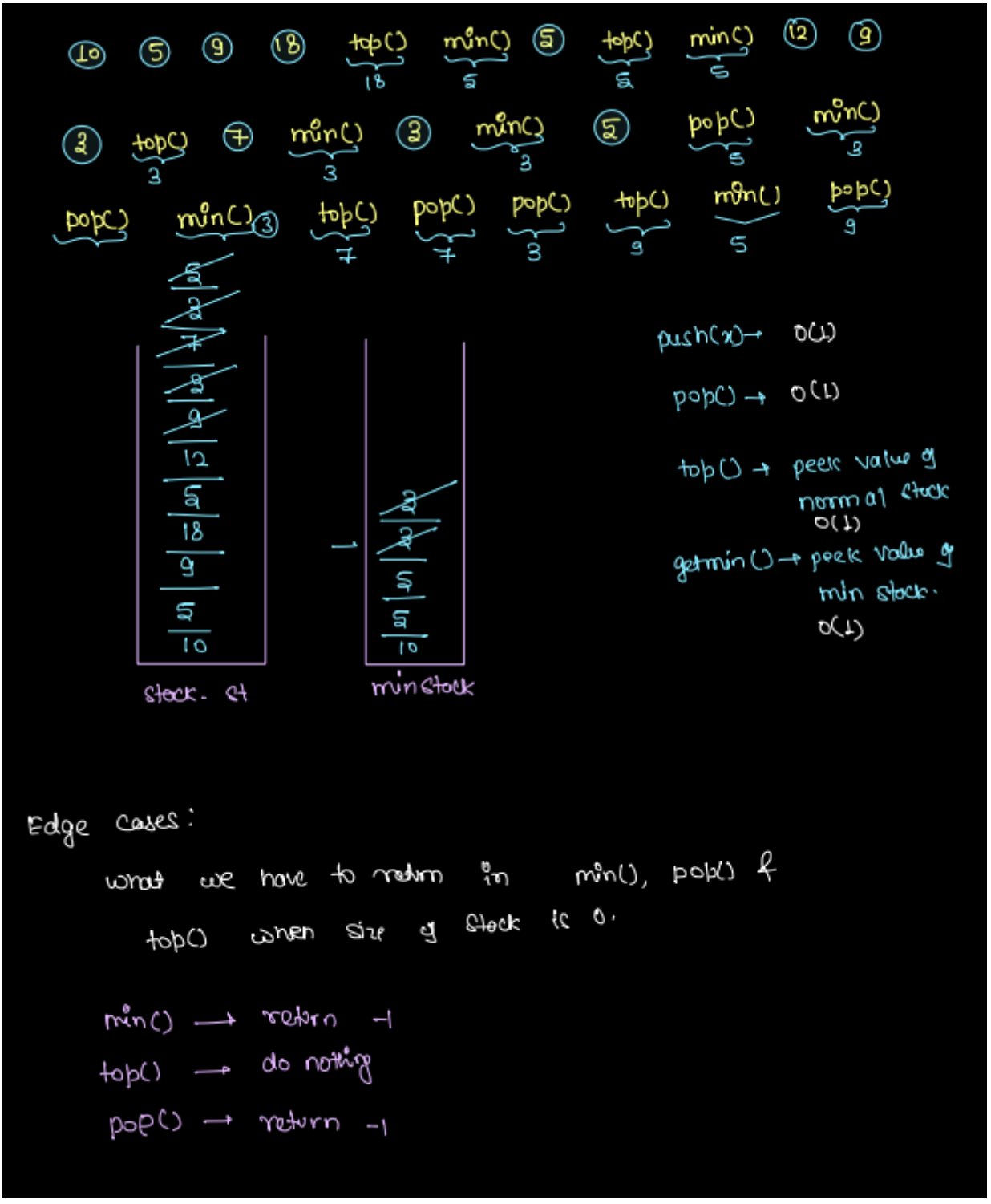
Main logic ->
First we will create class with the name of MinStack and define 2 stack stValue, stMin create constructor and inside it intialized both the stack. like
this.stVal = new Stack<>();
this.stMin = new Stack<>();

Create a four methods ->
push method -> Here we will check is stMin or stValue isEmpty then we will push value in both stack else we will push in stValue and check one condition if val <= minSt.peek then we will push it in minSt.

pop method -> Same edge case we will check it here stMin isEmpty if yes then return -1 else pop the value from stValue and check condition if value == minSt.peek then we will pop() value from stMin.

top Method -> Same edge case we will check stMin isEmpty return nothing else just return stValue.peek();

getMin Method -> check if minStack isEmpty return -1 elase return stMin.peek;



Structure Of Code :

```
class MinStack {
    → make two stacks   ① value stack or normal st.
                        ② min stack or min value

    void push(int x) {
        ① manage size == 0 condition.
        ② add value in valueStack.
        ③ add x in min stack (x <= top of minStack)
    }

    void pop() {
        ① manage size == 0 → Do nothing
        ② Pop the value from normal stack & store
          that val in a variable.
        ③ if minStack.peek() is equal to removed
          value, pop from minStack as well.
    }

    int top() {
        ① manage size == 0 → return -1
        ② Return peek from normal stack.
    }

    int getMin() {
        ① manage size == 0.
        ② return minStack.peek().
    }
}
```

```java
class Solution {

    Stack<Integer> normal;
    Stack<Integer> min;

    Solution() {
        this.normal = new Stack<>();
        this.min = new Stack<>();
    }
    public void push(int x) {
        if(min.size() == 0) {
            min.push(x);
            normal.push(x);
            return;
        }

        normal.push(x);

        if(x <= min.peek()) {
            min.push(x);
        }
    }
    public void pop() {
        if(min.size() != 0) {
            int rem = normal.pop();
            if(min.peek() == rem) {
                min.pop();
            }
        }
    }
    public int top() {
        if(min.size() == 0) {
            return -1;
        }
        return normal.peek();
    }
    public int getMin() {
        if(min.size() == 0) {
            return -1;
        }
        return min.peek();
    }
}
```

S.C. O(n)

Time Complexity of Every func. O(1)

2. Nearest Smaller in left [value Based]

https://www.interviewbit.com/snippet/52096872047280f380fe/

Main Logic ->

Begin by creating an empty array called ans with the same length as the given array, and also create a stack named minStack.

Set the default value of ans[0] to -1, and assign arr[0] as the initial value of minStack.

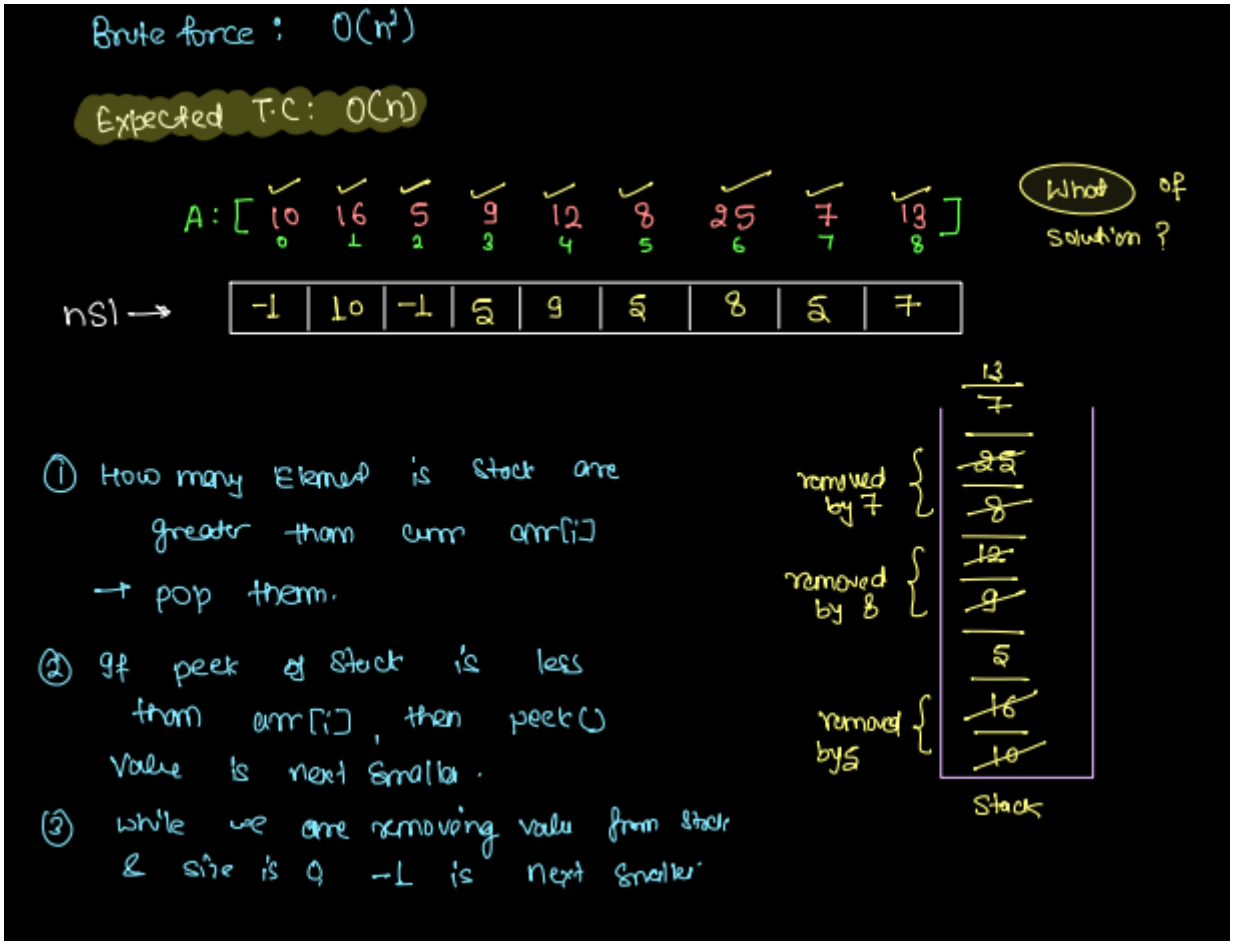
Iterate through the given arr from the start to the end, using an index variable i to keep track of the current index.

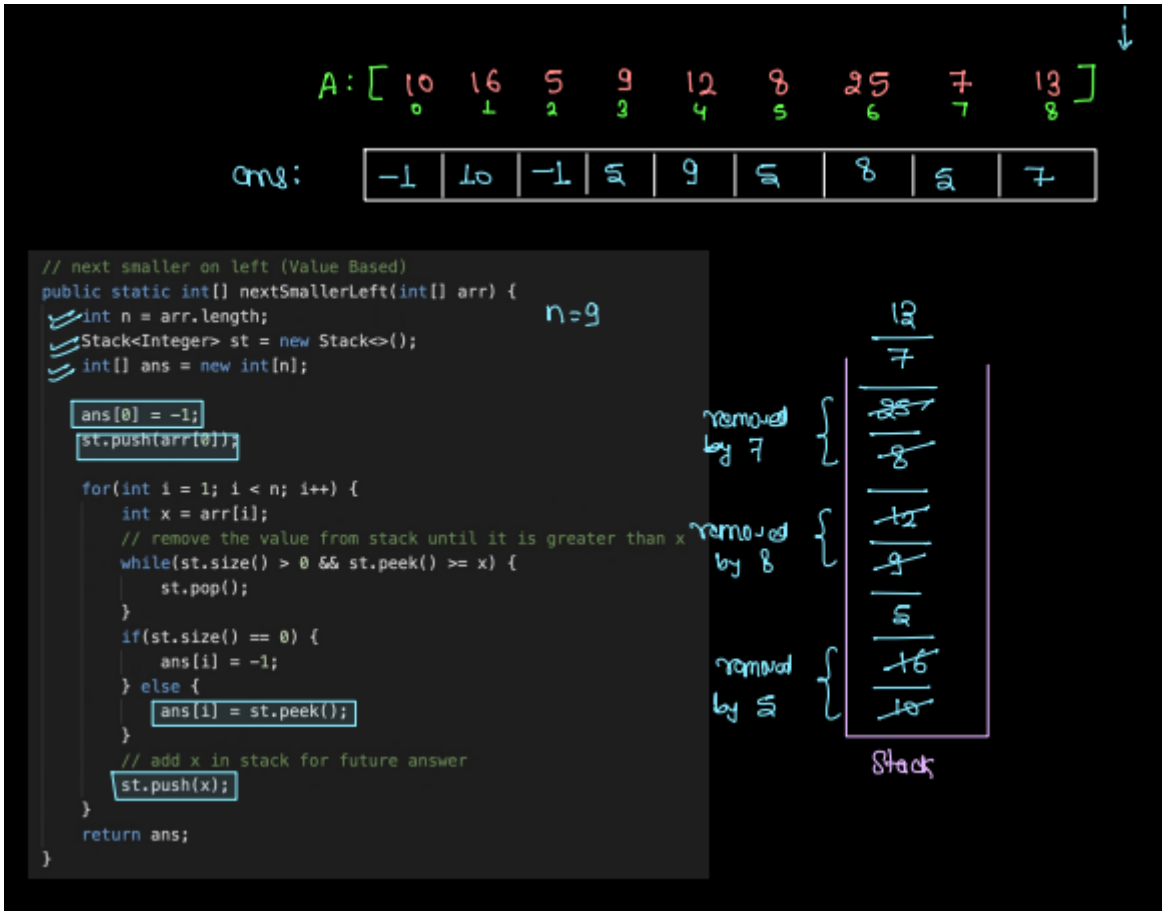
- Within the loop, assign the value at index i to a variable named val.
- Create a while loop that continues as long as minStack has elements and the top element of minStack is greater than val.
    - Inside the while loop, pop the top element from minStack.

After exiting the while loop, add a condition: if the size of minStack is equal to 0, add -1 to the ans array; otherwise, add the value at the top of minStack to the ans array.

Finally, push the value val to minStack.

After the completion of the for loop, return the ans array.





3. Nearest Smaller in left [index Based]

https://www.interviewbit.com/snippet/04a8bbe8e485527d14d5/

Main Logic ->

Begin by creating an empty array called ans with the same length as the given array, and also create a stack named minStack.

Set the default value of ans[0] to -1, and assign 0 as the initial value of minStack.

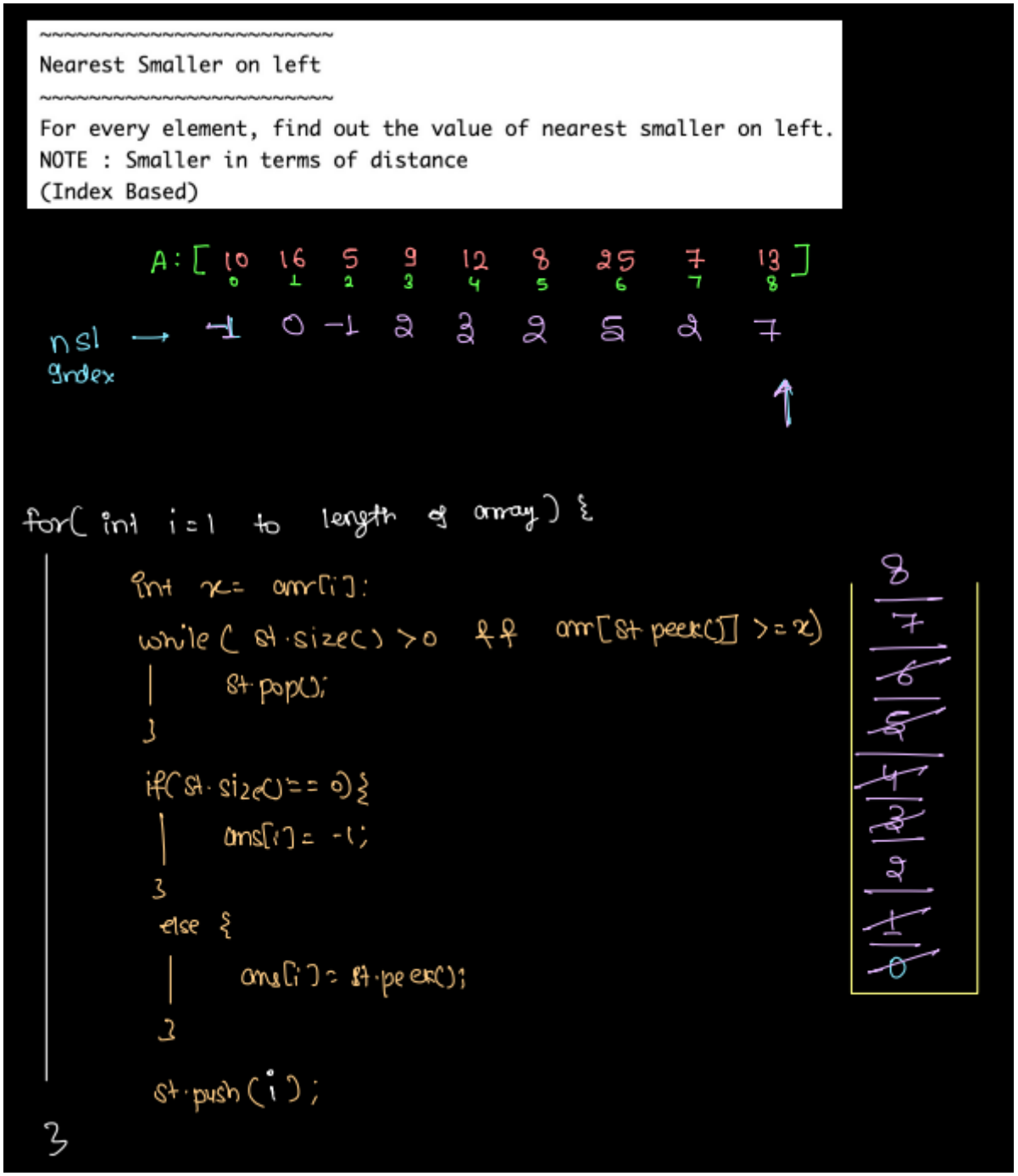
Iterate through the given arr from the start to the end, using an index variable i to keep track of the current index.

- Within the loop, assign the value at index i to a variable named val.
- Create a while loop that continues as long as minStack has elements and get the array value of index the top element of minStack arr[minStack.peek()] is greater than val.
    - Inside the while loop, pop the top element from minStack.

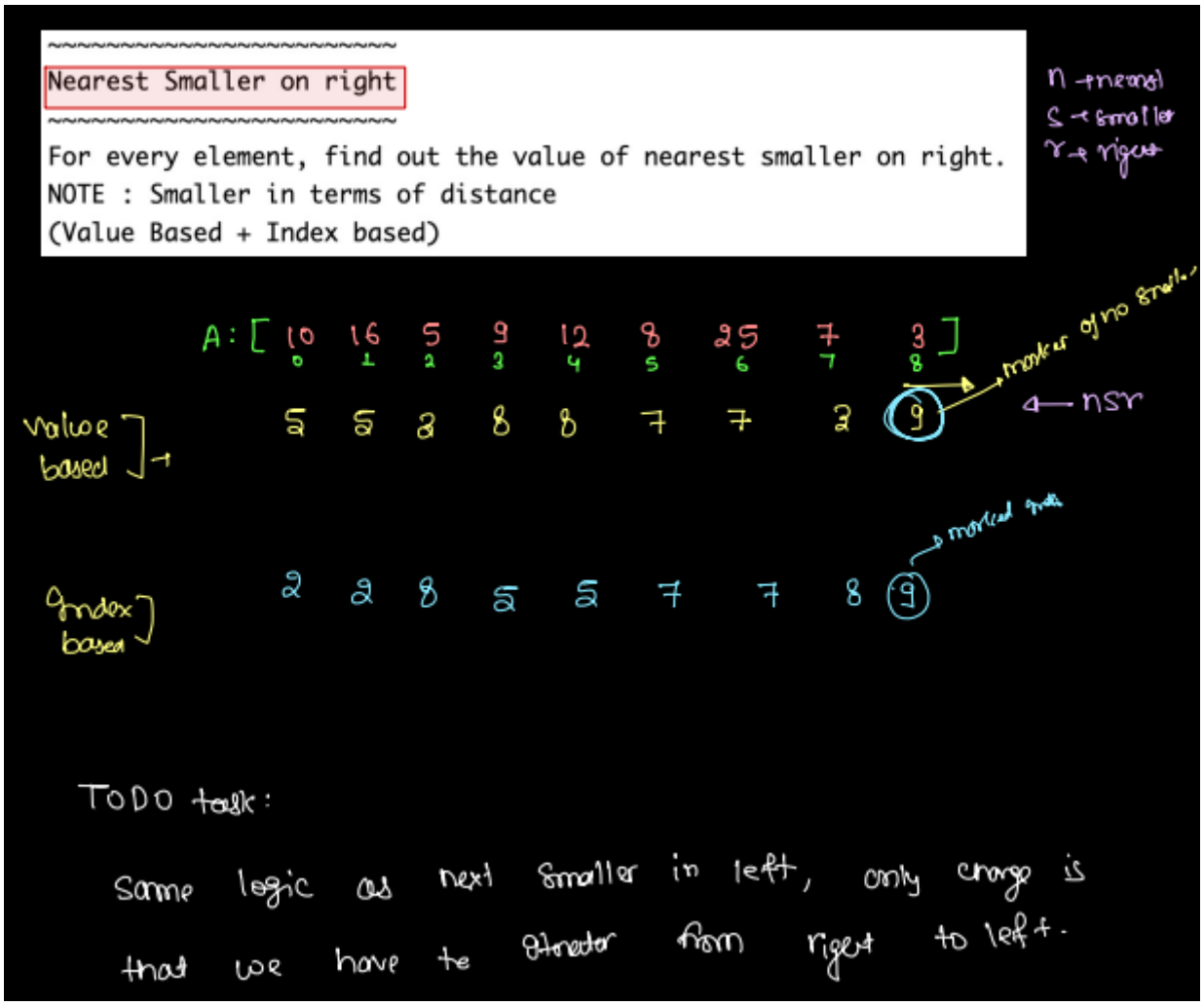
After exiting the while loop, add a condition: if the size of minStack is equal to 0, add -1 to the ans array; otherwise, add the value at the top of minStack to the ans array.

Finally, push the index to minStack.

After the completion of the for loop, return the ans array.



4. Nearest Smaller in right [index Based]
https://www.interviewbit.com/snippet/392a35f7c456108744af/



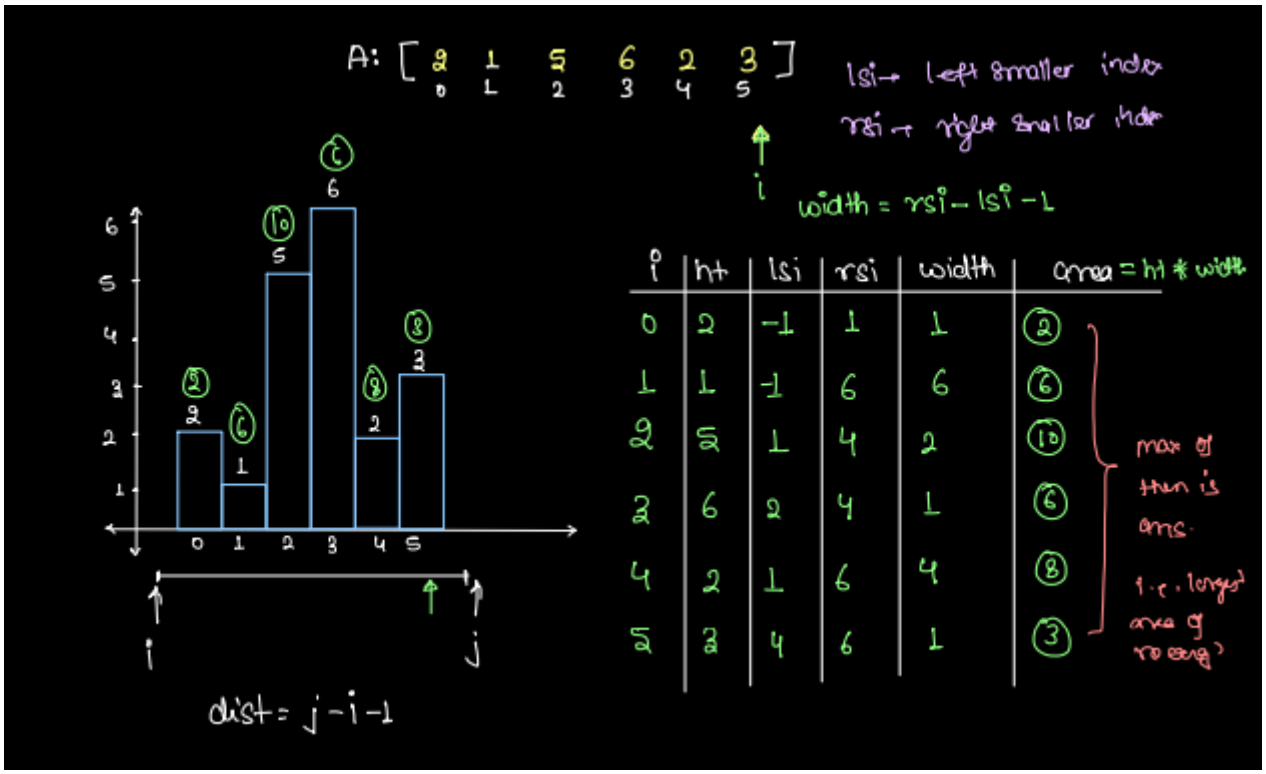
4. Largest Rectangle in Histogram

Given an array of integers A.
A represents a histogram i.e A[i] denotes the height of the ith histogram's bar. Width of each bar is 1.
Find the area of the largest rectangle formed by the histogram.
https://www.interviewbit.com/snippet/7f1c47f6f44eff2fadf7/

Main logic ->

- To solve this problem, begin by creating two functions: one to get the left minimum index and another to get the right minimum index. Assign the results of these functions to two separate arrays.
- Next, create a variable called MaxRect and set it to 0. This variable will be used to store the maximum rectangle area.
- Now, iterate over the loop from 0 to the length of the arr array. Within the loop, create a new variable and calculate the width of the rectangle using the rightMinIndex and leftMinIndex obtained by invoking the respective functions with the current index. To calculate the rectangle area, use the formula (width * height), where the width is given by (rightMinIndex - leftMinIndex - 1). Subtracting 1 is necessary as we do not include the first and last indices of the left and right minimum indexes.
- After calculating the rectangle area, compare it with the current maximum rectangle area using Math.max(MaxRect, rect). Assign the maximum value to MaxRect.

- Finally, after the loop, return MaxRect as the answer



5 - Next Greater - Additional problem

https://www.scaler.com/academy/mentee-dashboard/class/73408/homework/problems/263?navref=cl_tt_lst_nm

https://www.interviewbit.com/snippet/60cd5d99a4456d4faeb6/

Initialize variables: n as the length of array A, ans as a new integer array of size n, and st as an empty stack.

- Push the last element of A onto the stack st using st.push(A[n-1]).
- Set ans[n-1] as -1 since there are no elements to the right of the last element.
- Start a loop from i = n-2 and iterate backwards until i reaches 0.
- Within the loop, check if the stack is not empty and if the current element A[i] is greater than or equal to the top element of the stack.
- If the conditions are true, pop elements from the stack using st.pop() until either the stack is empty or the top element is greater than A[i]. because maybe current element is potential grater for next ekment.
- After the while loop, if the stack is empty, set ans[i] as -1 since there are no greater elements to the right of A[i]. Otherwise, set ans[i] as the top element of the stack.
- Push the current element A[i] onto the stack using st.push(A[i]).

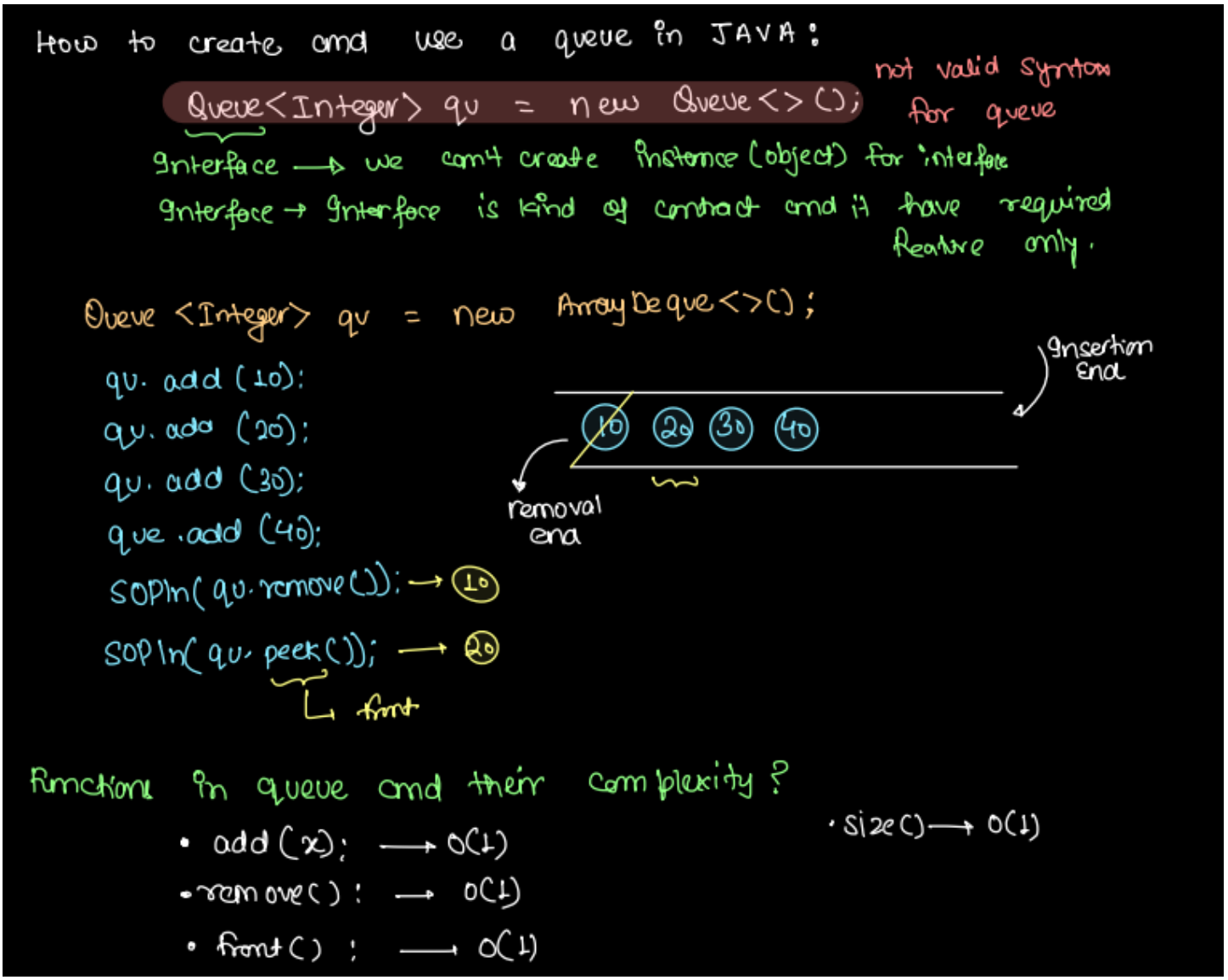
After the loop ends, the ans array will contain the next greater element for each element in A.

- Return the ans array as the output of the function.

DSA: Queues- 18 - July 2023

Introduction of Queue
Reverse first K element
Create N numbers using 1, 2, and 3
Inbuilt Linkedlist
Adaptors and implement of Queue using LL

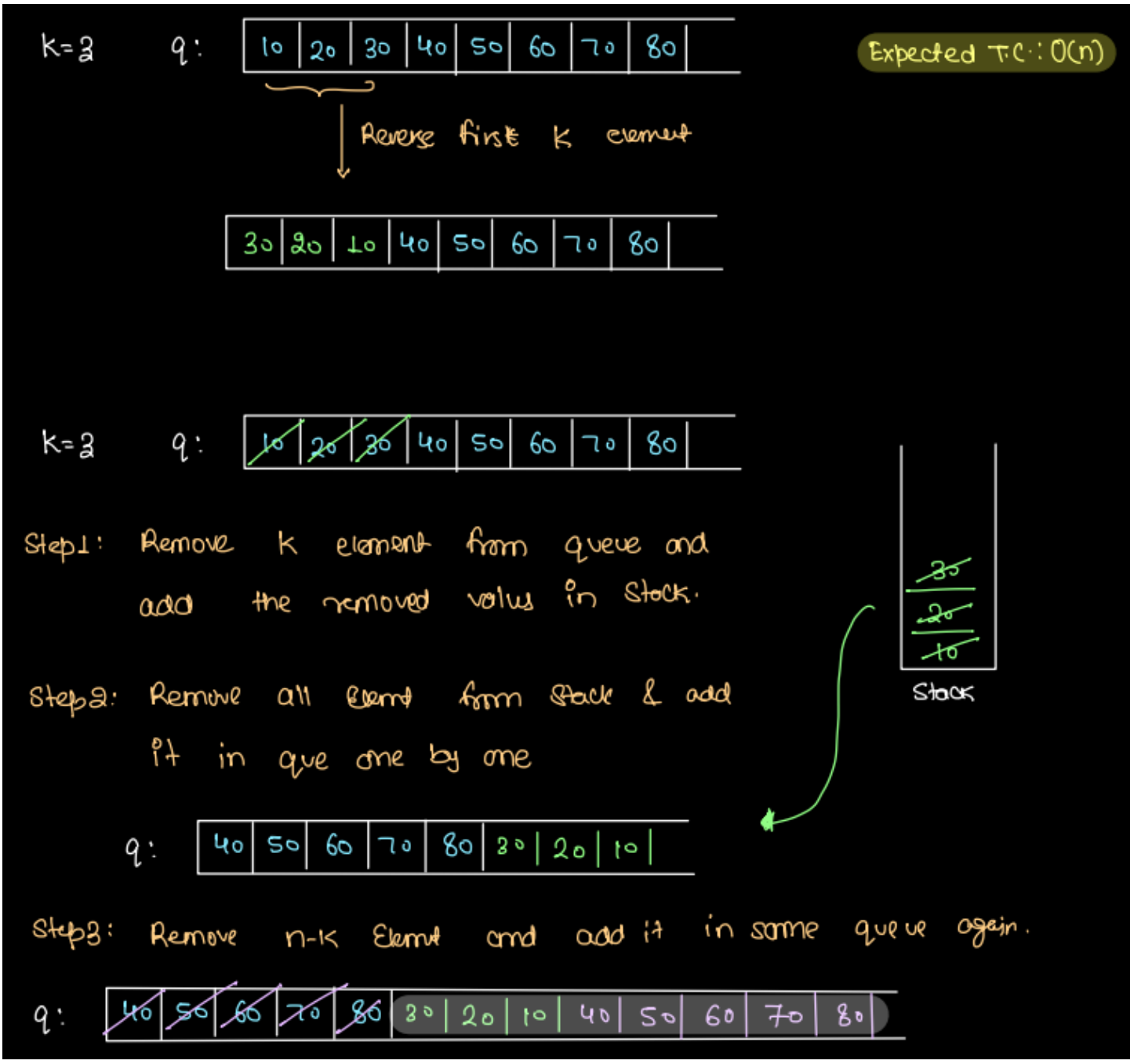
Introduction of Queue ->
Neture of queue is FIFO (First in first out). exp. Printing
queue<Integer> qu = new ArrayDeque<>();
Here we can se queue<Interger> this part is typeface it means it is a part of ArrayDeque. thatsby we use new ArrayDeque because this is the class.
function in queue and there complexity
add(x) -> O(1)
remove() -> O(1)
peek() -> O(1)



Problem1 ->
Given a queue, reverse first k element of it.
https://www.interviewbit.com/snippet/87885c085d0029e42f2c/

Main logic ->

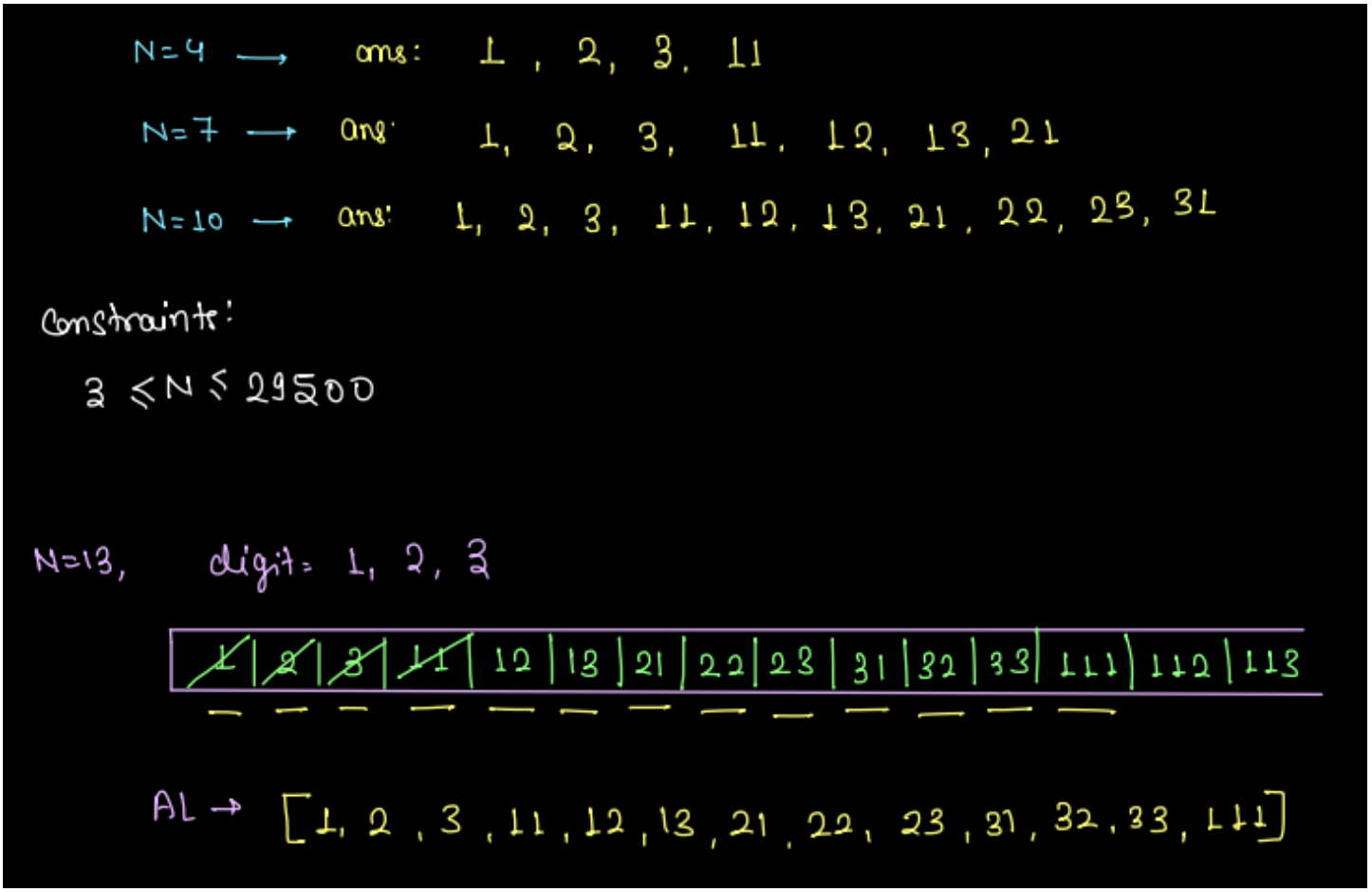
- To reverse the kth element, start by creating an empty stack. If the input is a queue, iterate through it until you reach the kth element. Remove each item from the queue using the "qu.remove()" function and add it to the stack.
- After that, enter a while loop that continues until the stack is empty. Inside the loop, remove an element from the stack and add it back to the queue using the "add(x)" function.
- Proceed to run a for loop from 0 to n-k (exclusive) to iterate over the remaining elements in the queue. Remove an element from the front of the queue and add it again to the back using the same loop.
- Finally, return the modified queue.

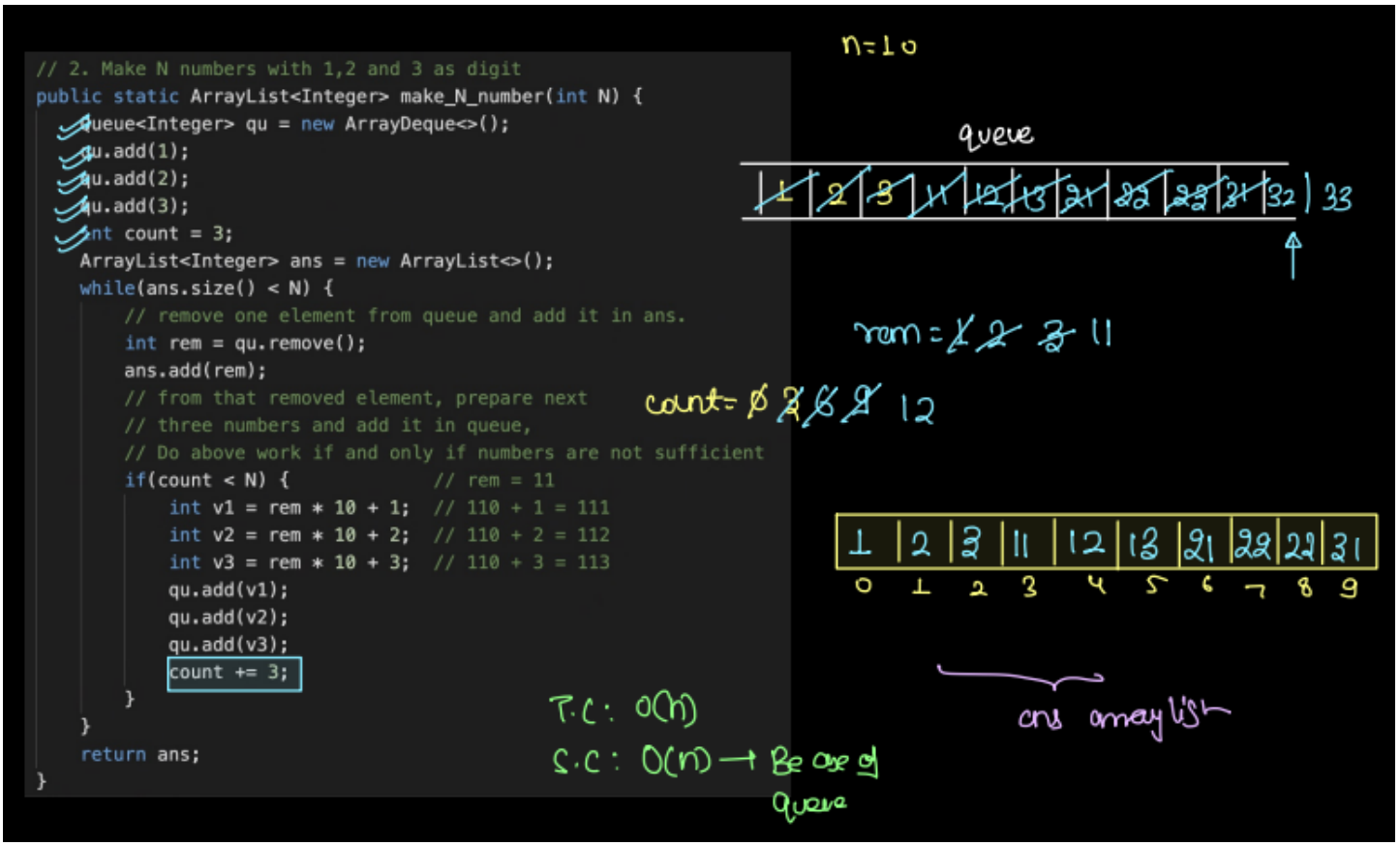


Probelm 2 -> **Create N numbers in ascending order using only 1,2, and 3 as digits and return these numbers in ArrayList**
https://www.interviewbit.com/snippet/8d7cad31d57e7f80eb8e/

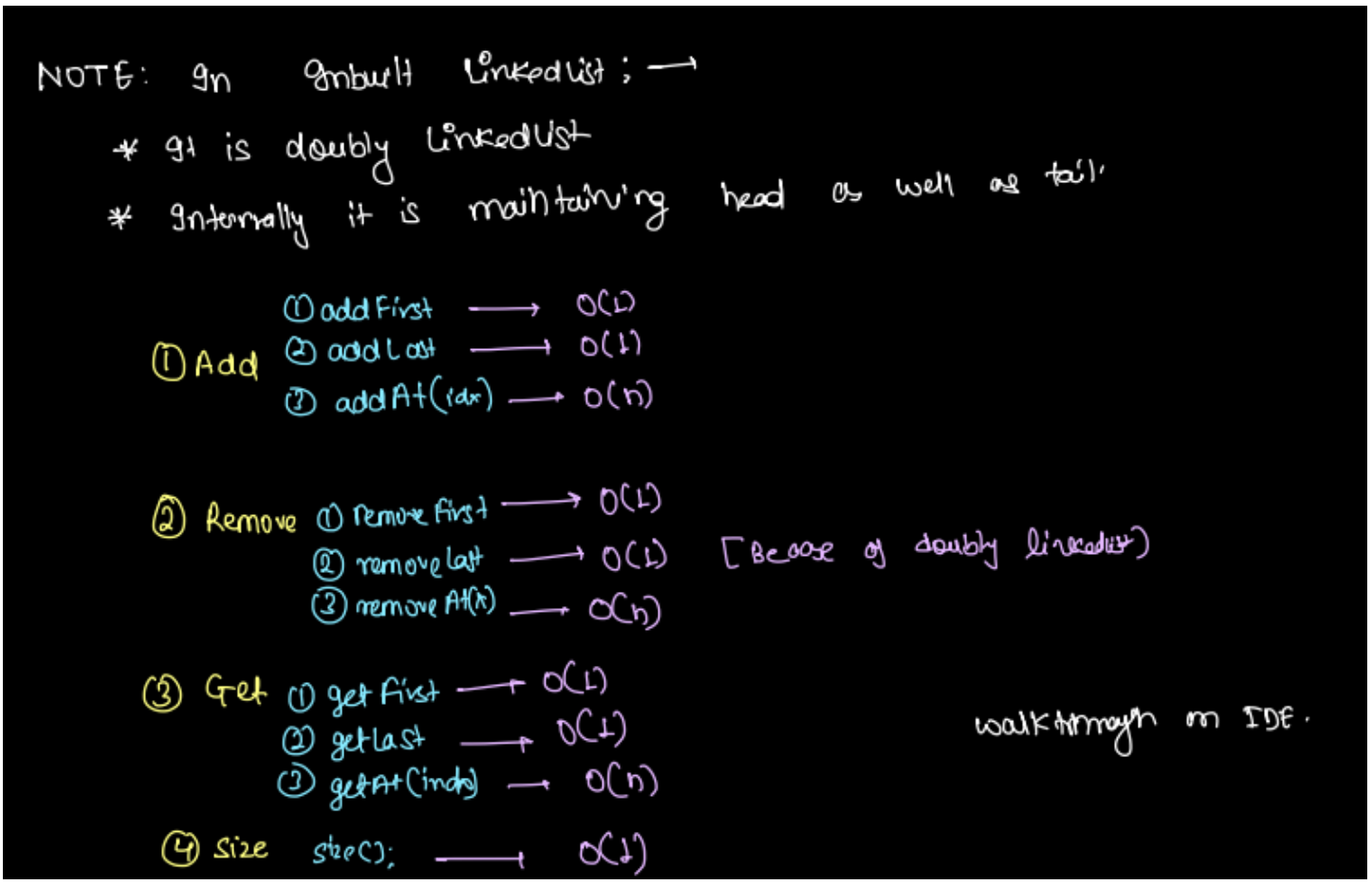
Main logic ->
Start by creating a Queue and populate it with the numbers 1, 2, and 3 as specified in the question.

- Create an ArrayList and initialize a variable called "count" with a value of 3 since we have added 3 numbers to the queue.
- Enter a while loop that continues until the size of the ArrayList is less than N.
- Inside the while loop, remove an element from the queue and add it to the ArrayList.
- Add a condition: if the count is less than N, perform the following steps:
    - Create three numbers: num1 = removedElem * 10 + 1, num2 = removedElem * 10 + 2, and num3 = removedElem * 10 + 3.
    - Add these three numbers to the queue.
- Increment the count by 3.
- Finally, after the while loop, return the ArrayList.

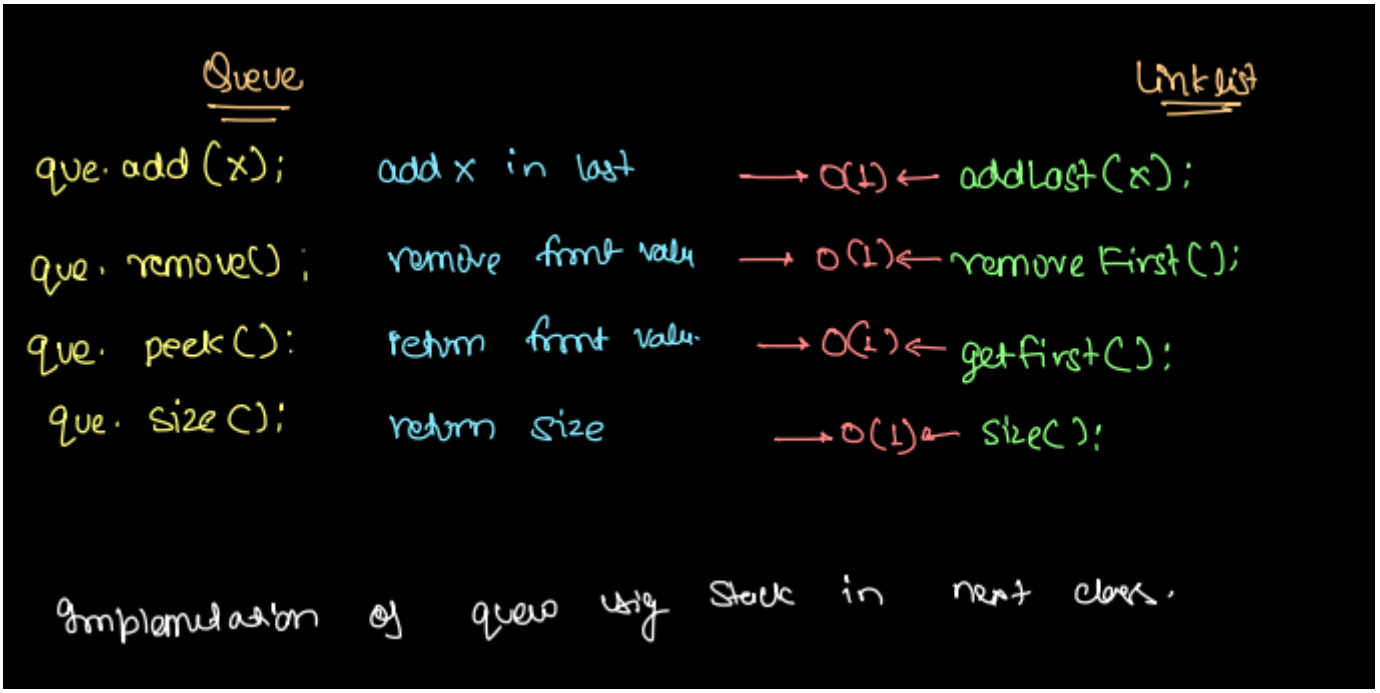




Inbuilt Linkedlist -> In java Inbuilt link list is dubly linked list. It has following function
addFirst(x), addLast(x), add(index, x) add at index
getFirst(), getLast(), get(index)
removeFirst(), removeLast(), remove(index)



Adaptors and implement of Queue using LL




```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Implementation of Queue using Linkedlist
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
public static class myQueue {
LinkedList<Integer> list;
public myQueue() {
this.list = new LinkedList<>();
}
public void add(int data) {
this.list.addLast(data);
}
public int remove() {
int rem = this.list.removeFirst();
return rem;
}
public int front() {
return this.list.getFirst();
}
public int size() {
return this.list.size();
}
public void print() {
System.out.println(this.list);
}
}
// use of myQueue
public static void demo3() {
myQueue qu = new myQueue();
```

```
qu.add(10);
qu.add(20);
qu.add(30);
qu.add(40);
qu.print();
System.out.println(qu.remove());
qu.print();
System.out.println(qu.front());
System.out.println(qu.size());
}
```

DSA: Deque - 20 - July 2023

First Non-Repeating Character
Introduction of Deque
Sliding Window Maximum
Queue using Stack (peek and remove efficient)
Assignment - Perfect Numbers

First Non-Repeating Character ->

Given a string A denoting a stream of lowercase alphabets, you have to make a new string B. B is formed such that we have to find the first non-repeating character each time a character is inserted to the stream and append it at the end to B. If no non-repeating character is found, append '#' at the end of B.
https://www.interviewbit.com/snippet/a6db762c2b3063ddafe7/

Main Logic =>
To begin, initialize three data structures: a queue, a hashmap, and a string builder.

- Proceed with a for loop iterating from 0 to the length of the given string.
- Within the loop, create a variable ch and assign it the ith character of the string.
- Check if the hashmap does not contain ch. If true, add ch to the hashmap with a frequency of 1 and enqueue ch into the queue. Otherwise, increment the frequency count of ch in the hashmap.
- Following the condition, add a while loop. Continue iterating as long as the queue is not empty and the frequency of the character at the front of the queue (retrieved from the hashmap) is greater than 1. Within the while loop, dequeue elements from the queue.
- After the while loop, introduce another condition to check if the queue is empty. If the queue is empty, append '#' to the string builder; otherwise, append the character ch.
- Upon finishing the for loop, convert the string builder to a string using toString and return it as the final result (ans).

Structure of Code :

HashMap< Char. , Int. > map —
Queue < Char > que —
StringBuilder ans —

for(i=0 to n-1)

any ith character is unique (not available in map)
→ add it in map with freq -1
→ add it in queue for upcoming potential ans.

otherwise
→ add in freqp with increased freq.

// prepare ans.

while (front of queue . freq is not 1)
remove front ()

if (que. size () == 0)
otherwise → que. front.

Ans → StringBuilder to String.

```java
public static String nonRepeating(String str) {
    HashMap<Character, Integer> map = new HashMap<>();
    Queue<Character> qu = new ArrayDeque<>();
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);
        // prepare frequency map
        if(map.containsKey(ch)) {
            // update frequency
            int old = map.get(ch);
            map.put(ch, old + 1);
        } else {
            // add it in map and queue
            map.put(ch, 1);
            qu.add(ch);
        }
        // prepare ans
        while(qu.size() > 0 && map.get(qu.peek()) > 1) {
            qu.remove();
        }

        if(qu.size() == 0) {
            sb.append('#');
        } else {
            sb.append(qu.peek());
        }
    }
    return sb.toString(); ⟶ aab#c
```

A: a b a b c
i

Sb: a a b # c

a | b | c

a → ✗ 2
b → ✗ 2
c → 1

i= 0 1 2 3 4 . . . . . n-1

while loop iteration ⟶ c_1 c_2 c_3 c_4 c_5 . . . c_n

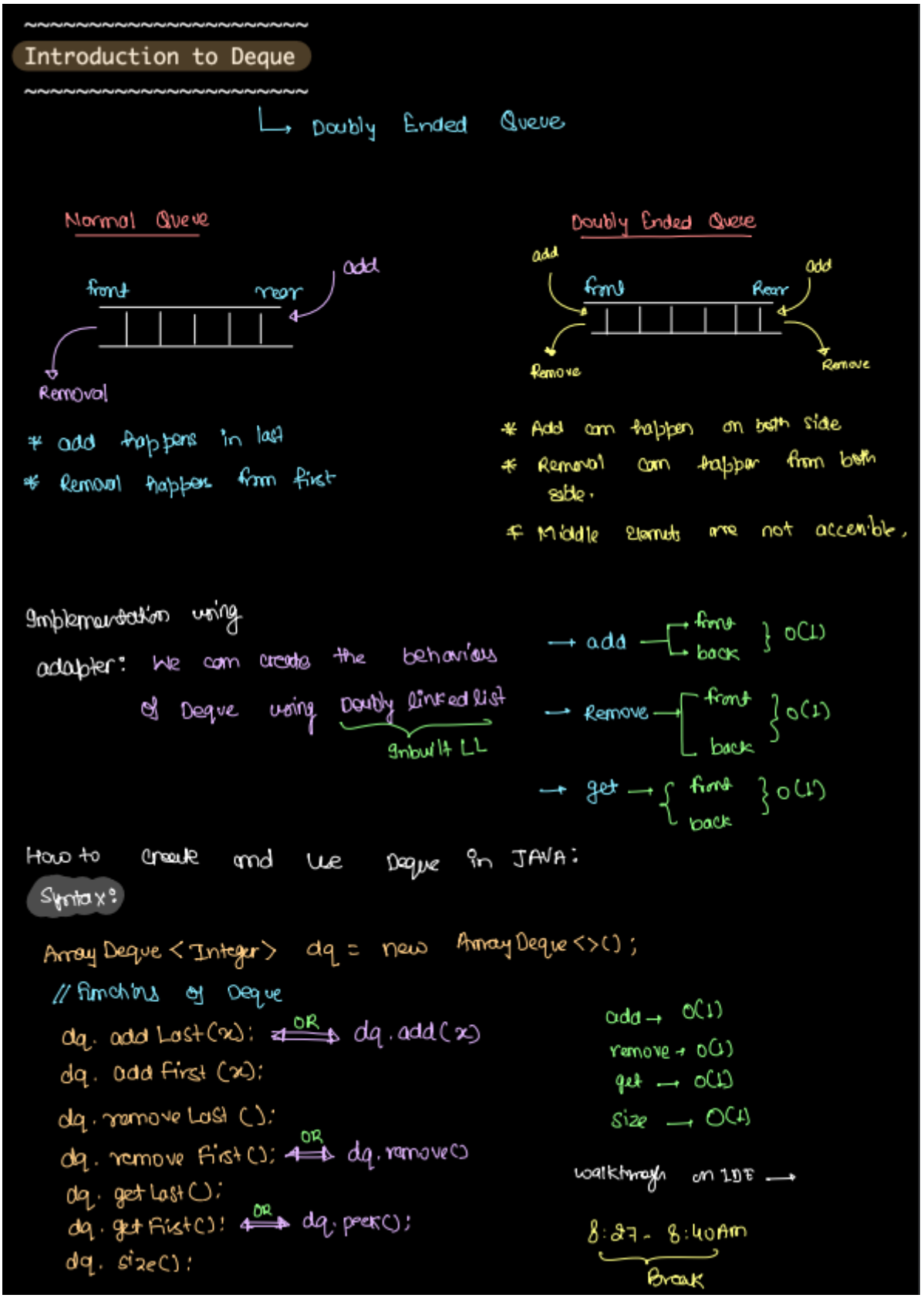
g_n worst case : $c_1 + c_2 + c_3 + \dots + c_n = n$

g_n worst case total code iter = $2n$

T.C : O(n)

S.C : Constant [map can have at most 26]

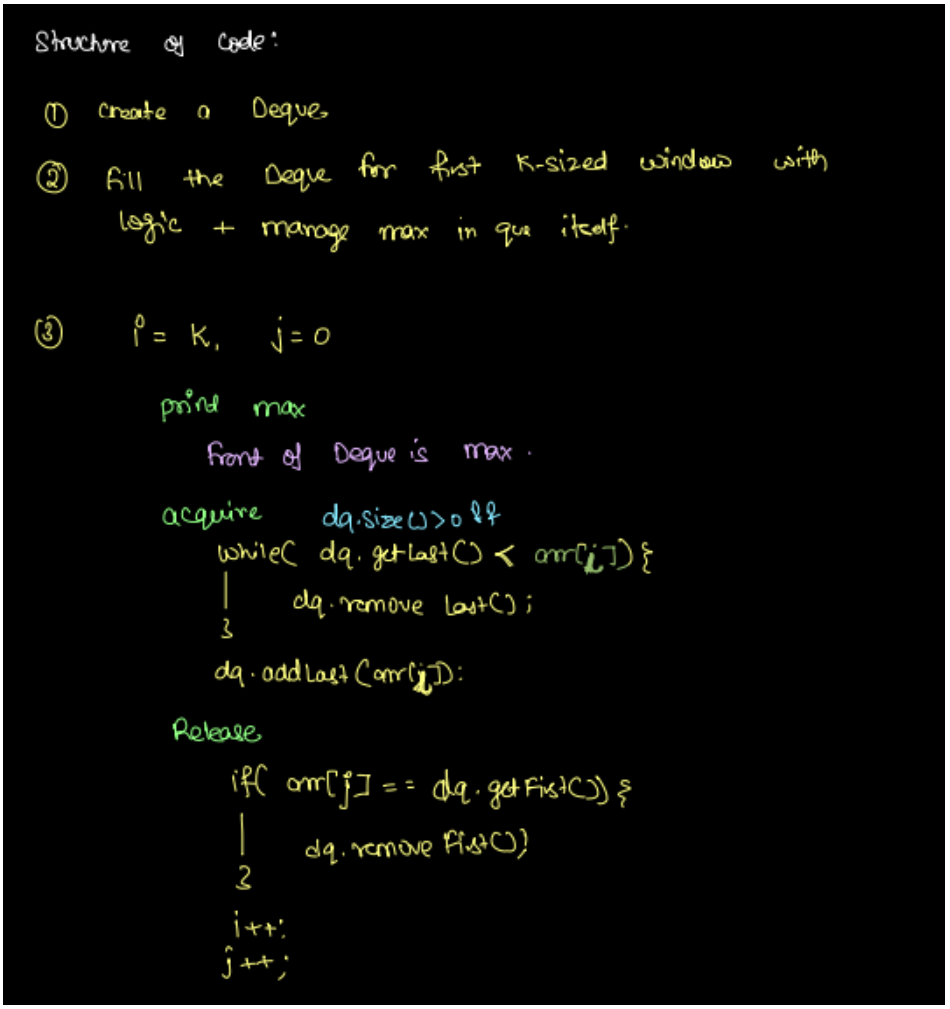
Sliding Window Maximum
Introduction of Deque



Sliding Window Maximum
https://www.interviewbit.com/snippet/75b6b4fc8fd033faecb8/

Main logic ->
- Create a ArrayDeque and 1 n variable and intilize n to A.lenght and create array ans till size (n - B + 1);
- First check edge case if array length 1 then return array
- we will get first window till B size.
 iterate for 0 to < B.lenght and inside loop use while and intrate dq.size() > 0 && dq.getLast < A[i] then dq.removeLast() after while end dq.addLast(A[i]);
    - Now we will define i = B and j = 0;
    - Start iterate while loop i <= n inside while
        - add dq.getFirst() in ans[i];
        - Now write a condition if(i >= n) then break the loop it will protect arrayoutofbound
        - Acquire one more item now use while dq.size() > 0 && dq.getFirst() < A[i] then dq.removeLast
        - after while dq.addLast(a[i]);
        - relise first item of dq write a condition if dq.getFirst() == A[j] then dq.removeFirst()
        - now increase i++, j++

after end the loop return ans array



```java
public static void slidingWindowMaximum(int[] arr, int K) {
    int n = arr.length;
    ArrayDeque<Integer> dq = new ArrayDeque<>();
    // calculate answer for first window (from i = 0 to i < k)
    for(int i = 0; i < K; i++) {
        while(dq.size() > 0 && dq.getLast() < arr[i]) {
            dq.removeLast();
        }
        dq.addLast(arr[i]);
    }
    int i = K, j = 0;
    while(i < n) {
        // print the answer of widnow
        System.out.println(dq.getFirst());
        // acquire ith index
        while(dq.size() > 0 && dq.getLast() < arr[i]) {
            dq.removeLast();
        }
        dq.addLast(arr[i]);
        // release jth index
        if(dq.getFirst() == arr[j]) {
            dq.removeFirst();
        }
        i++, j++;
    }
    System.out.println(dq.getFirst());
}
```

Q1. Sum of min and max
Given an array A of both positive and negative integers.
Your task is to compute the sum of minimum and maximum elements of all sub-array of size B.

Main logic ->

Begin by creating two ArrayDeque objects, 'dq' and 'dqlow', and a variable 'n' initialized to the length of array A. Also, create a variable 'ans' and set it to 0.

- **Check the edge case where the array length is 1, and if so, return the only element in the array A[0].**
- **Obtain the first window of size B from array A.**
    - **Iterate from 0 to B-1, and within the loop, use a while loop to repeatedly remove elements from the end of 'dq' while its size is greater than 0 and its last element is less than the current element A[i]. Also, use another while loop to remove elements from the end of 'dqlow' while its size is greater than 0 and its last element is greater than the current element A[i]. After the while loops, add the current element A[i] to the end of both 'dq' and 'dqlow'.**
- **Set 'i' to B and 'j' to 0 to start the main iteration.**
- **Start a while loop while 'i' is less than or equal to 'n':**

- **Calculate the sum of the first elements from 'dq' and 'dqlow', and add it to the 'ans' variable.**
- **Check if 'i' is greater than or equal to 'n', and if so, break the loop to avoid an array out-of-bounds error.**
- **Obtain the next item from array A and use a while loop to repeatedly remove elements from the end of 'dq' while its size is greater than 0 and its first element is less than the current element A[i]. Also, use another while loop to remove elements from the end of 'dqlow' while its size is greater than 0 and its first element is greater than the current element A[i]. After the while loops, add the current element A[i] to the end of both 'dq' and 'dqlow'.**
- **Remove the first item from 'dq' if it matches A[j].**
- **Remove the first item from 'dqlow' if it matches A[j].**
- **Increment 'i' and 'j' by 1.**

- **After exiting the loop, return the 'ans' variable as the final result.**

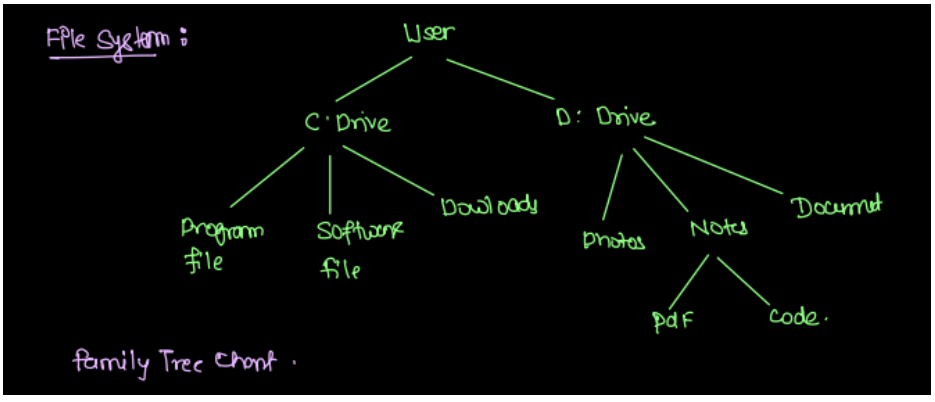
DSA: Tree1 - 22 - July 2023

Why Tree is required?
Terminologies in tree
Traversal in Binary Tree
Dry run with euler diagram
Pre, In and Post Order with area
Size of Tree
Sum of node in tree

Why Tree is required -> below are the Linear datastructure and we have to store data in Hierarchial order like tree compony orgnization or family tree in that it is very hard to store in Linear data structure thats by we will use tree.

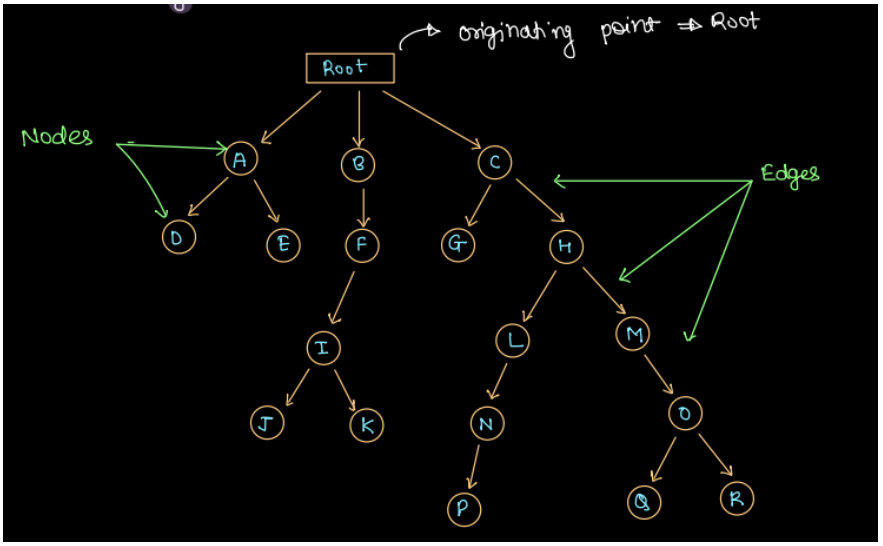
Linear Data Struncture =>
Continue memory allocation ->
Arrays, ArrayList, String, Stack, (Hashmap, Hashset) (combination of arraylist and linkedlist) this all
Non continous memory allocation ->
Linkedlist

Non linear Data Strucre or Hierarchical like tree

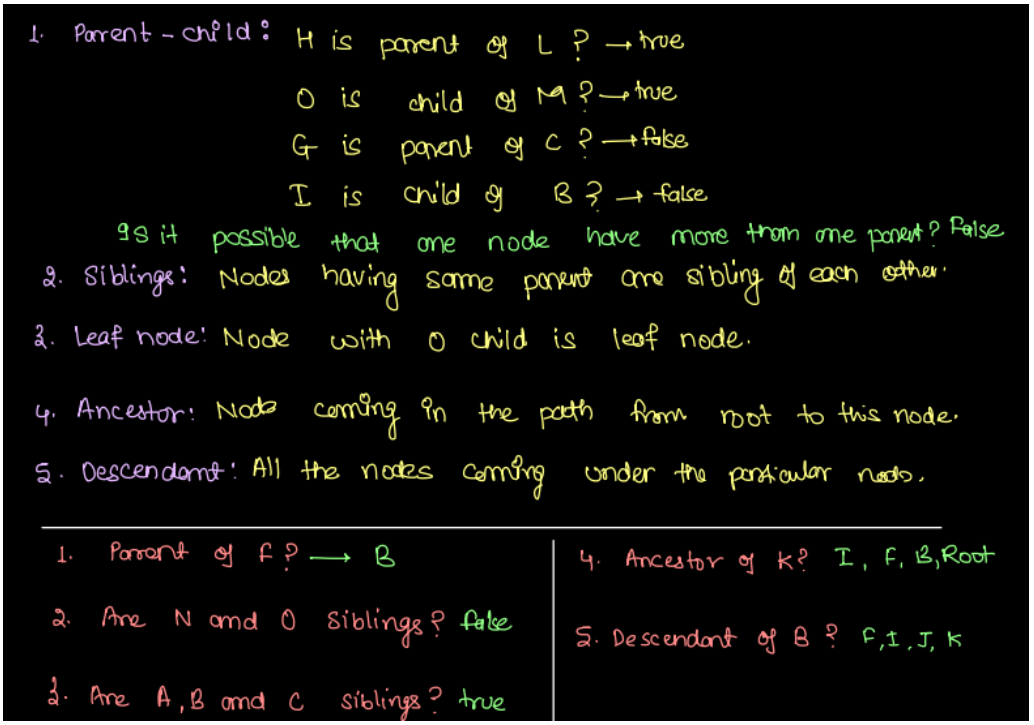


Terminologies in tree —----------

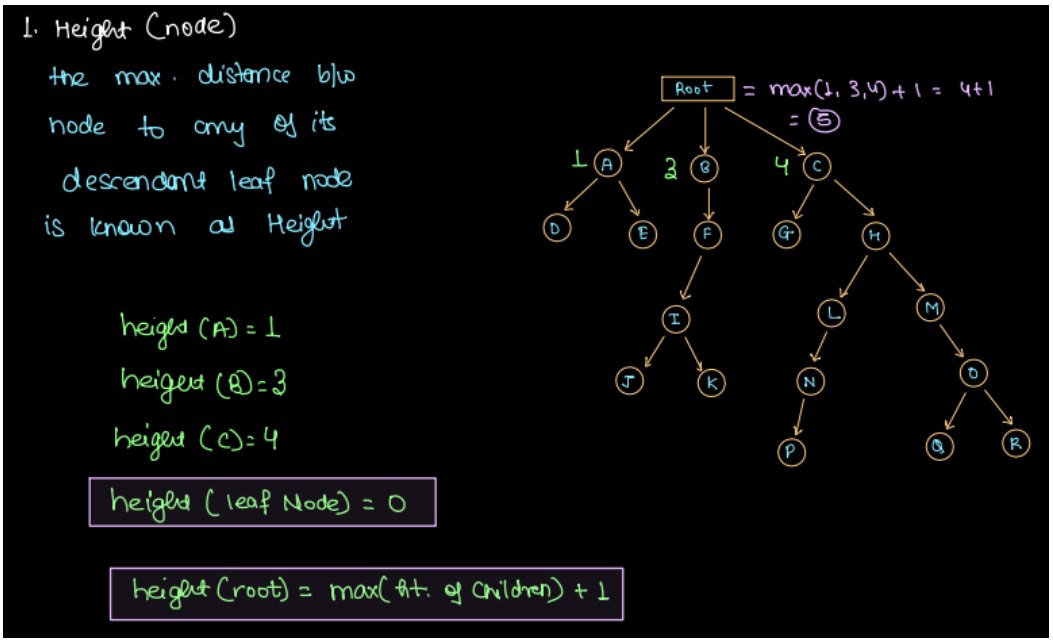
Root -> Top most node of tree
Edges -> distance between 2 nodes
Nodes -> items in tree



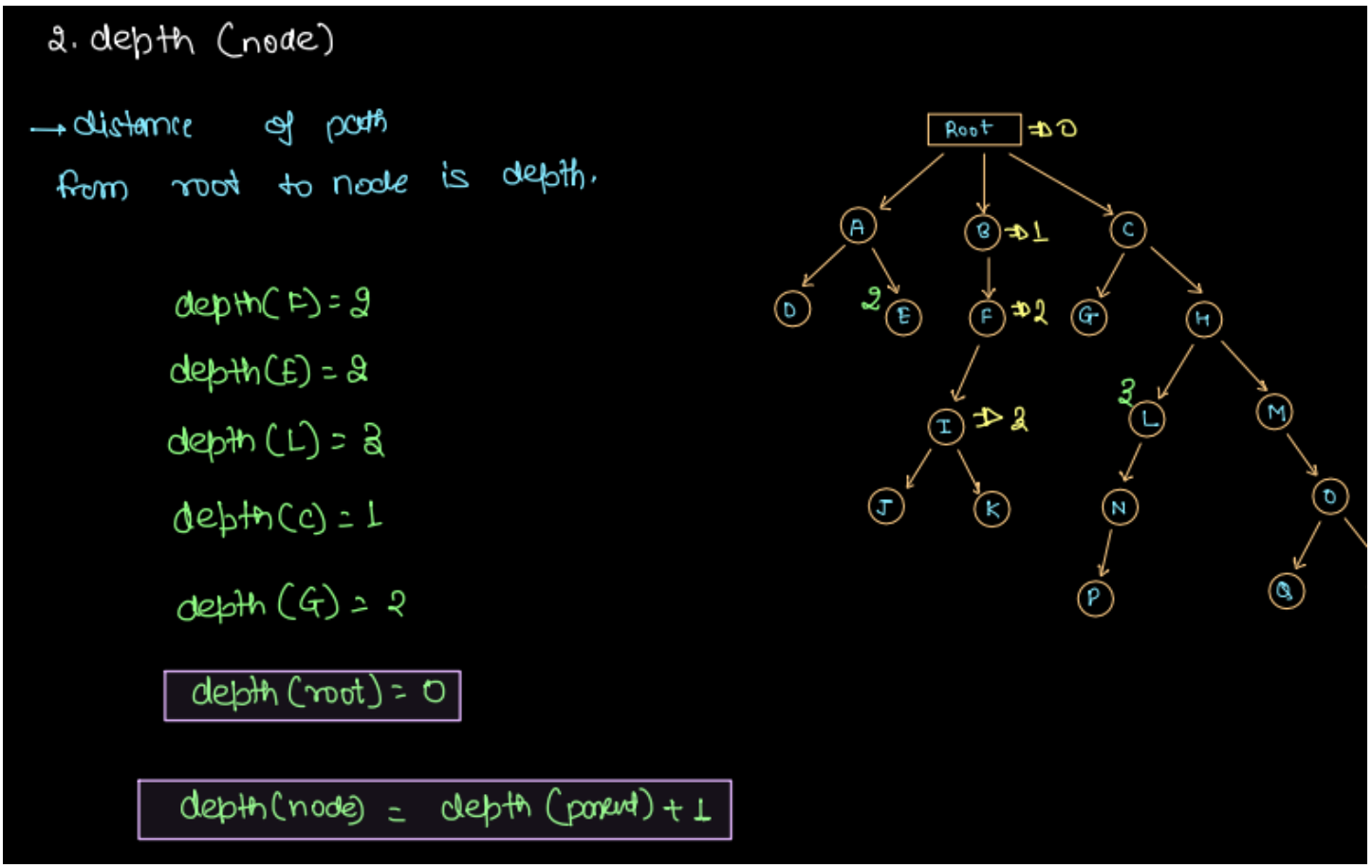
Parent-> just previous node of current node which connected directly
Child: Direct decendent of node like 1 lavel
Siblings -> Nodes having same parnet are sibling of each other
Leaf node -> Node with 0 Child is leaf node.
Acestor -> Node coming in the path from root to this node.
Descendent -> All the notes coming under the particulre node.



Height (node) -> The max distance between node to any of its desendant leaf node is known as height
Height of tree(root) = Max(leaft_nodes, right_nodes) + 1;



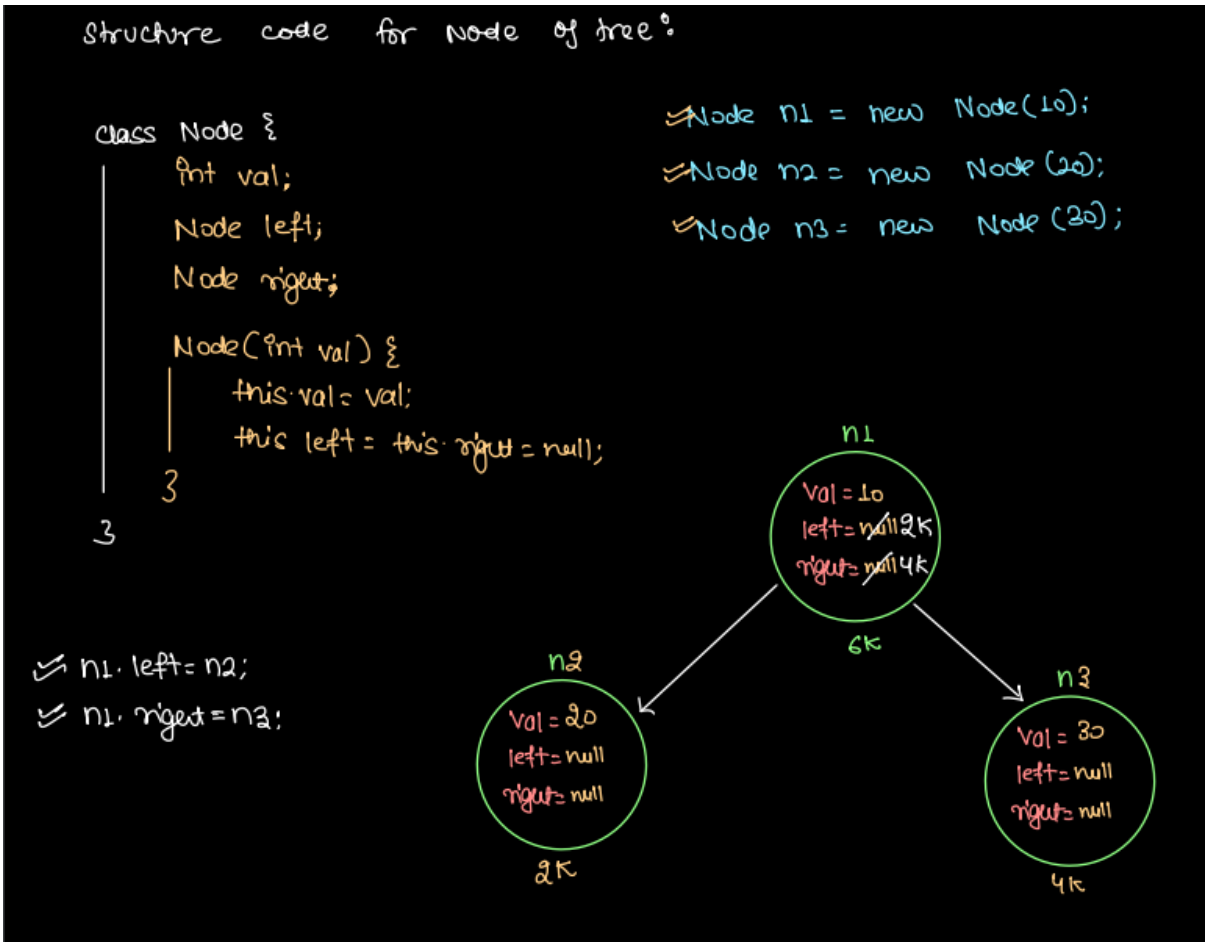
Depth (node) -> Distance of path from root to node is depth
depth(node) = depth(parent) + 1



Genric tree (N-ary tree) -> Whenever we have 0, 1, 2 or more then 2 child like N tree is possible. Children count is not fix but dynamic. it's called Genric tree.
Binary Tree -> It can have only 0, 1 or max 2 child.

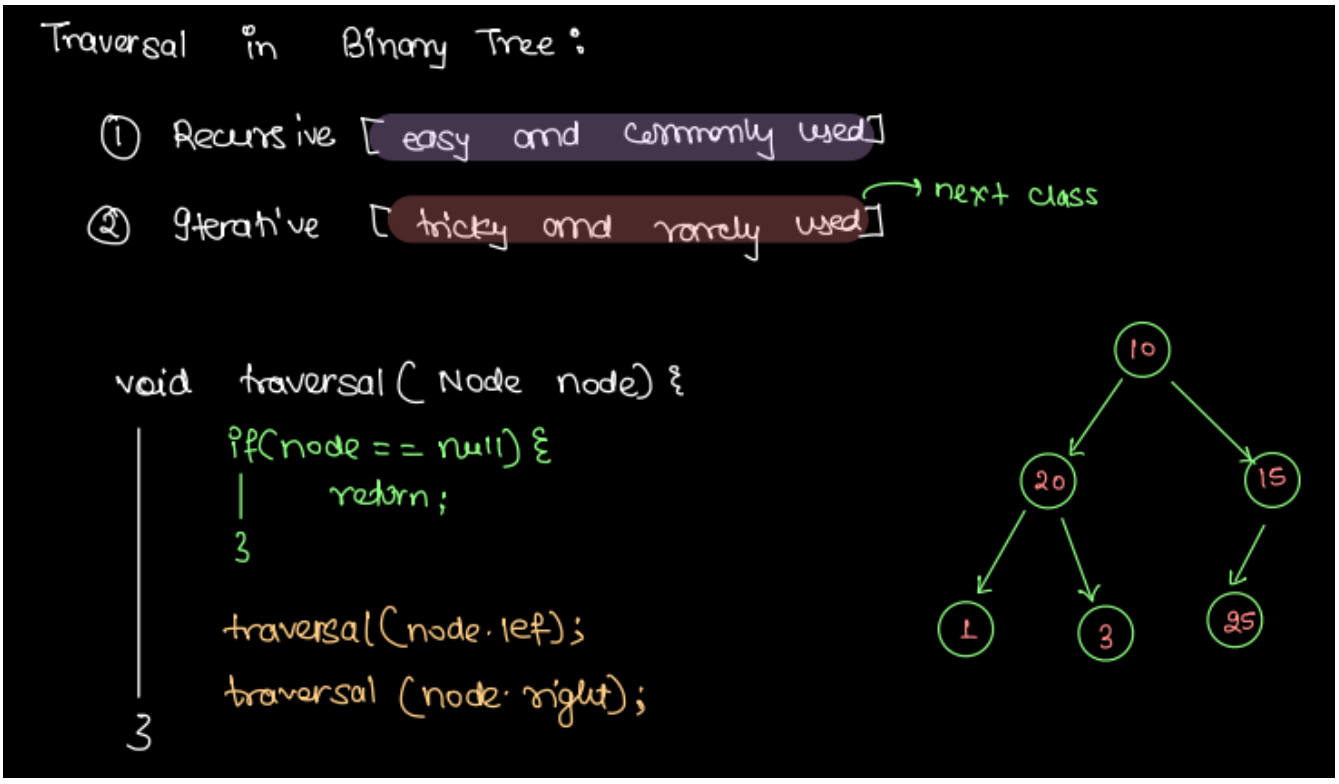
Structure of Node =>

Every node has these information value, left ndoe address and right node address.



Traversal in Binary Tree -> —----------

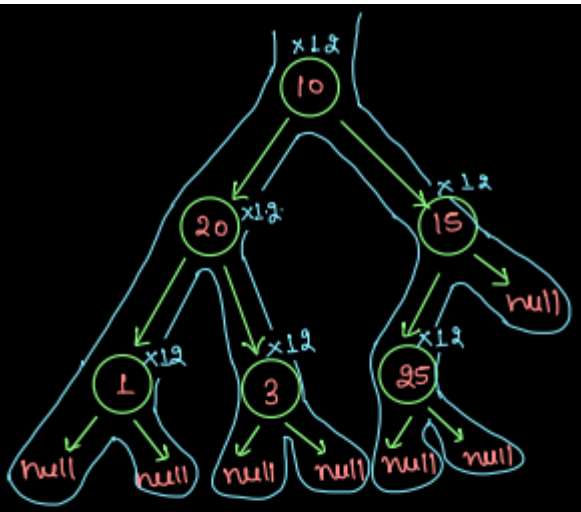
https://www.interviewbit.com/snippet/b79072e1cb4741cce7da/

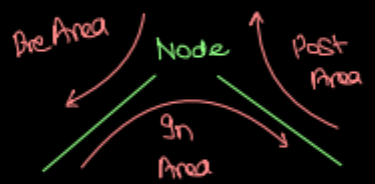


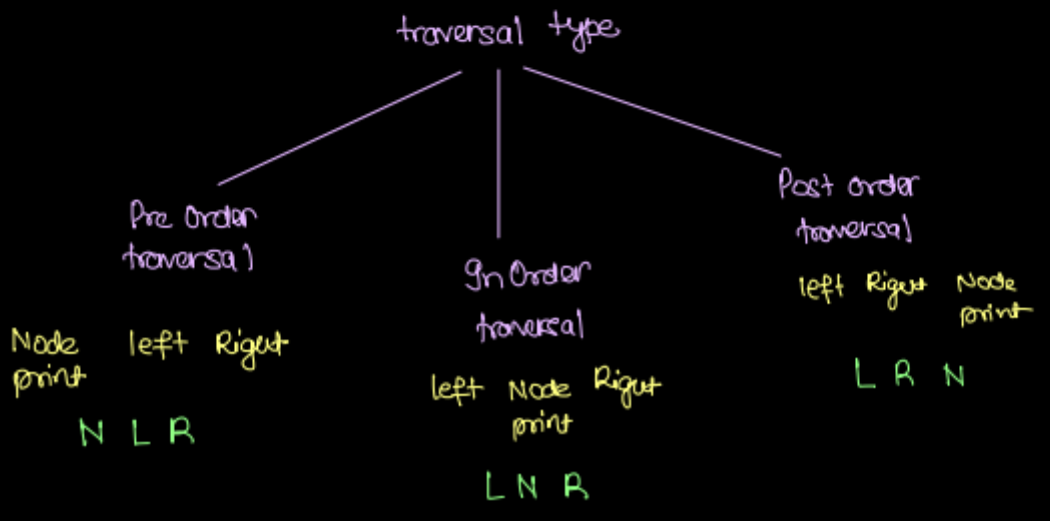
Dry Run :

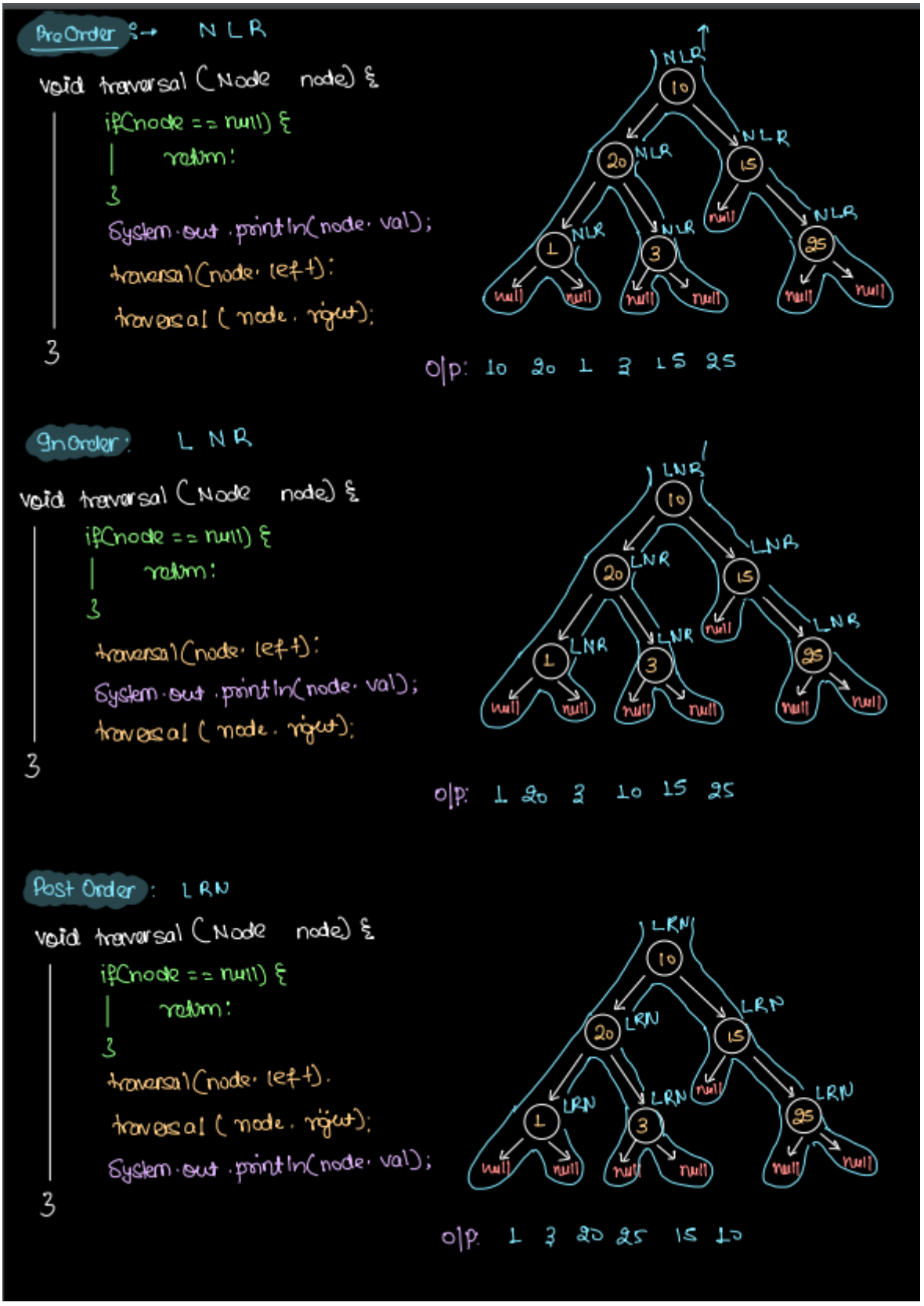
with help of Euler Diagram.

```
void traversal ( Node node) {
    if [ if(node == null) {
         |     return;
         3
    1. [traversal(node.left);
    2. [traversal (node.right);
3
```

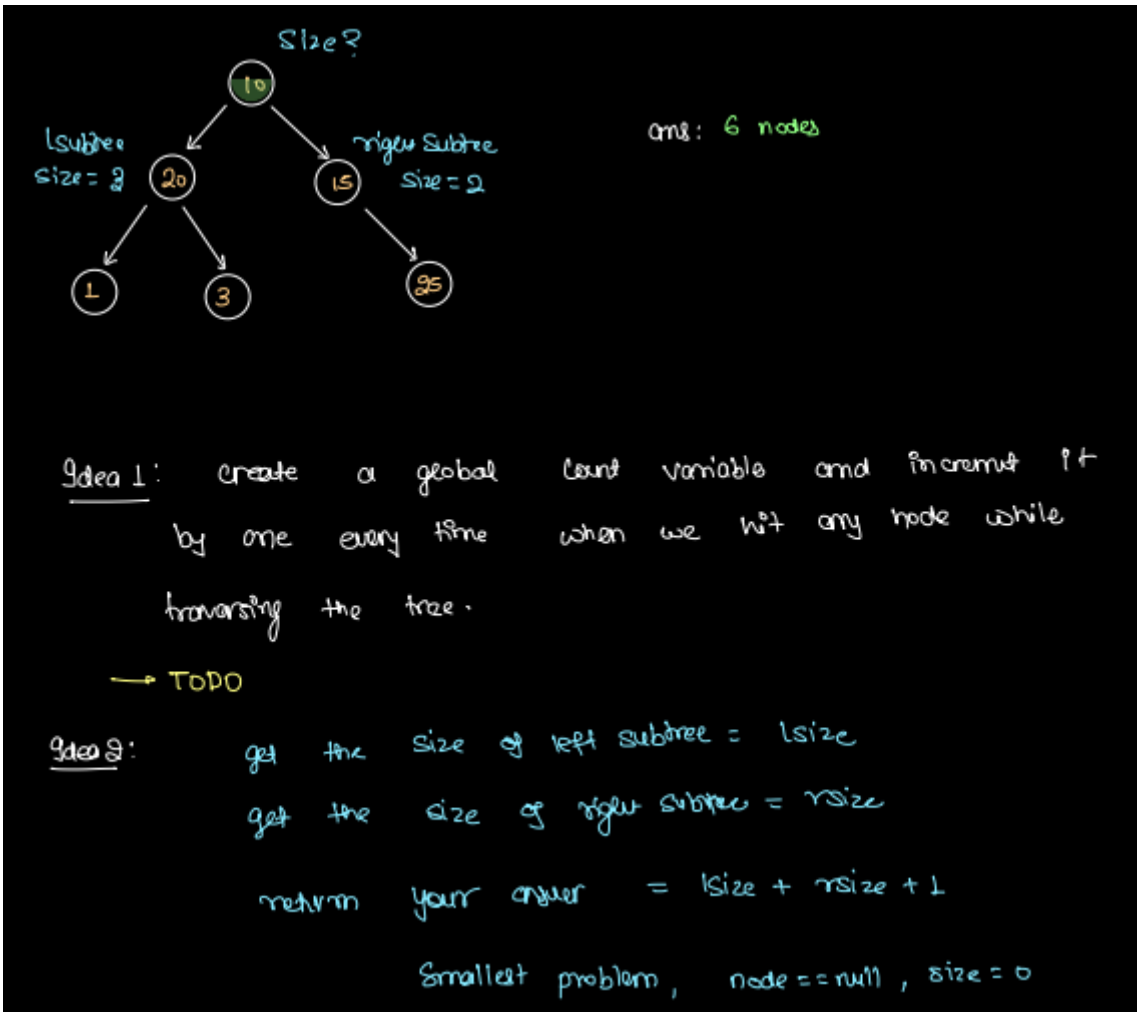


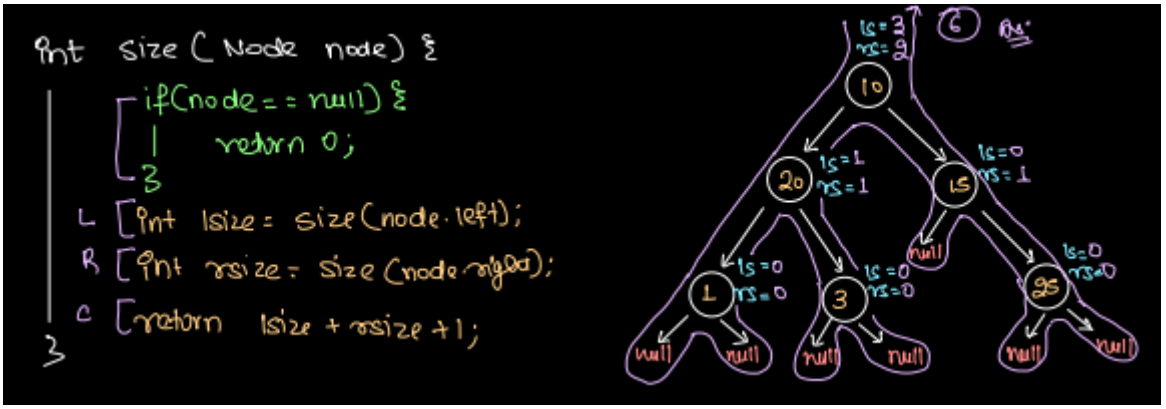




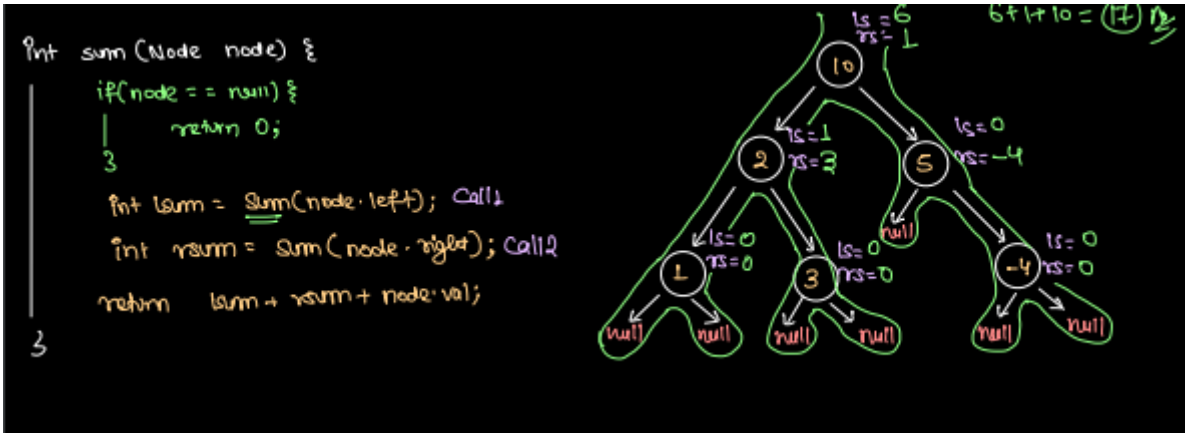


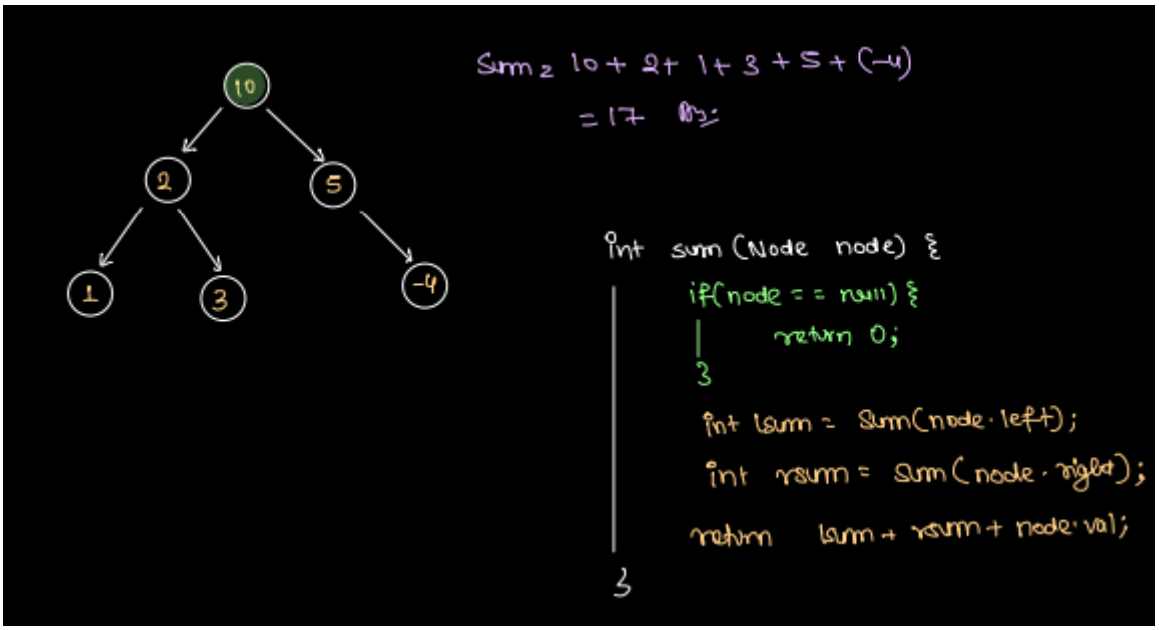
Problem 1: Given root of binary tree, find its size (count of nods)
https://www.interviewbit.com/snippet/922e22954fca6075af5a/





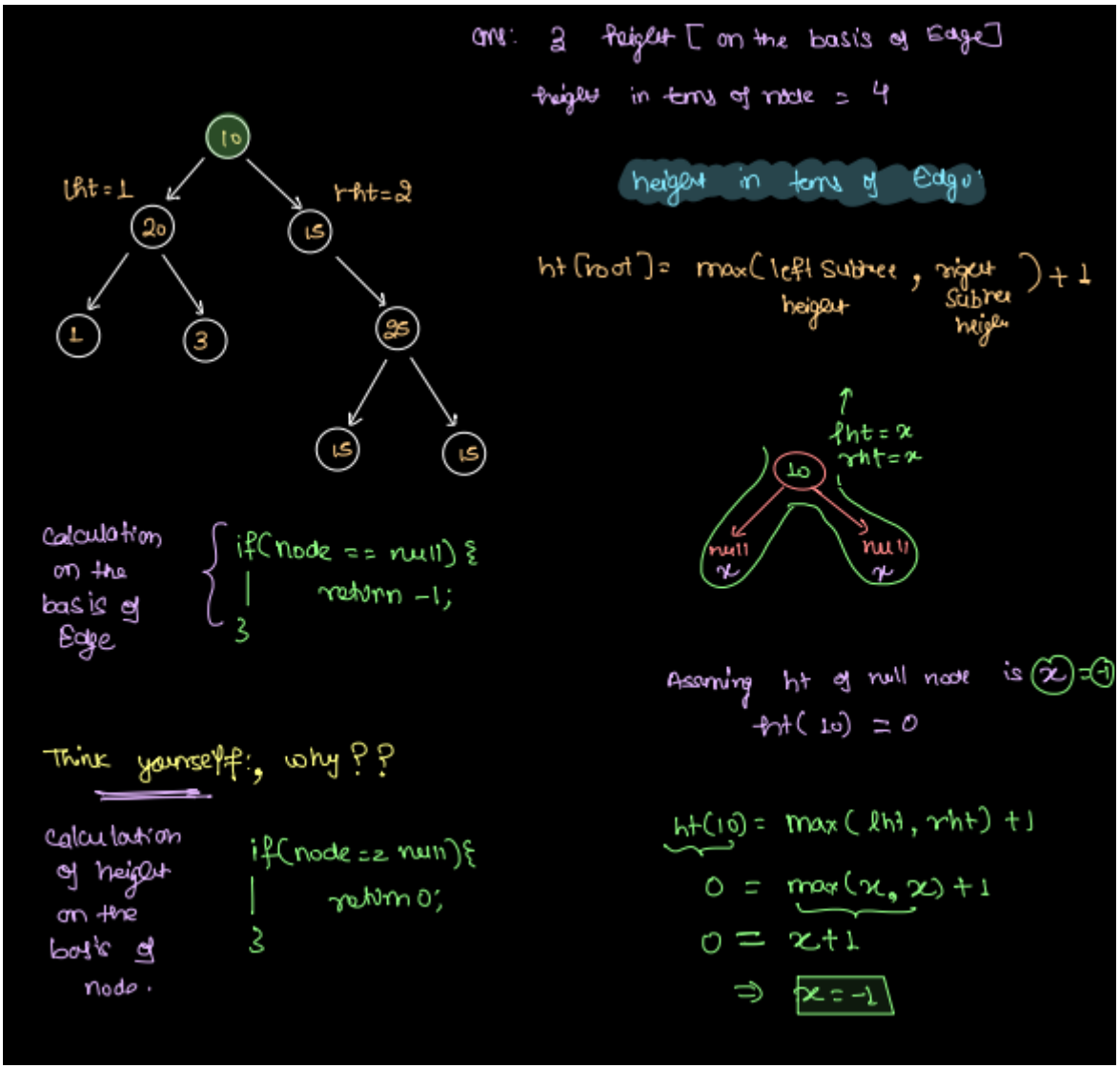
// Problem 2: Given root of binary tree, find sum of all nodes
https://www.interviewbit.com/snippet/91de4465c2293abc7140/

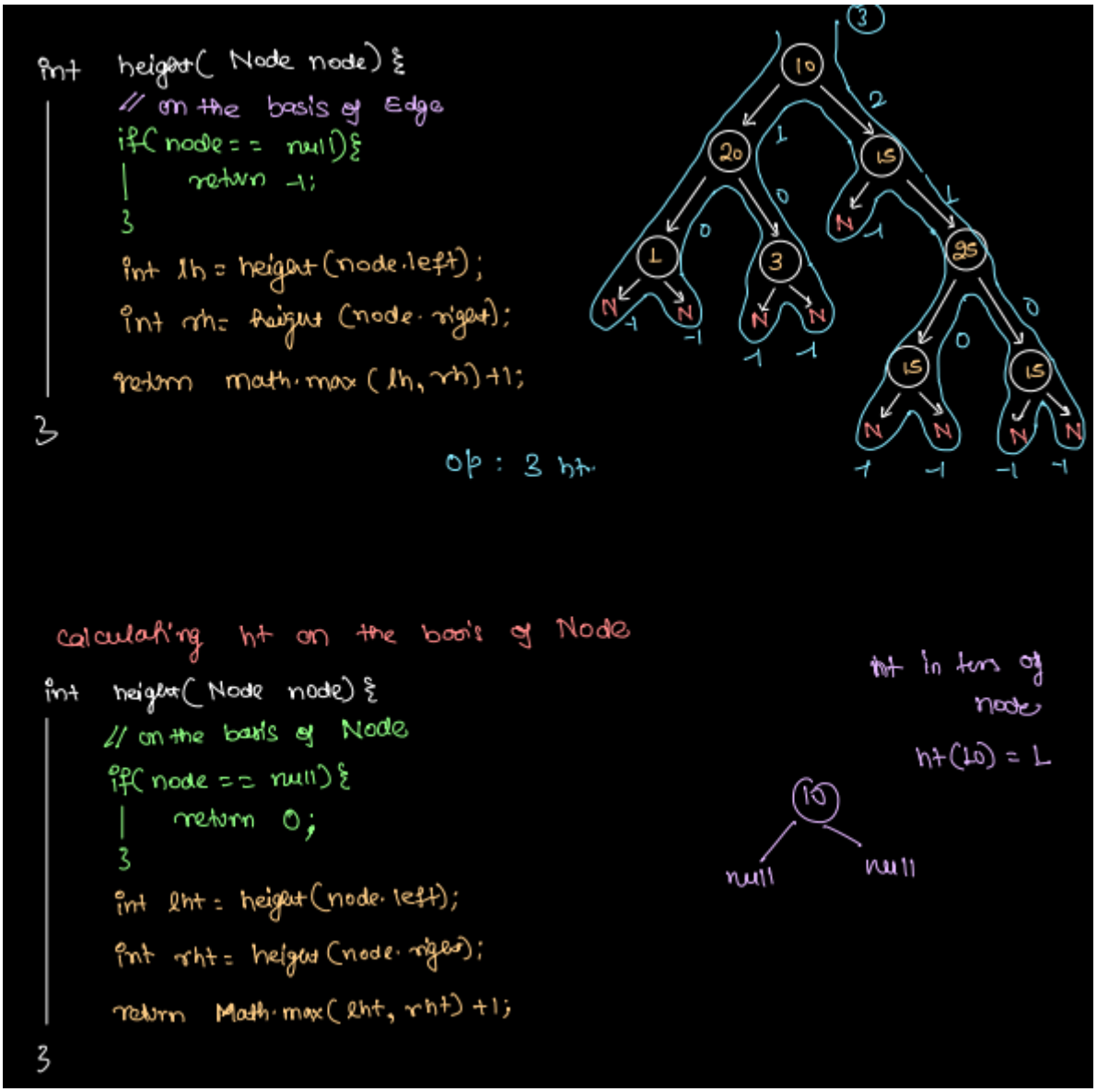




Problem 3 : Given root of Binary tree. Find height of binary tree. Hight of tree using Edge

https://www.interviewbit.com/snippet/ebf5b926b0894c2555f2/





DSA: Tree2 - 25 - July 2023

Iterative Inorder traversal
Construct Tree with Inorder and Preorder
Level Order printing
Some Problems using level Order
Left View
Right View + Extra Problems using level order

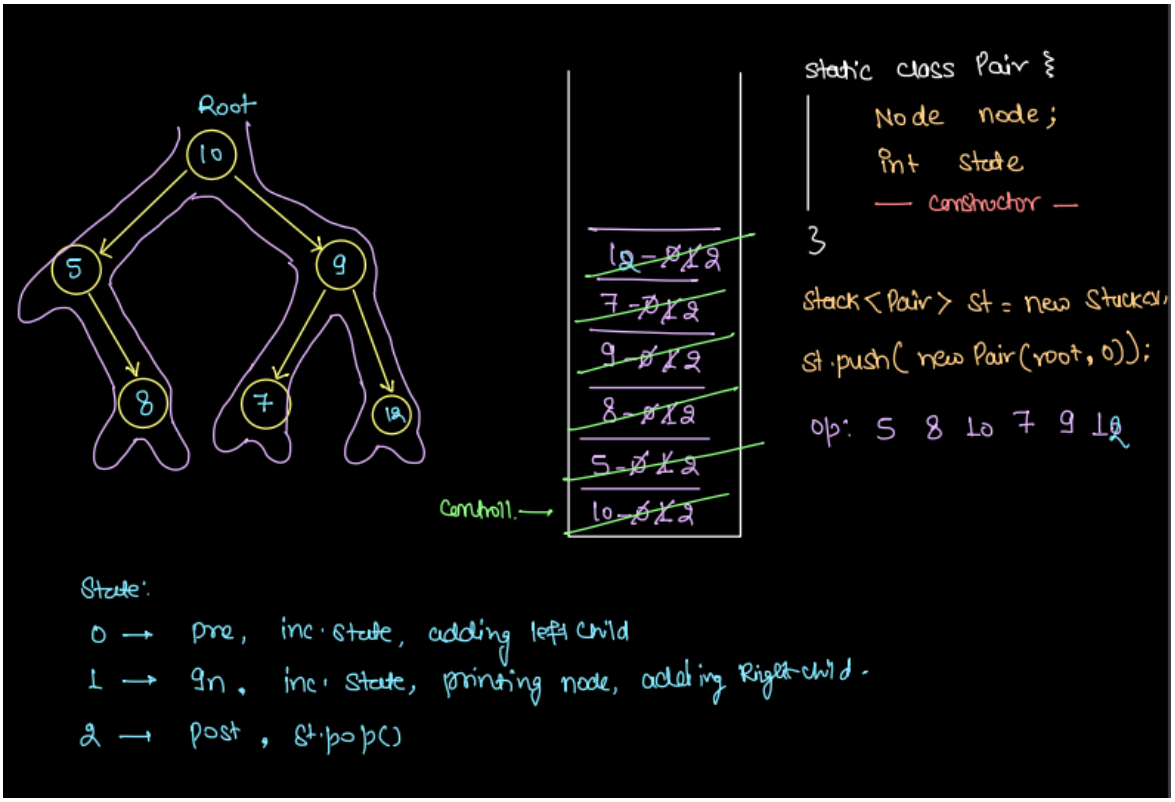
Iterative Inorder traversal
Problem1: Given root of a binary tree, return its inorder
Note:recursion is not allowed.
https://www.interviewbit.com/snippet/daa4b0889689d1eafaa9/

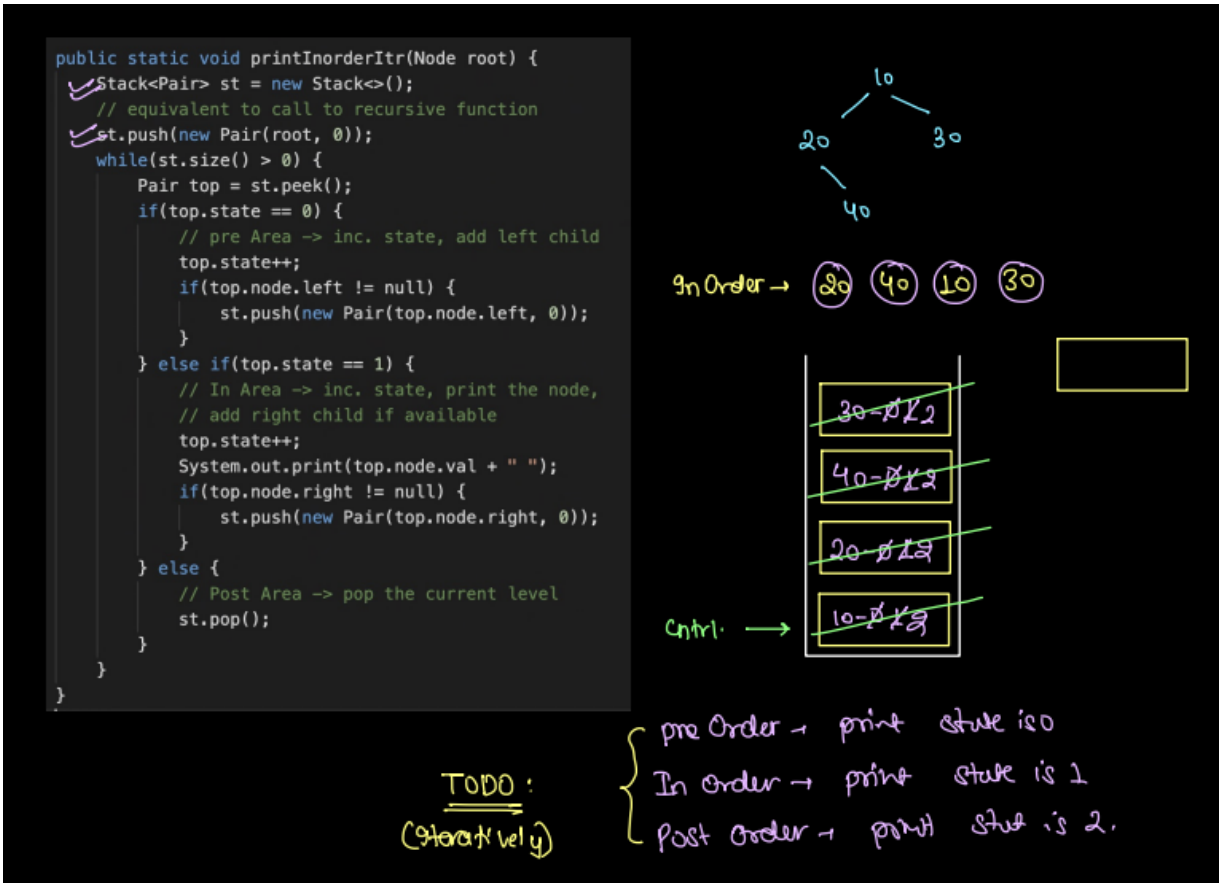
Main Logic ->

For iterative traversal, we use a stack and create 1 custom datatype class.

- Create TreeNode class with int val, TreeNode left, and TreeNode right. Initialize them in the constructor with this.val = val and this.left = this.right = null.
- Create Pair as a public static class with fields TreeNode node and int state. Initialize them in the constructor with this.node = node and this.state = state.

- Implement the iterativeInorderTraversal function and initialize a stack st with custom datatype Pair.
- Initially, push the node to st using st.push(new Pair(node, 0)).
- In a loop while st is not empty, do the following:
    - Initialize topNode with st.peek().
    - If topNode.state is 0, increase the state using topNode.state++, then check if topNode.node.left is not null, and if so, push a new Pair(topNode.node.left, 0) to the stack.
    - Else if topNode.state is 1, increase the state, print topNode.node.val, and check if topNode.node.right is not null. If so, push a new Pair(topNode.node.right, 0) to the stack. If not, pop from the stack.



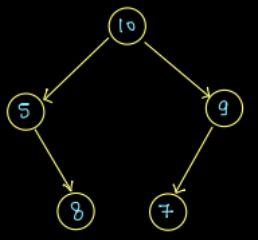


Construct Tree with Inorder and Preorder
Problem2: Construct a binary tree with given preorder and inorder and return root of binary tree.

Main logic ->

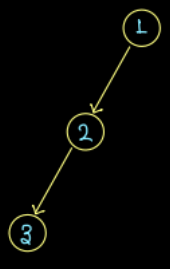
1. We will solve this problem using a recursive approach. We'll create a function called `createTreeHelper`, which returns a `TreeNode` object and takes the following parameters: `pre` (a preorder array), `pst` (start index of the preorder array), `ped` (end index of the preorder array), `in` (an inorder array), `ist` (start index of the inorder array), and `ied` (end index of the inorder array).
2. In order to construct the binary tree, we need to find the index of the current node in the inorder array `in[]`. We accomplish this by recursively dividing the main problem into smaller subproblems. The left part of the preorder array `pre[]` and the left part of the inorder array `in[]` before the node index represent the left nodes of the tree, while the right part of both arrays represents the right nodes of the tree.
3. The first step is to write the base condition for the recursion. If the start index of the preorder array (`pst`) is greater than the end index (`ped`), or the start index of the inorder array (`ist`) is greater than the end index (`ied`), we return `null` to indicate an empty subtree.
4. Next, we find the index of the current node in the inorder array (`in[]`). We initialize a variable called `nodeIndex` with the value of `pst`. We then use a loop that starts from `i = ist` and iterates until `i <= ied`. During each iteration, we check if the value of `pre[pst]` (the current node in the preorder array) is equal to the value of `in[i]` (the current node in the inorder array). If they match, we update `nodeIndex` to this value and break out of the loop.
5. Once we have the `nodeIndex`, we calculate the number of nodes in the left subtree by subtracting `ist` from `nodeIndex`, and we store this value in a variable called `count`.
6. Now, we recursively create the left subtree of the current node by calling `createTreeHelper` with updated parameters for the left subtree. We pass the start index for the preorder array as `pst + 1`, the end index as `pst + count`, the start index for the inorder array as `ist`, and the end index as `nodeIndex - 1`.
7. Similarly, we create the right subtree of the current node by calling `createTreeHelper` with updated parameters for the right subtree. The start index for the preorder array is `pst + count + 1`, the end index is `ped`, the start index for the inorder array is `nodeIndex + 1`, and the end index is `ied`.
8. Finally, we return the current node, which now has its left and right subtrees properly set.
9. To construct the entire binary tree, we call the `createTreeHelper` function within a `solve` function and pass initial parameters as follows: `createTreeHelper(pre[], 0, n-1, in[], 0, n-1)`.

Problem 2: Construct a binary tree with given preorder and inorder and return root of binary tree.



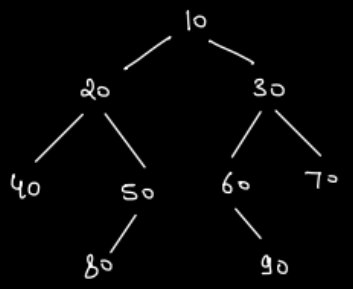
Pre: 10 5 8 9 7
0 1 2 3 4

In: 5 8 10 7 9
0 1 2 3 4



Pre: 1 2 3
0 1 2

In: 3 2 1
0 1 2



Node left Right

Pre: 10 20 40 50 80 | 30 60 90 70
0 1 2 3 4 | 5 6 7 8

→ Node, left, Right

Node

In: 40 20 80 50 10 60 90 30 70
0 1 2 3 4 5 6 7 8

→ left Node Right

pre[], in[]

pst, pend, ist, iend

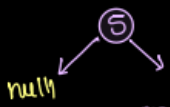
pre: 10 5 9 | 8 7
0 1 2 | 3 4
pst pend

pre[], in[]
pst = 1, pend = 2
inst = 0, inEnd = 1

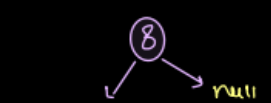
pre[], in[]
preSt = 3, preEnd = 4
inSt = 3, inEnd = 4

In: 5 9
0 1

In: 7 8
3 4

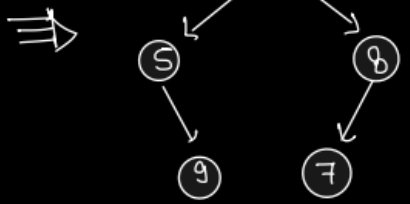


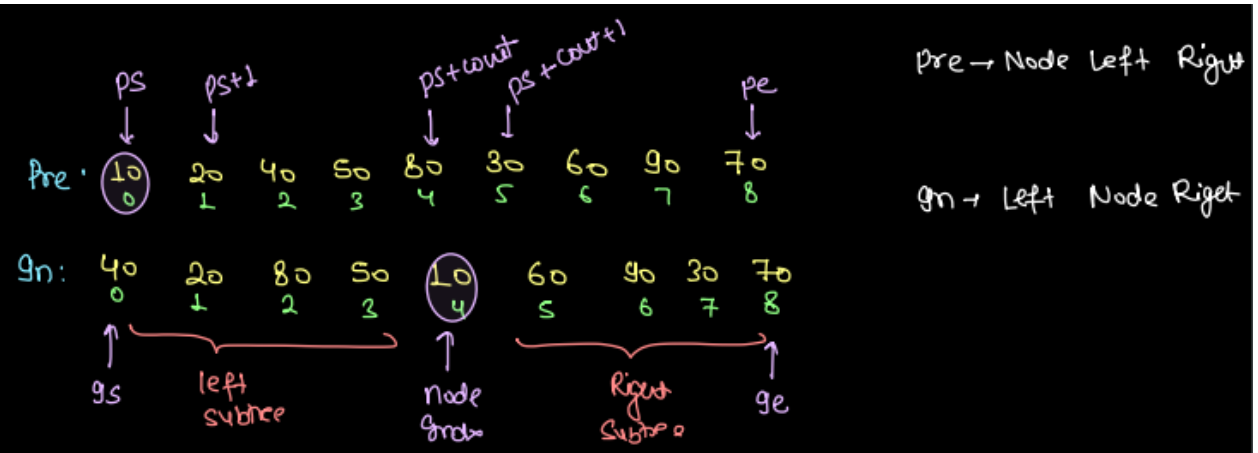
pre[], in[]
pst = 2, pend = 2
inst = 1, inEnd = 1
9



pre[], in[]
pst = 4 pend = 4
inst = 3 inEnd = 3
7

gn: 5 9 10 7 8
0 1 2 3 4
ist gndx iend



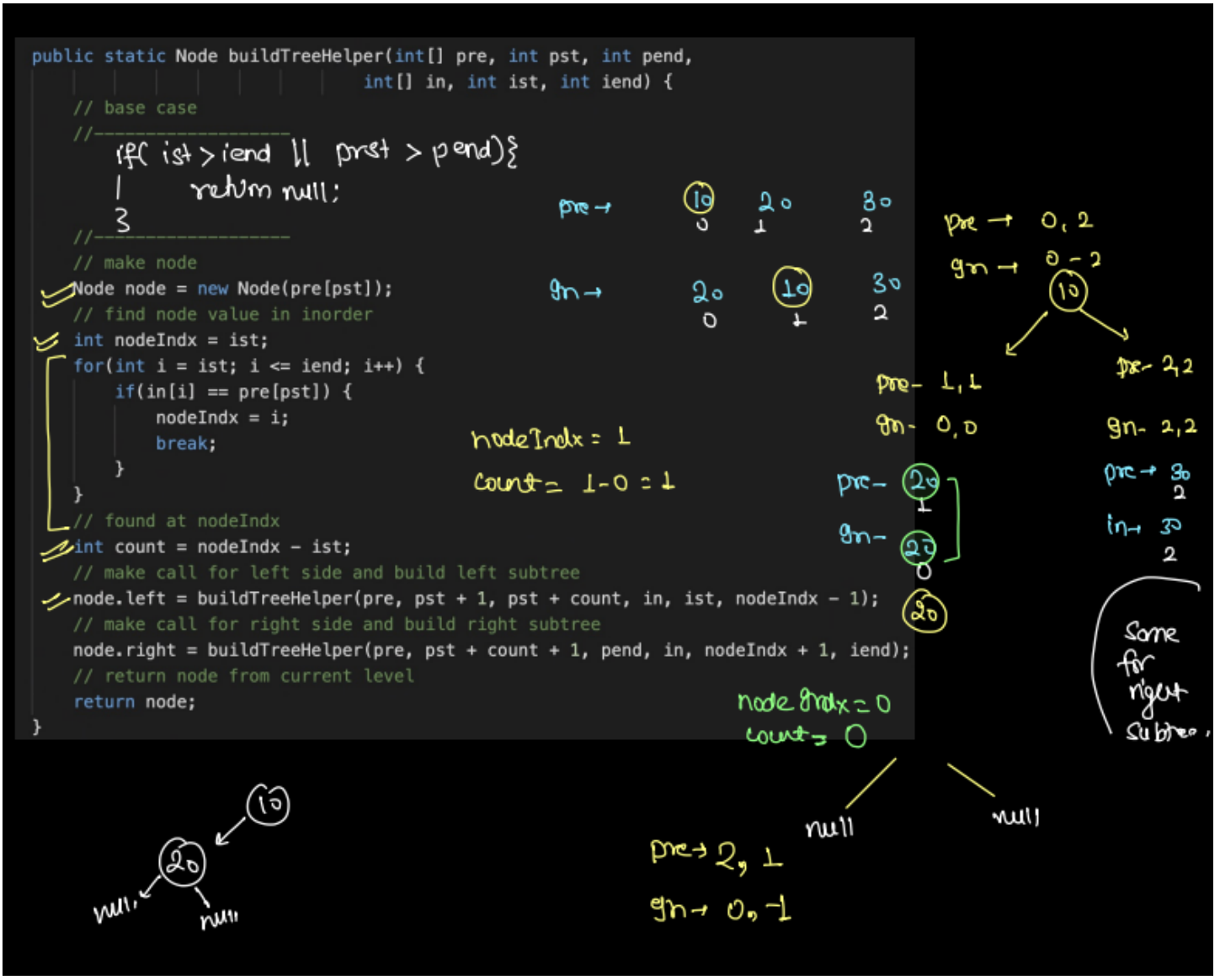


Pre → Node Left Right

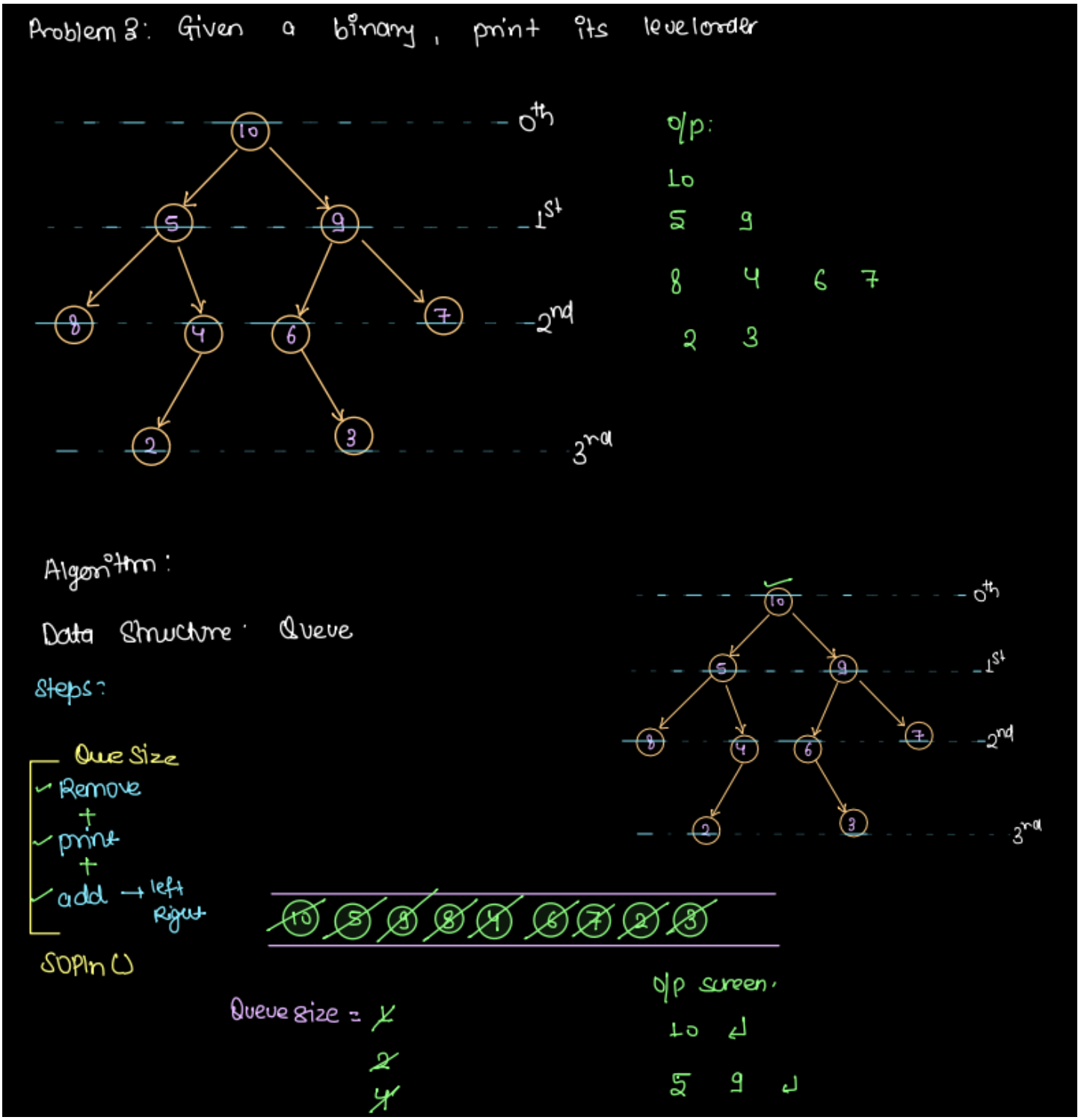
In → Left Node Right

```
Node createTree(int[] pre, int[] in, int ps, int pe, int is, int ie){

    pre[], in[]
    ps-pe, is-ie

    Node node = new Node(pre[ps]);
    // try to find node value in the inOrder array.
    // found node val at node index.
    int count = nodeIndx - is.

    node.left = createTree(pre, in, ps+1, ps+count, is, nodeIndx-1);

    node.right = createTree(pre, in, ps+count+1, pe, nodeIndx+1, ie);

    return node;

}
```

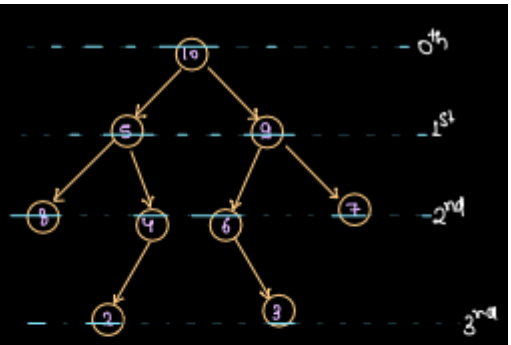
```java
public static Node buildTreeHelper(int[] pre, int pst, int pend,
                                   int[] in, int ist, int iend) {
    // base case
    //--------------------
    if( ist > iend || pst > pend){
        return null;
    }
    //--------------------
    // make node
    Node node = new Node(pre[pst]);
    // find node value in inorder
    int nodeIndx = ist;
    for(int i = ist; i <= iend; i++) {
        if(in[i] == pre[pst]) {
            nodeIndx = i;
            break;
        }
    }
    // found at nodeIndx
    int count = nodeIndx - ist;
    // make call for left side and build left subtree
    node.left = buildTreeHelper(pre, pst + 1, pst + count, in, ist, nodeIndx - 1);
    // make call for right side and build right subtree
    node.right = buildTreeHelper(pre, pst + count + 1, pend, in, nodeIndx + 1, iend);
    // return node from current level
    return node;
}
```



Problem3 : Given a binary, print its levelorder



```
Queue<Node> qu = new ArrayDeque<>();
qu.add(node);
while(qu.size()>0){
    int sz = qu.size();
    for(int i=0; i<sz; i++){
        // Remove
        // print
        // add left - Right child
    }
    SOPln();
}
```



sz = 1 2 4 2

o/p screen

10 ↵

5 9 ↵

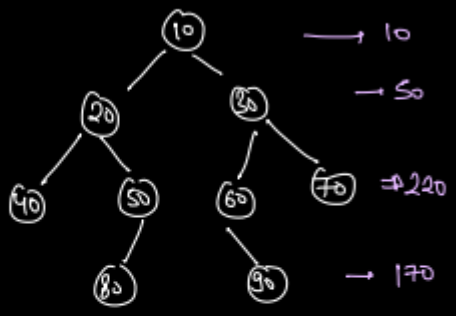
8 4 6 7 ↵

2 3 ↵

```
public static void printLevelOrder(Node root) {
    Queue<Node> qu = new ArrayDeque<>();
    qu.add(root);
    while(qu.size() > 0) {
        // pritn single level
        int sz = qu.size();
        for(int i = 0; i < sz; i++) {
            // remove
            Node remNode = qu.remove();
            // print
            System.out.print(remNode.val + " ");
            // add left and right child
            if(remNode.left != null) {
                qu.add(remNode.left);
            }
            if(remNode.right != null) {
                qu.add(remNode.right);
            }
        }
        // hit an enter to change the line for next level
        System.out.println();
    }
}
```

helpful

level wise sum



→ 10

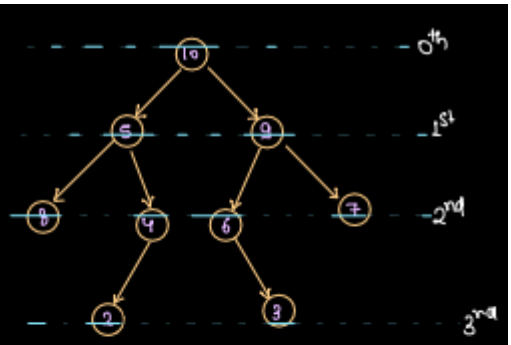
→ 20 30

→ 40 50 60 70

→ 80 90

10 50 220 170

```
Queue<Node> qu = new ArrayDeque<>();
qu.add(node);
while(qu.size()>0){
    int sz = qu.size();
    for(int i=0; i<sz; i++){
        // Remove
        // print
        // add left - Right child
    }
    SOPln();
}
```



sz = 1 2 4 2

o/p screen

10 ↵

5 9 ↵

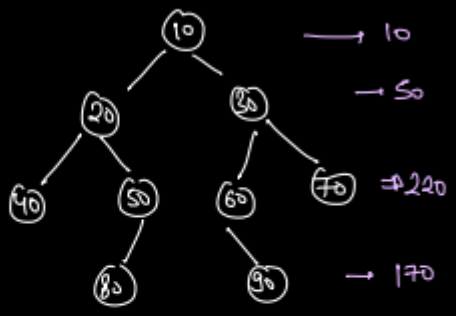
8 4 6 7 ↵

2 3 ↵

```
public static void printLevelOrder(Node root) {
    Queue<Node> qu = new ArrayDeque<>();
    qu.add(root);
    while(qu.size() > 0) {
        // pritn single level
        int sz = qu.size();
        for(int i = 0; i < sz; i++) {
            // remove
            Node remNode = qu.remove();
            // print
            System.out.print(remNode.val + " ");
            // add left and right child
            if(remNode.left != null) {
                qu.add(remNode.left);
            }
            if(remNode.right != null) {
                qu.add(remNode.right);
            }
        }
        // hit an enter to change the line for next level
        System.out.println();
    }
}
```

helpful

level wise sum



→ 10

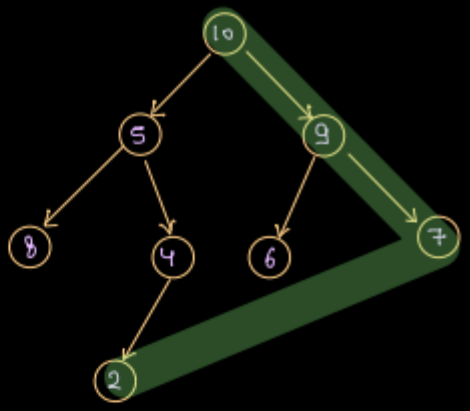
→ 20 30

→ 40 50 60 70

→ 80 90

10 50 220 170

Right View of Binary Tree:



Right View = 10 9 7 2

Hint:

Instead of print in level wise function, Update last Element variable by removed value.

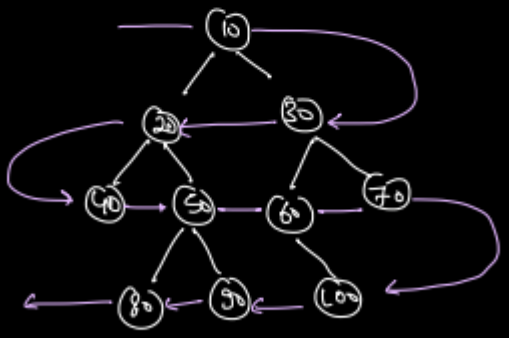
TODO: Dry Run + Code.

level Order

1. level Order traversal
2. level-wise sum
3. height (with help of level order)
4. left view
5. Right view
6. Zig-zag traversal

Small problem → count node, sum of tree, min in tree, max in tree

(To Do)



10

30 20

40 50 60 70

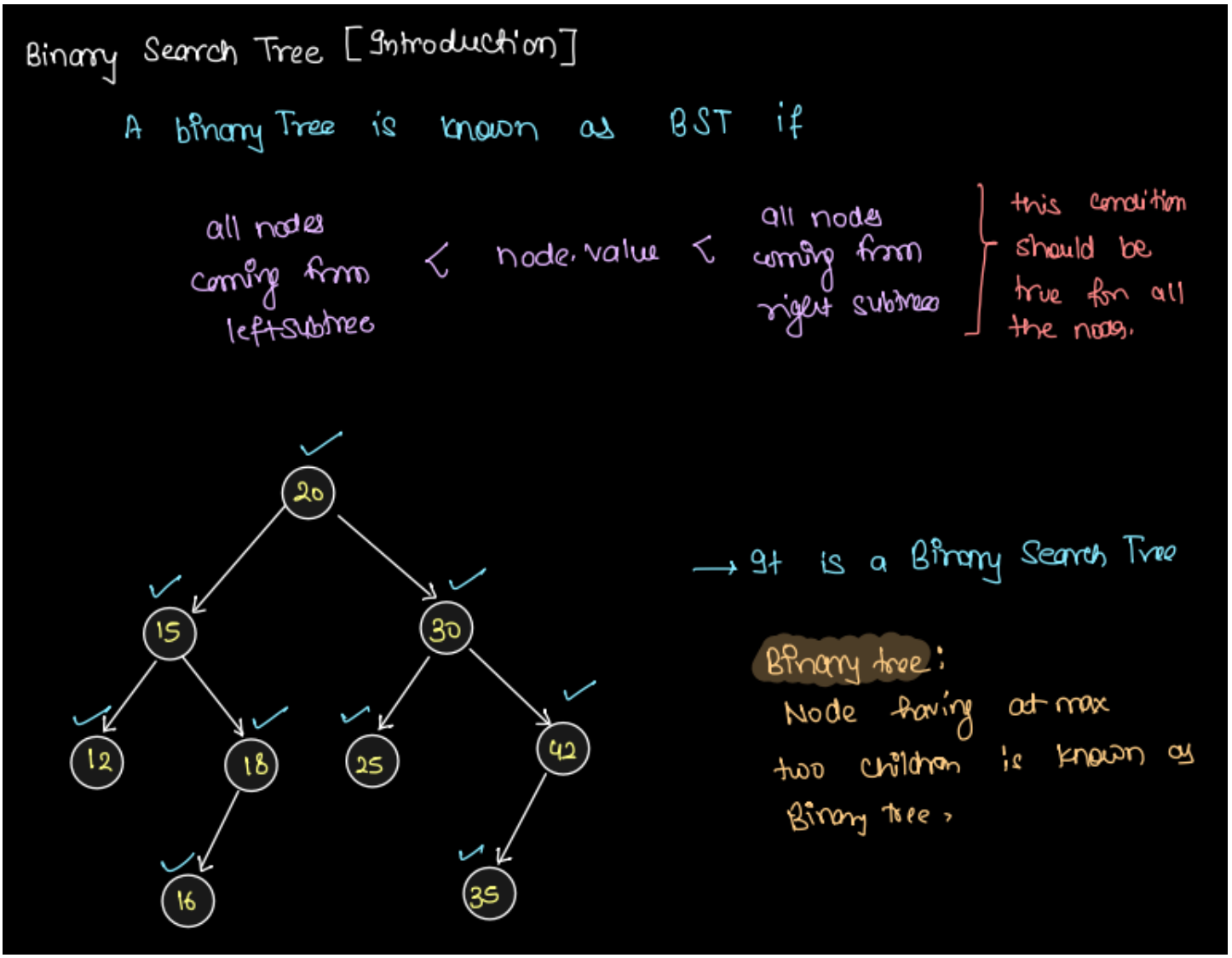
100 90 80

DSA: Trees 3 - BST - 1 - Aug 2023

- Introduction of BST
- Properties of BST
- BT and BST : Comparison
- Search in BST
- Insert K in BST
- Check if BT is BST?
- Construct BST from sorted Array

Introduction of BST -> A binary tree is known as BST if all nodes coming from left structure < node.value < all nodes coming from right subtree. This condition sholud be true for all the nodes.

A binary tree can be a BST but all BST can be a binary tree.



Properties of BST -> Inorder of BST is always sorted

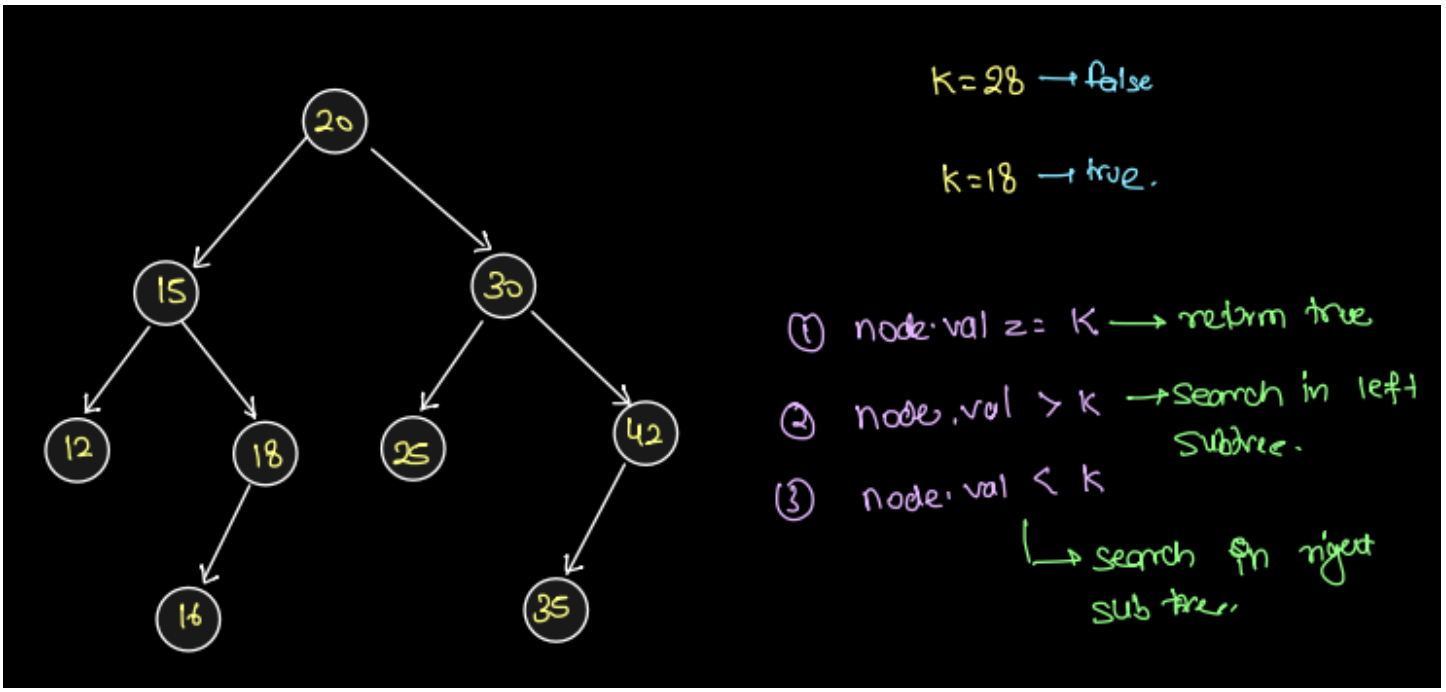
- BT and BST : Comparison ->
BT (Binary tree) -
- No specific order in values.
- Searching in BT TC: O(n) and SC: O(hieght)

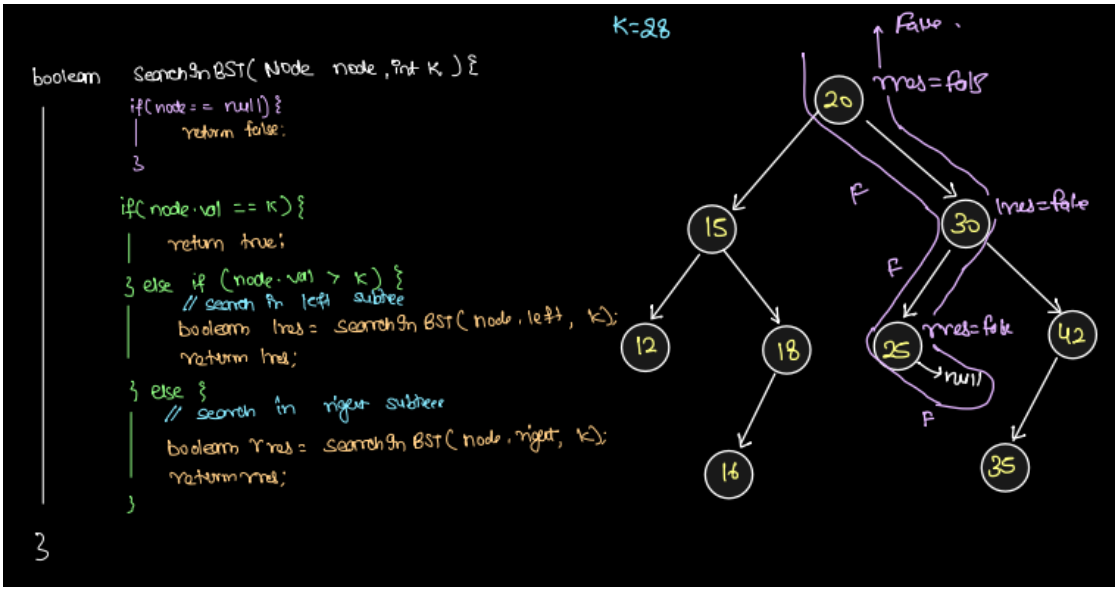
BST (Binary search tree) -
- Thre is an order of values **Left < Node < Right.**
- Searching in BST TC: O(hieght) and SC: O(hieght)

Problem1: Given root node of a BST, search if K exist or not.
https://www.interviewbit.com/snippet/ef45c8ab596b40ffbfdb/

Main logic ->
There is simple logic if K is grater then traverse tree to left or right and if node.value == k return 1 elase base case return 0.

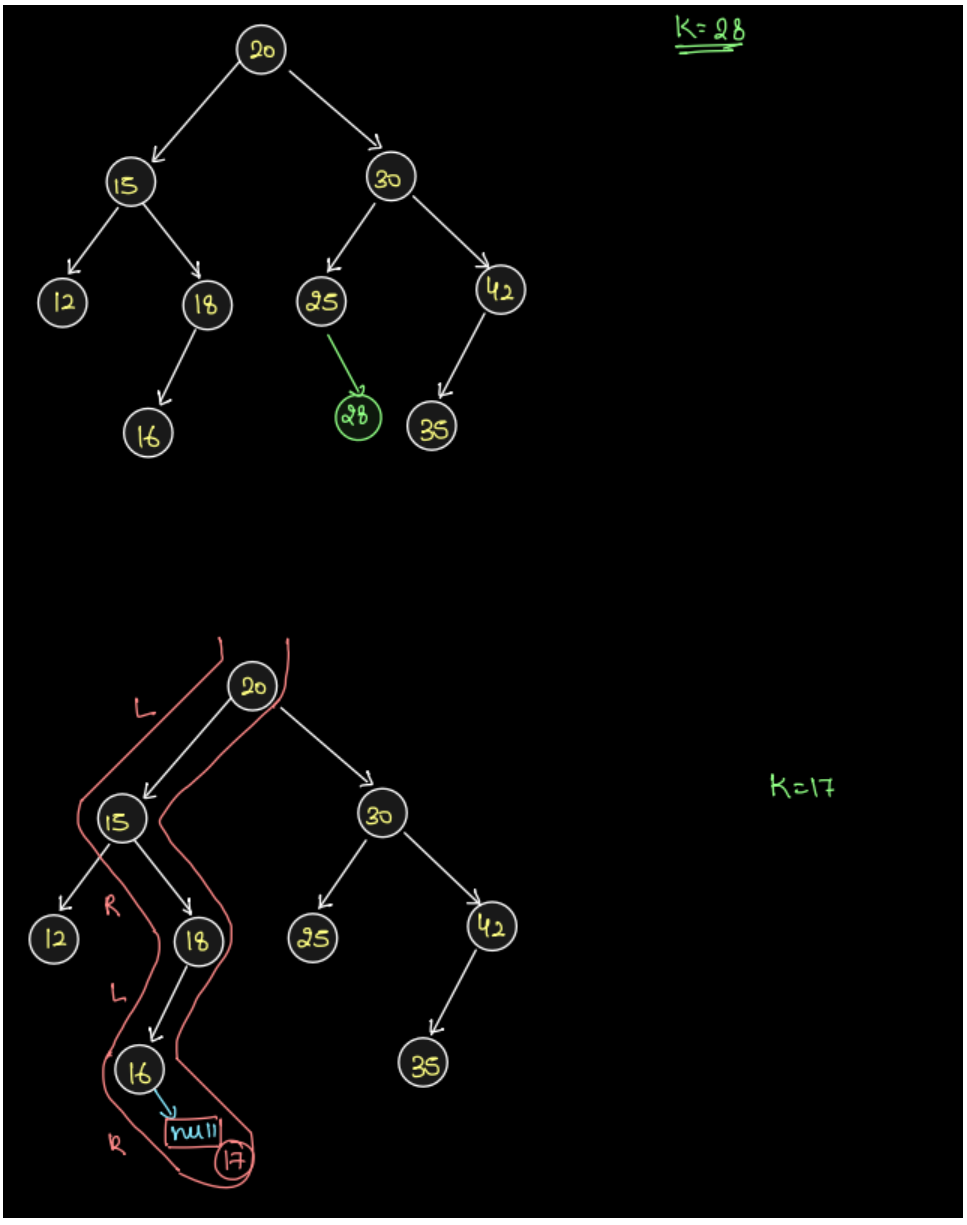
- Write a function searchBST
- Write a base case condition node == null return 0
- write a main condition
  - If node.val == k return 1.
  - if node.val > k return searchBST pass node.left;
  - else return searchBST pass node.right;

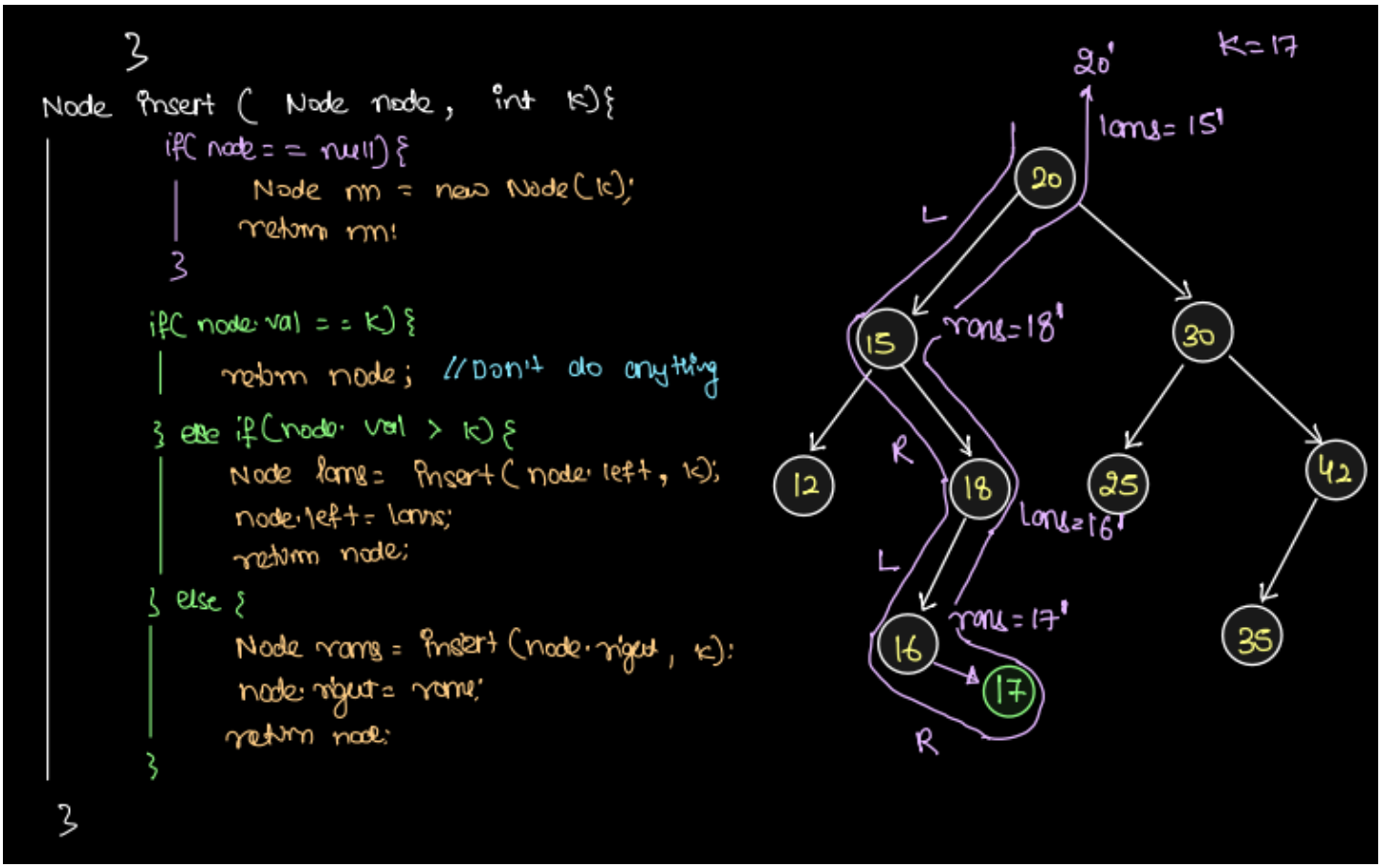




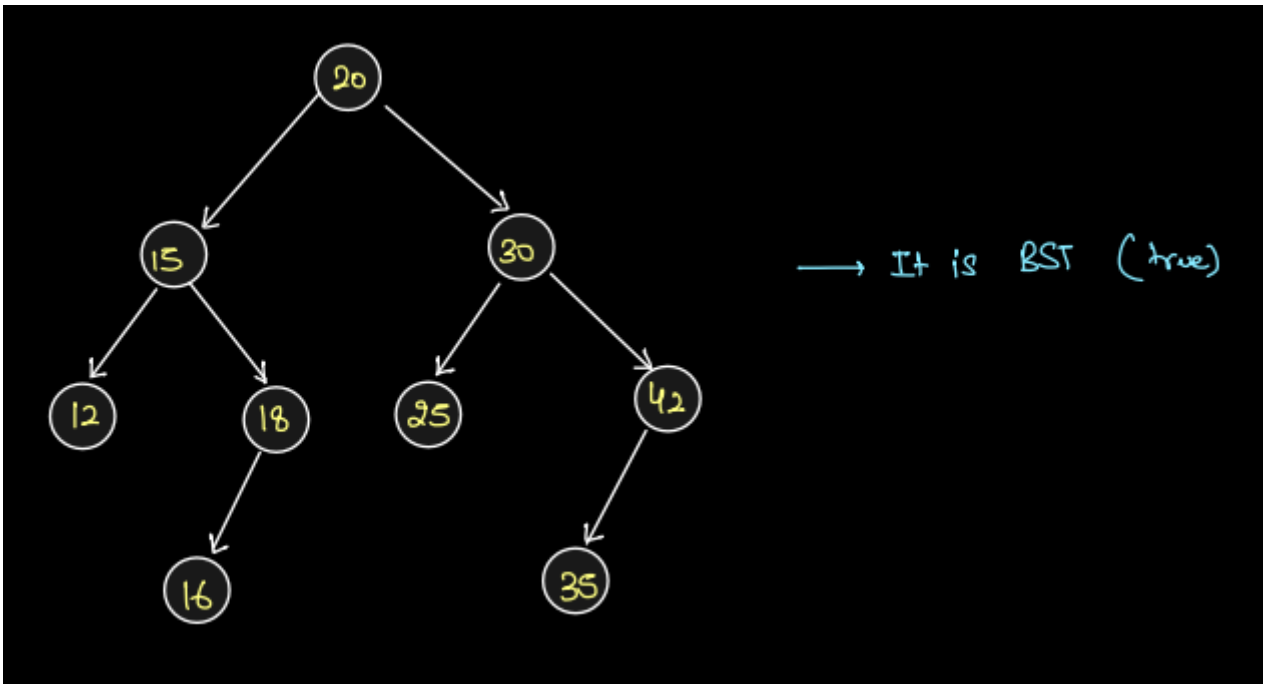
Problem2: Given root of BST, insert node with data k in the BST. Inserting should be done without suffling the existing node.
https://www.interviewbit.com/snippet/203a71c71aa7c58c119b/
Main Logic ->

- Create a function called insertKInBST.
- Write the base condition: If the node is null, create a new node with the value 'k', assign it to 'B', and return it.
- Write the main condition:
  - If the value of the current node is equal to 'k', return the node.
  - Otherwise, if the value of the current node is greater than 'k', create a variable 'lAns' and assign the result of calling the 'insertKInBST' function with the arguments 'node.left' and 'k'. Then, assign 'lAns' to 'node.left' and return the node.
  - Otherwise, create a variable 'rAns' and assign the result of calling the 'insertKInBST' function with the arguments 'node.right' and 'k'. Then, assign 'rAns' to 'node.right' and return the node.





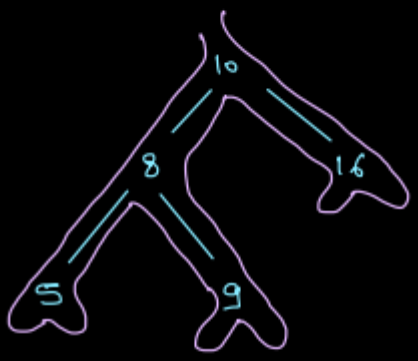
Problem3: Given root of a binary tree, check if it is BST or not?
https://www.interviewbit.com/snippet/a37c3d832295740a7c76/



Idea: * Iterate on entire tree and fill inorder in an arraylist.

* check if that list is sorted or not.
* If it is sorted then tree is BST otherwise not a BST.

```
ArrayList<Integer> list = new ArrayList<>();
void traversal(Node node){
    if(node == null) {
        return;
    }
    traversal(node.left);
    list.add(node.val);
    traversal(node.right);
}
```



5 8 9 10 16

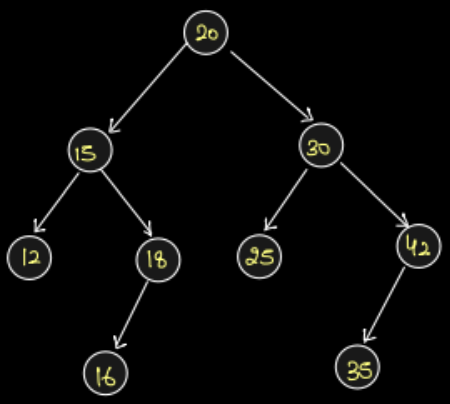
Check if it is sorted or not.

T.C: O(n) + O(n) = O(n)
traversal checking

S.C: O(h) + O(n) = O(n)
Recursive space, list space

can we solve it using

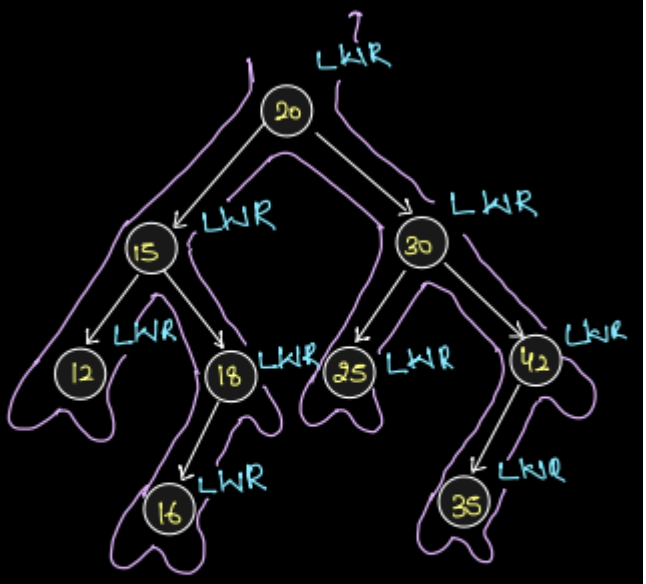
can we solve it using recursive space only ??



In: L < N < R

if(prev < Node.val){
→ inorder is valid
} else {
→ ans = false

```
boolean ans = true;
int prev = -∞
void traversal(Node node) {
    if(node == null) {
        return;
    }
    L [traversal(node.left);
    if(prev > node.val) {
        ans = false;
        return;
    }
    W
    prev = node.val;
    R [traversal(node.right);
}
```

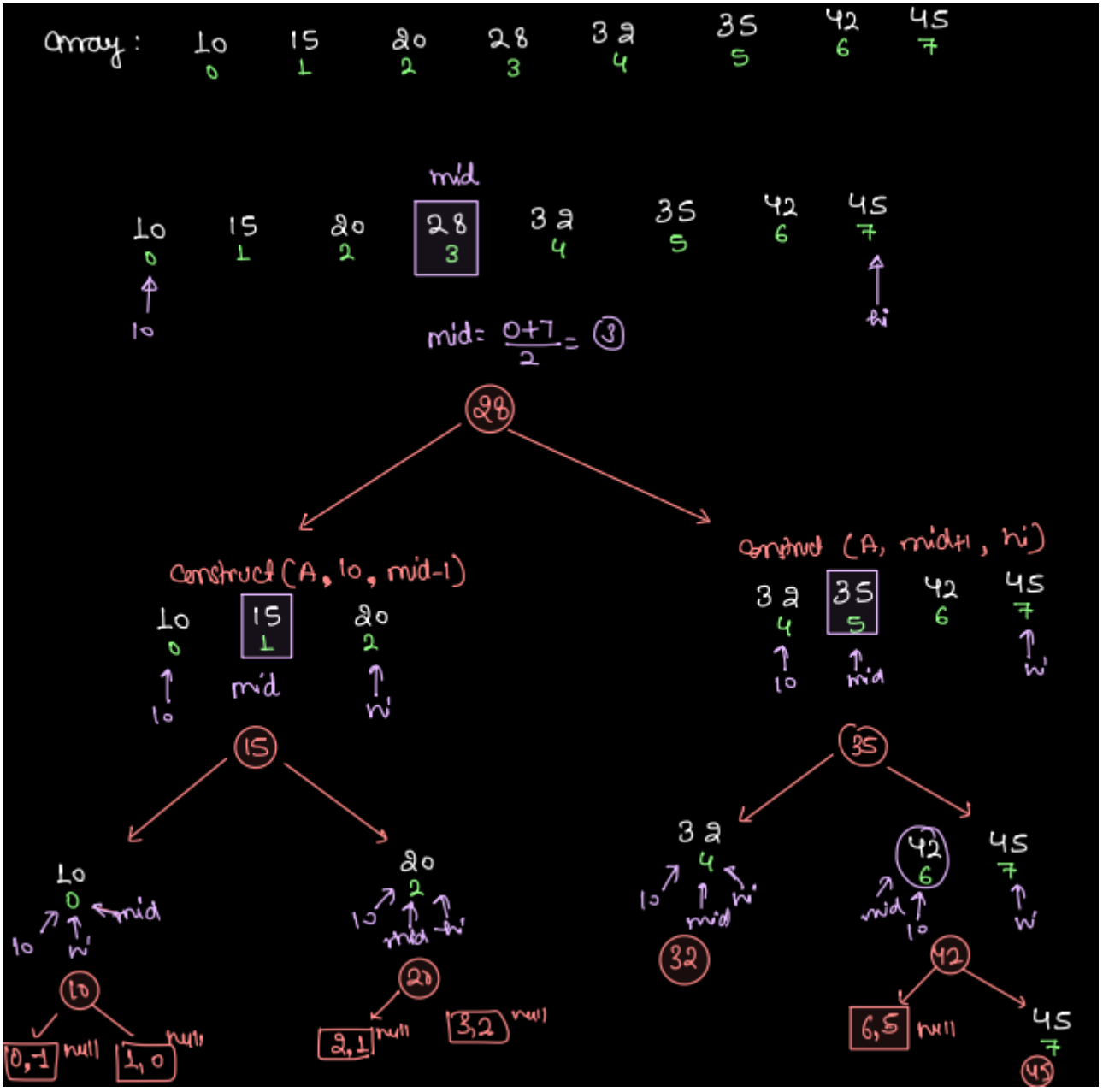


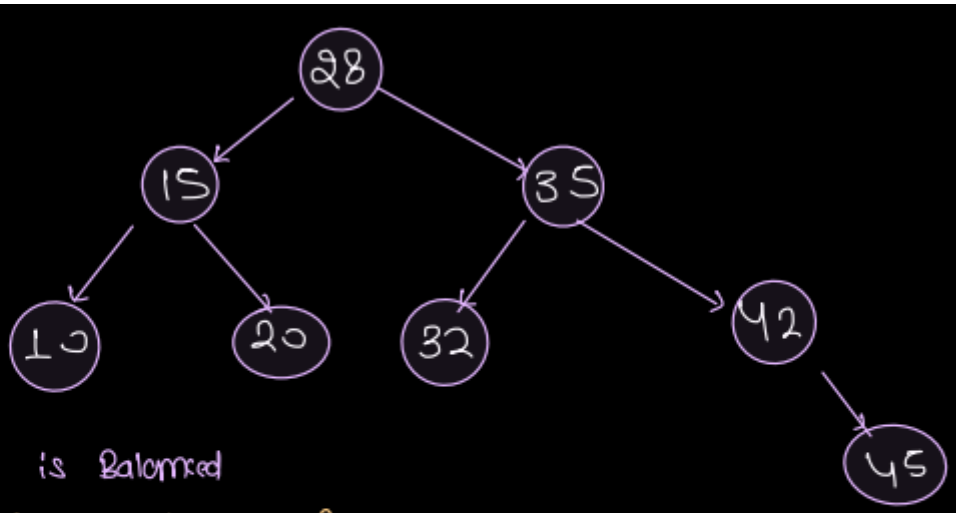
prev = -∞ 12 15 16 18 20 25 30 35 42

ans = true

```java
static int prev;
static boolean isbst;

public static void isBST_Helper(Node node) {
    if(node == null) {
        return;
    }

    isBST_Helper(node.left);

    if(prev > node.val) {
        isbst = false;
        return;
    }
    prev = node.val;

    isBST_Helper(node.right);
}

public static boolean isBST(Node root) {
    prev = Integer.MIN_VALUE;
    isbst = true;

    isBST_Helper(root);
}   return isbst;
```

Problem4: Given a sorted array. Construct balance BST using this array and return its root node.
https://www.interviewbit.com/snippet/c99ec3c9803330f6da81/



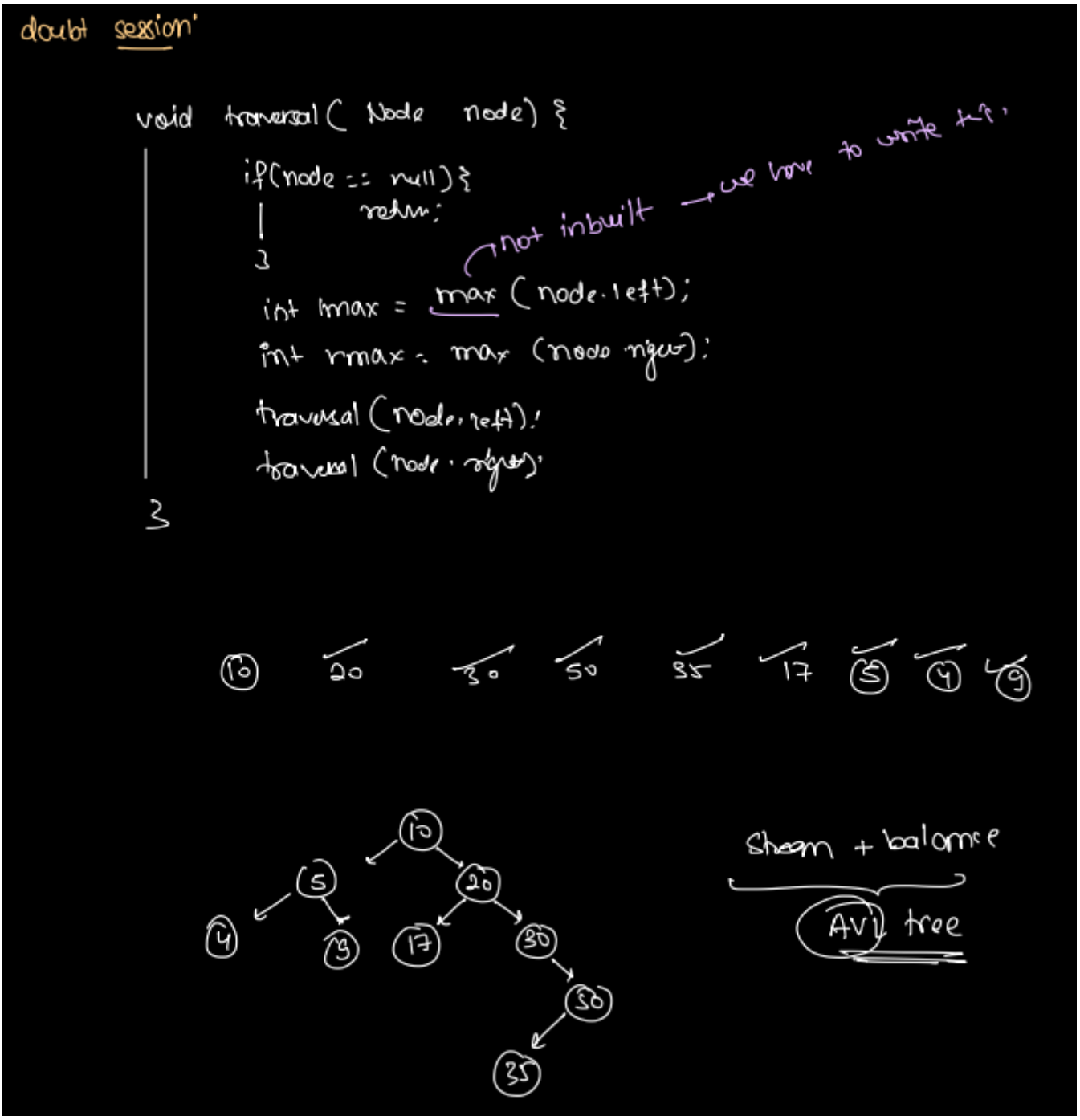


#How it is Balanced

→ We are finding mid node from array to make to root node at every level.

```
Node solve(int[] arr) {
    return construct(A, 0, A.length-1);
}

Node construct(int[] arr, int lo, int hi) {
    if(lo > hi) {
        return null;
    }
    int mid = (lo+hi)/2;
    Node nn = new Node(arr[mid]);
    nn.left = construct(A, lo, mid-1);
    nn.right = construct(A, mid+1, hi);
    return nn;
}
```

get MAx

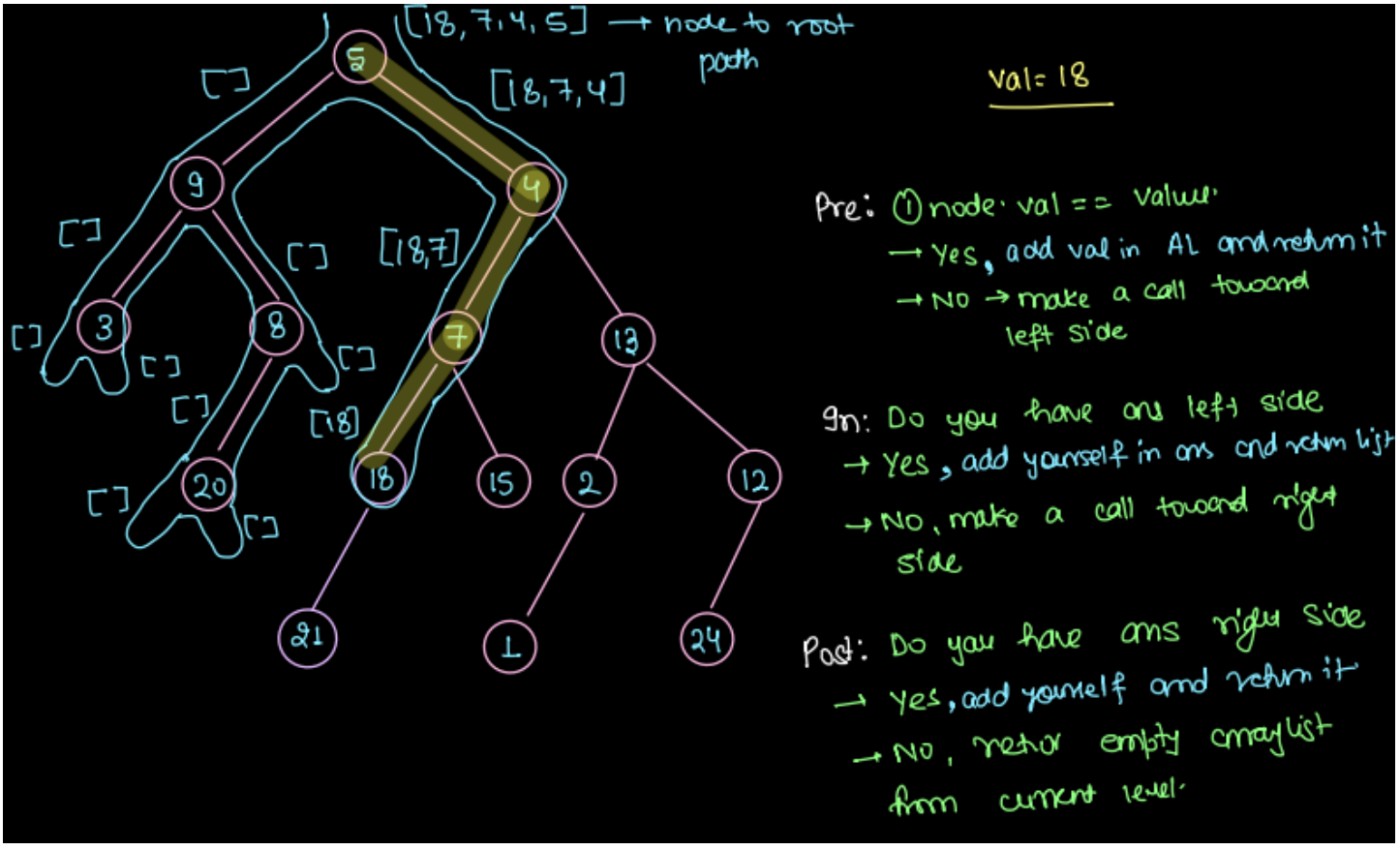

doubt session
void traversal (Node node) {
if(node == null){
return;
}
int lmax = max (node.left);
int rmax = max (node.right);
traversal (node.left);
traversal (node.right);
}
not inbuilt → we have to write it?
10 20 30 50 35 17 5 4 9
10
5
20
4
9
17
30
50
35
Steam + balance
AVL tree

DSA: Trees 4 - BST - 3 - Aug 2023

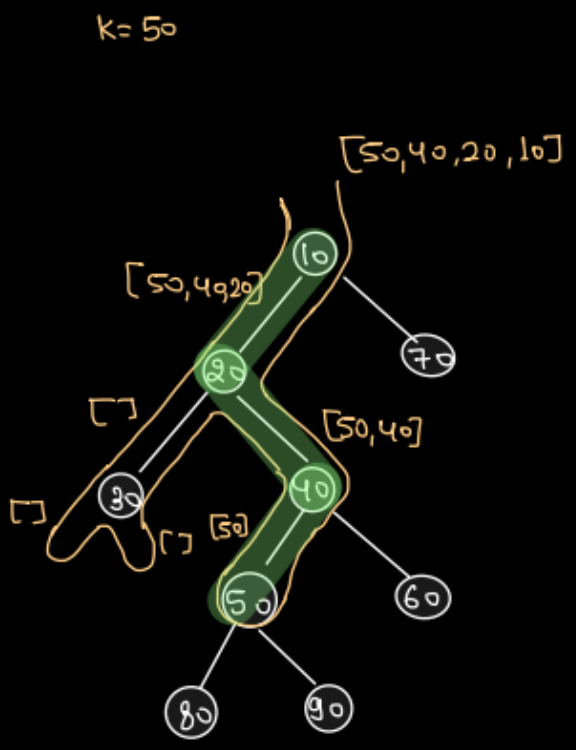
- Lowest Common Ancestor
- K Down (Helper for K Far)
- Kfar with help of K down

Given a root Node of Binary Tree and a Value, Find node to root path.
Note all value in BT are unique.
https://www.interviewbit.com/snippet/ffe93d0278be50ab9189/
Main Logic ->

- "We will start by creating a function called 'findRootPath' with a return type of 'ArrayList' and two parameters: 'node' and 'val'.
- First, we'll handle the base case where if 'node' is null, the function will return an empty ArrayList.
- Next, in the pre-area of the function, we'll check if 'node.val' is equal to 'val'. If it is, we'll create a new ArrayList, add 'node' to it, and return the ArrayList.
- Then, we'll create another ArrayList called 'lList' and assign it the result of calling the 'findRootPath' function with 'node.left' as the parameter.
- In the pre-area, we'll check if the size of 'lList' is greater than 0. If it is, we'll add 'node' to 'lList' and return 'lList'.
- After that, we'll create a new ArrayList called 'rList' and assign it the result of calling the 'findRootPath' function with 'node.right' as the parameter.
- In the post-area, we'll check if the size of 'rList' is greater than 0. If it is, we'll add 'node' to 'rList' and return 'rList'.
- Finally, in the post-area, after all conditions have been checked, we'll create one more empty ArrayList and return it."



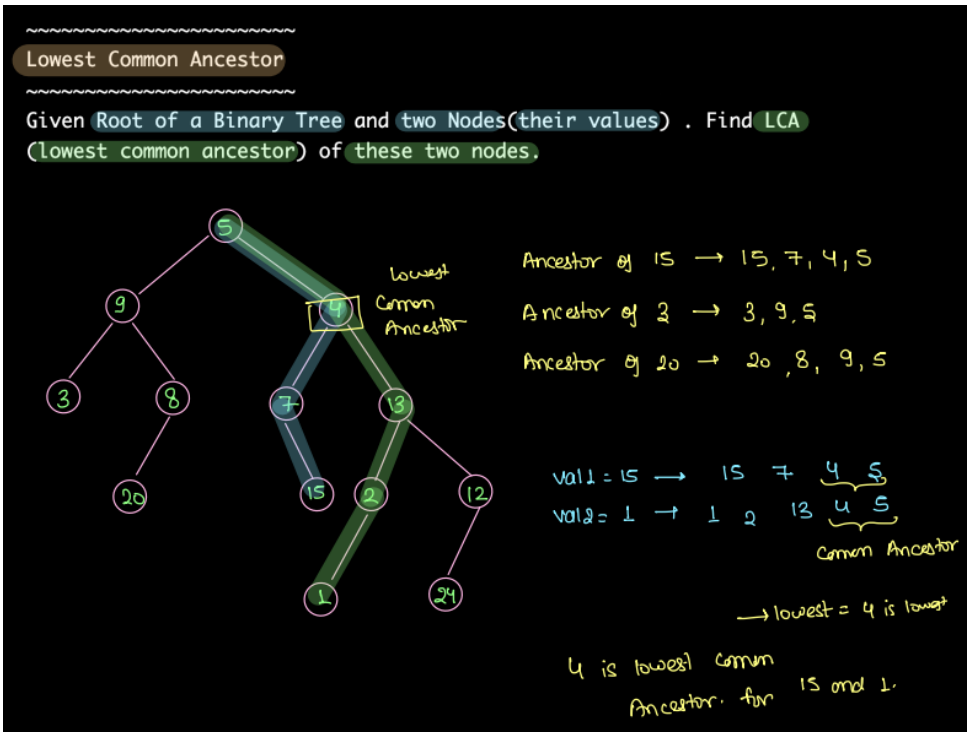
```java
static ArrayList<Node> nodeToRootPath(Node node, int K) {
    // Base Case
    if(node == null) {
        // ArrayList<Node> bres = new ArrayList<>();
        // return bres;
        // OR
        return new ArrayList<Node>();
    }
    // Pre - Area
    if(node.val == K) {
        ArrayList<Node> list = new ArrayList<>();
        list.add(node);
        return list;
    }
    ArrayList<Node> lans = nodeToRootPath(node.left, K);
    // In - Area
    if(lans.size() > 0) {
        lans.add(node);
        return lans;
    }
    ArrayList<Node> rans = nodeToRootPath(node.right, K);
    // Post - Area
    if(rans.size() > 0) {
        rans.add(node);
        return rans;
    }
    return new ArrayList<Node>();
}
```



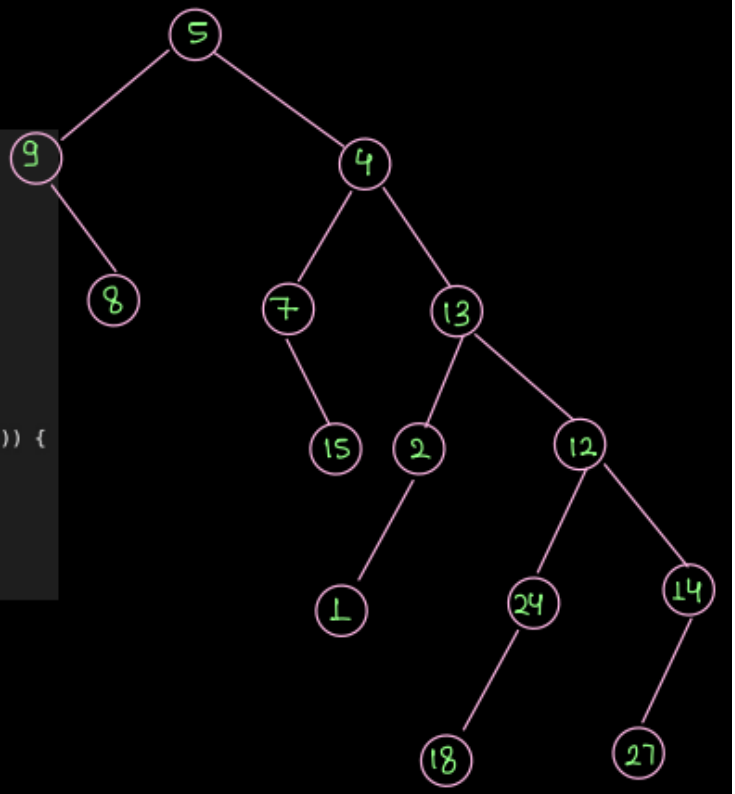
Lowest Common Ancestor

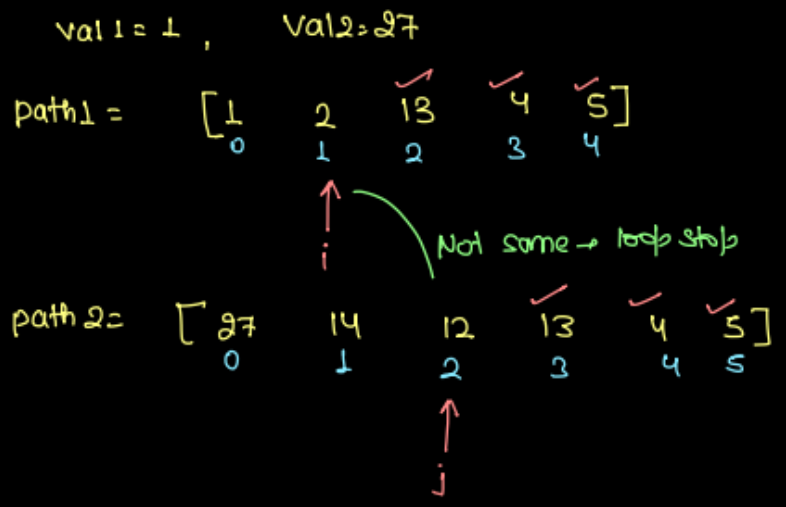
Given root of a binary tree and two Nodes(their values). Find LCA (lowest common ancestor) of these two nodes.

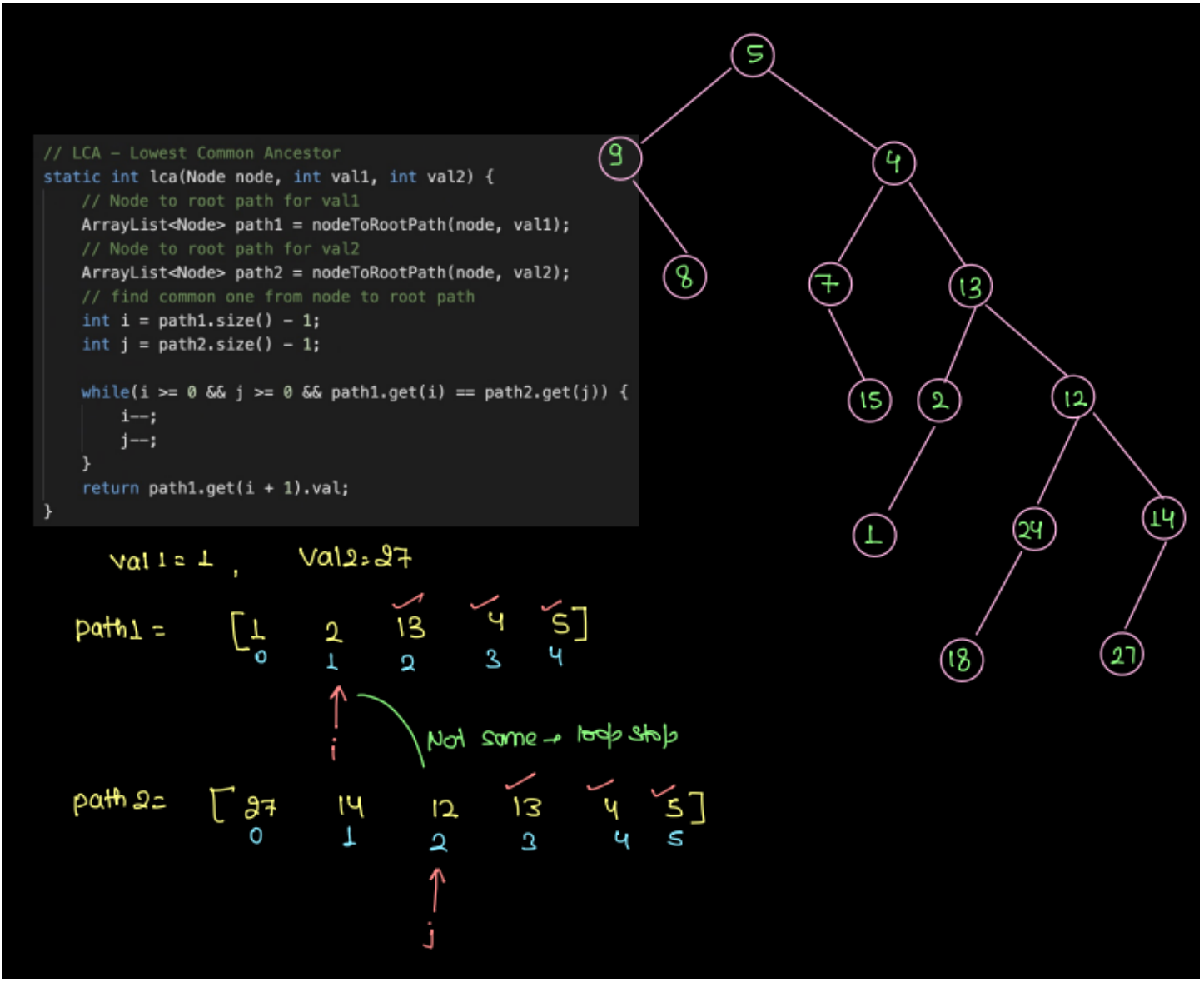
https://www.interviewbit.com/snippet/3c79aa221760879afeaa/



```java
// LCA - Lowest Common Ancestor
static int lca(Node node, int val1, int val2) {
    // Node to root path for val1
    ArrayList<Node> path1 = nodeToRootPath(node, val1);
    // Node to root path for val2
    ArrayList<Node> path2 = nodeToRootPath(node, val2);
    // find common one from node to root path
    int i = path1.size() - 1;
    int j = path2.size() - 1;

    while(i >= 0 && j >= 0 && path1.get(i) == path2.get(j)) {
        i--;
        j--;
    }
    return path1.get(i + 1).val;
}
```

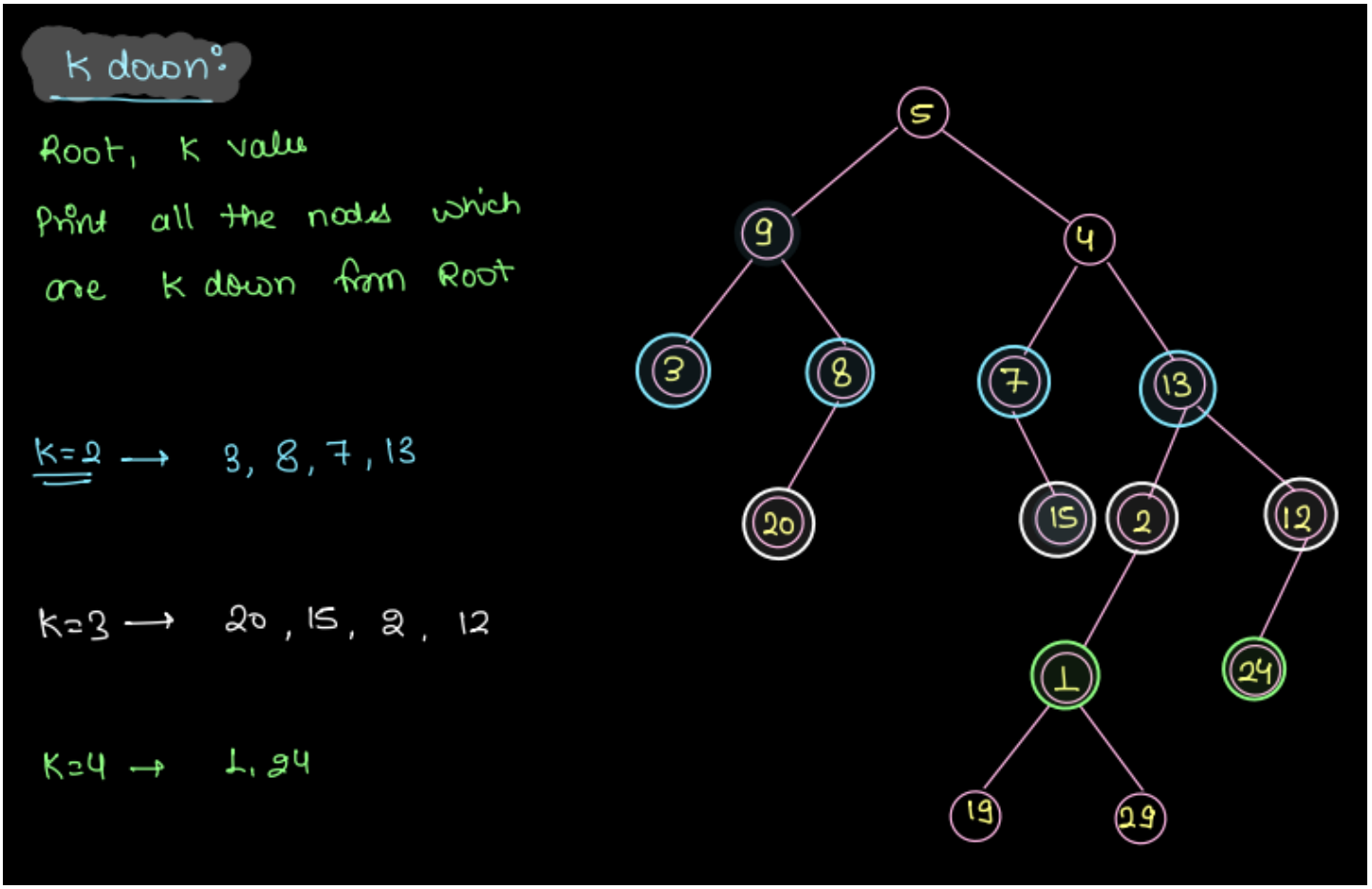


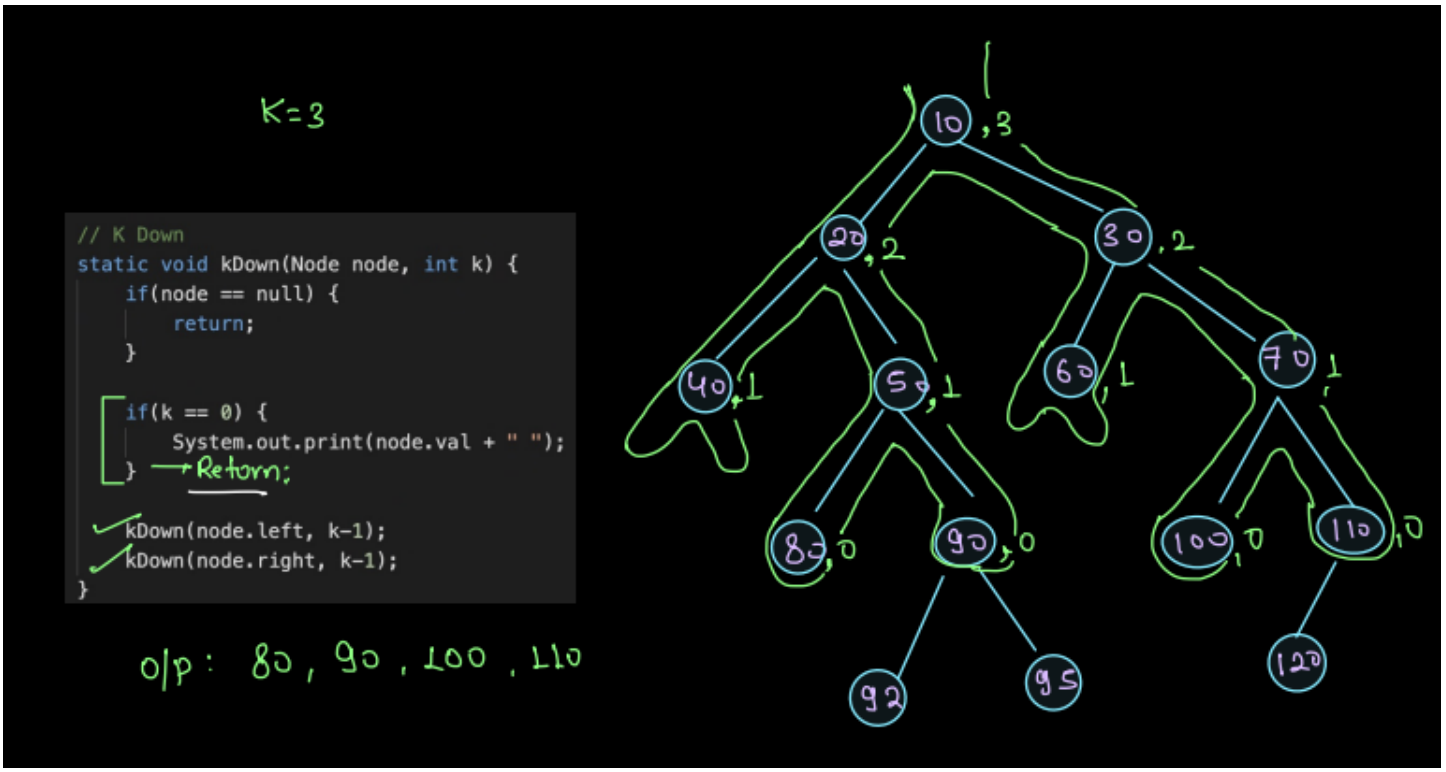




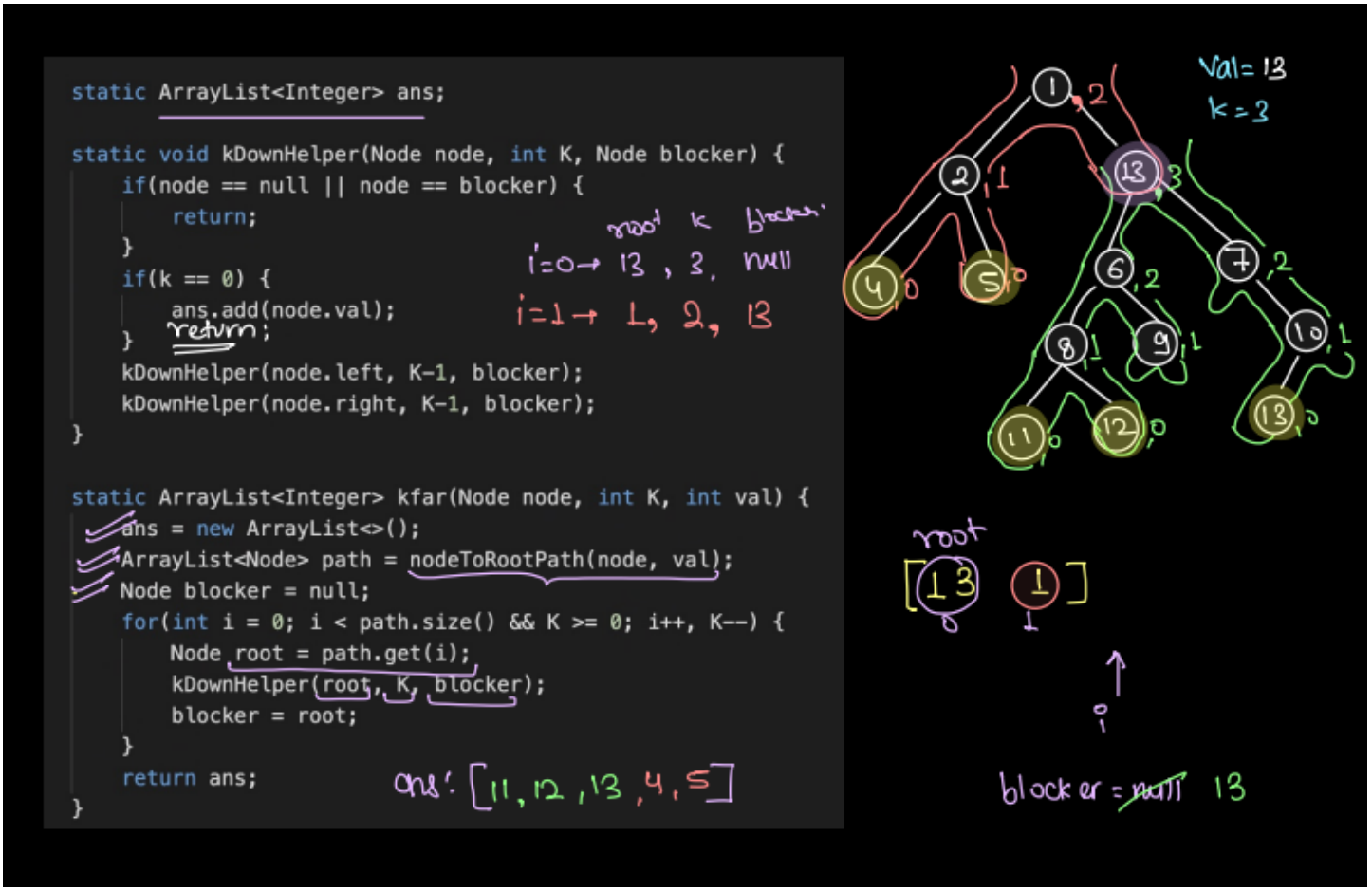
Find K Down

Root, k value, print all the nodes which are k down from root.





K-Far / K-Away
Given root of a binary tree, a value and a distance K.
Return ArrayList<Integer> consisting all the nodes that are k distance away from node containin the given value.



DSA: Trees 5 - 6 - Aug 2023

- Diameter of Binary tree
- Diameter Bruteforce
- Diameter in height function
- Serialisation and Deserialisation
- Intro and Usage of TreeMap

Problem1 - Diameter of Binary Tree
Given root of binary tree, find it's diameter.
Note:
The diameter of binary tree is the length of the longest path between any two nodes. this path may or may not pass through the root.

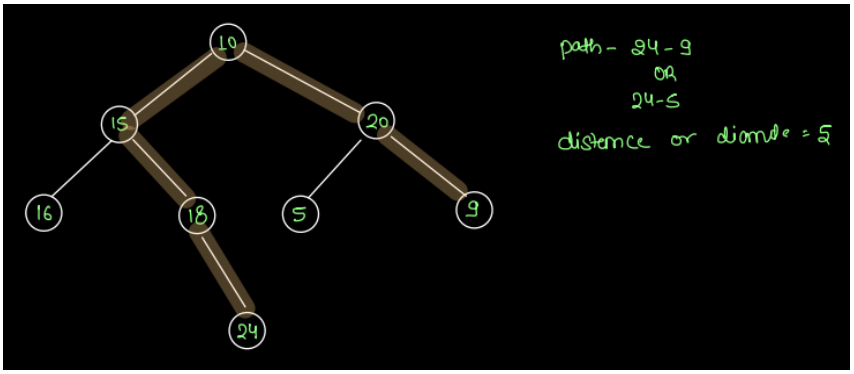
Main logic ->
Farthest distance between two nodes is called diameter.

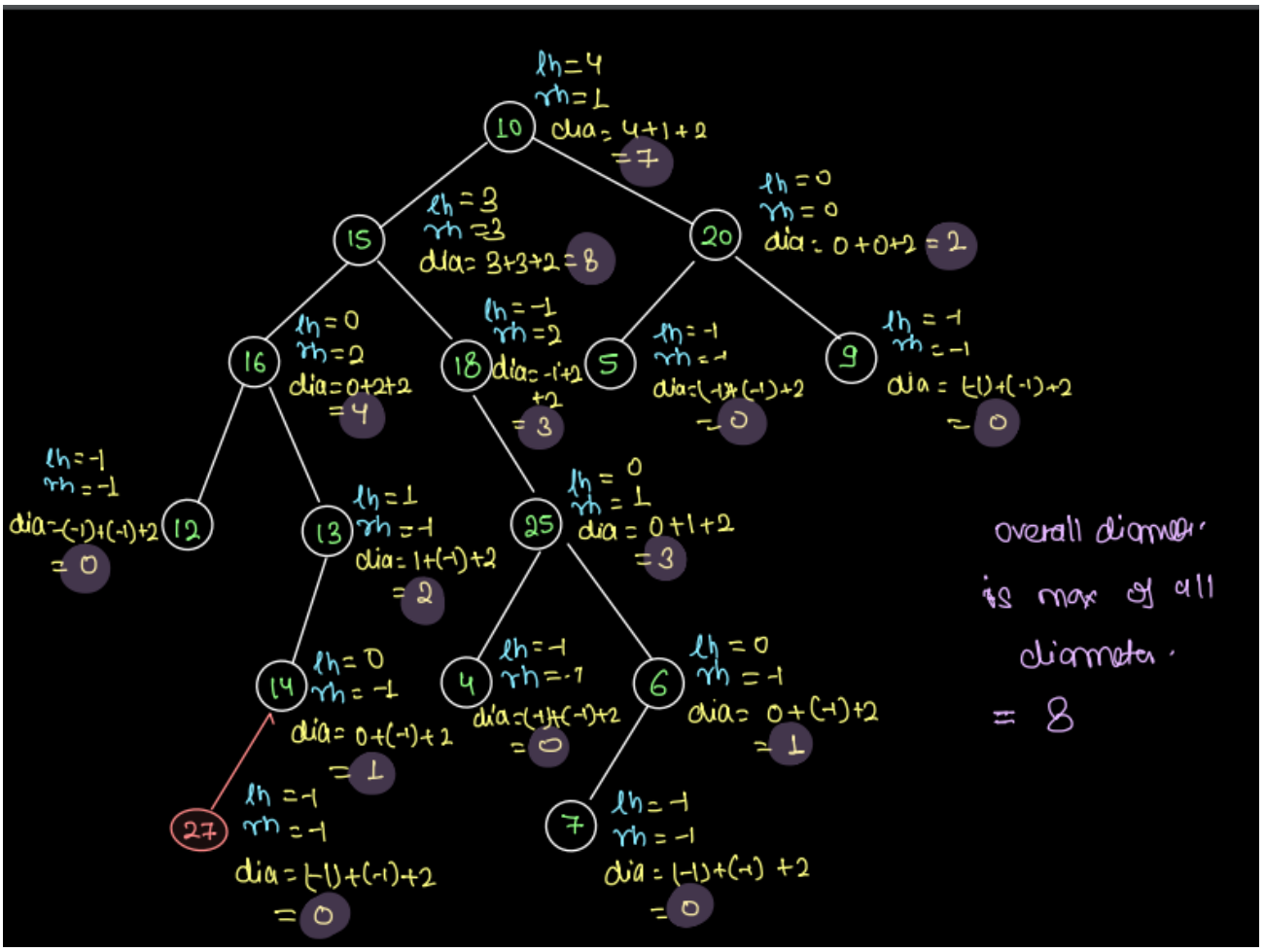
for getting a dimeter of subtree we will get left subtree height and right subtree height and add + 2 to it.
left subtree height menas next node of any node in left and distance from that node tofarthest leaf node.

right subtree height means next node of any node in right and distance from that node to farthest leaf node

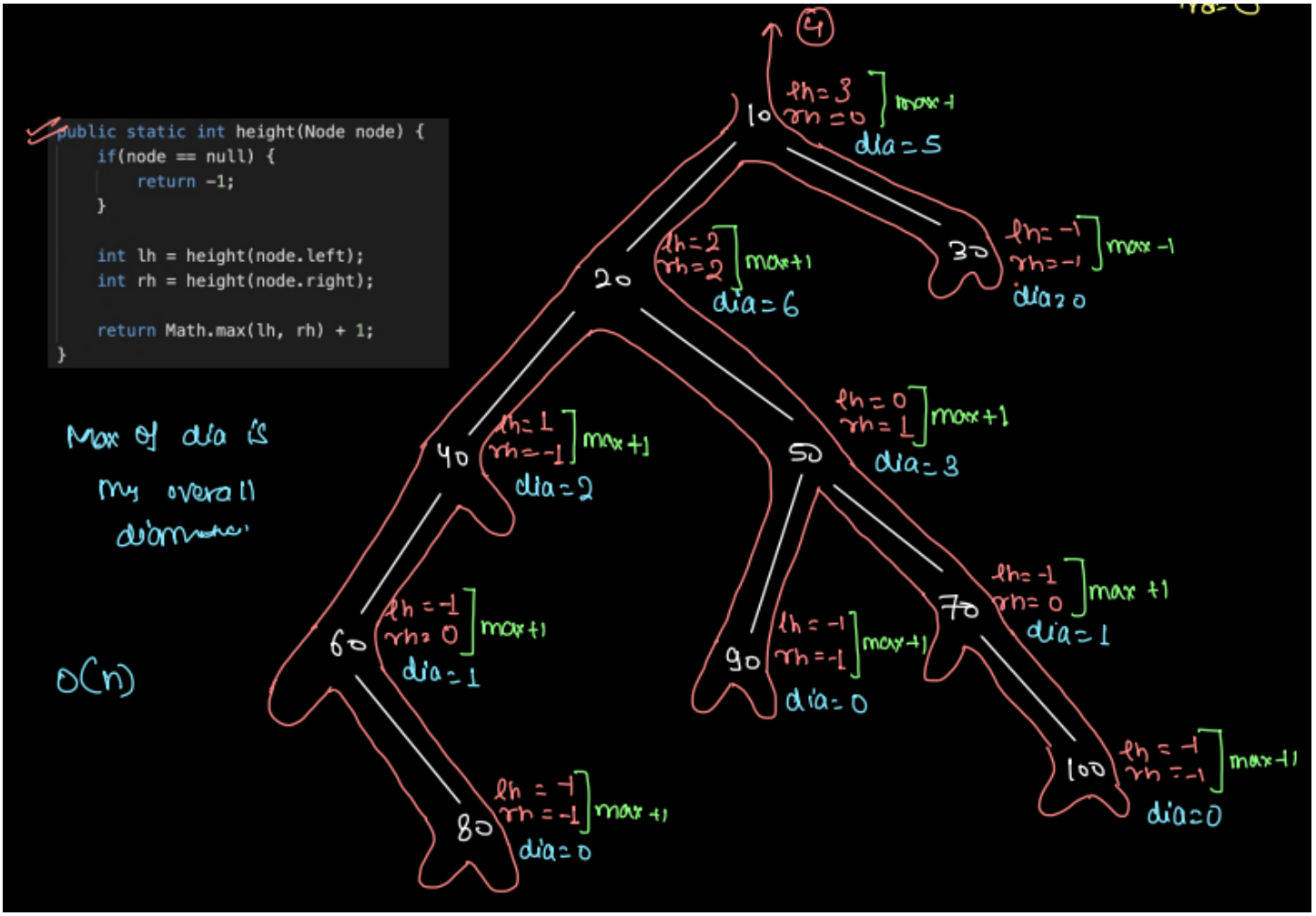
like LeftSubtreeHeight + RightSubrtreeHeight + 2 (this is the two edges from parent node because we are taking subtree height in both side);

https://www.interviewbit.com/snippet/5552311dfb5a8e80cf6f/



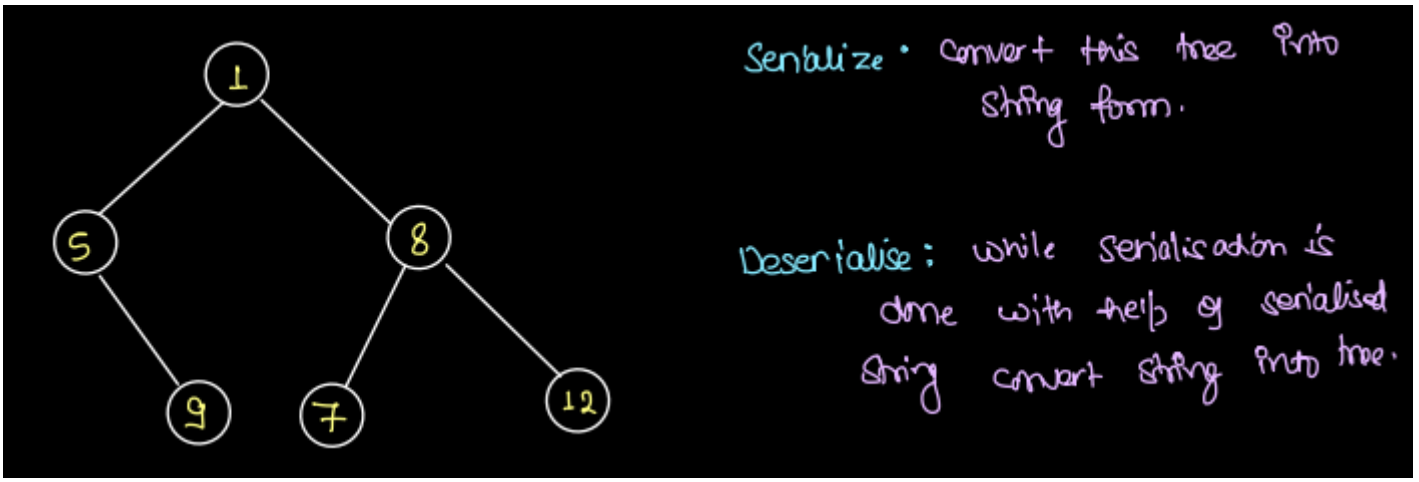


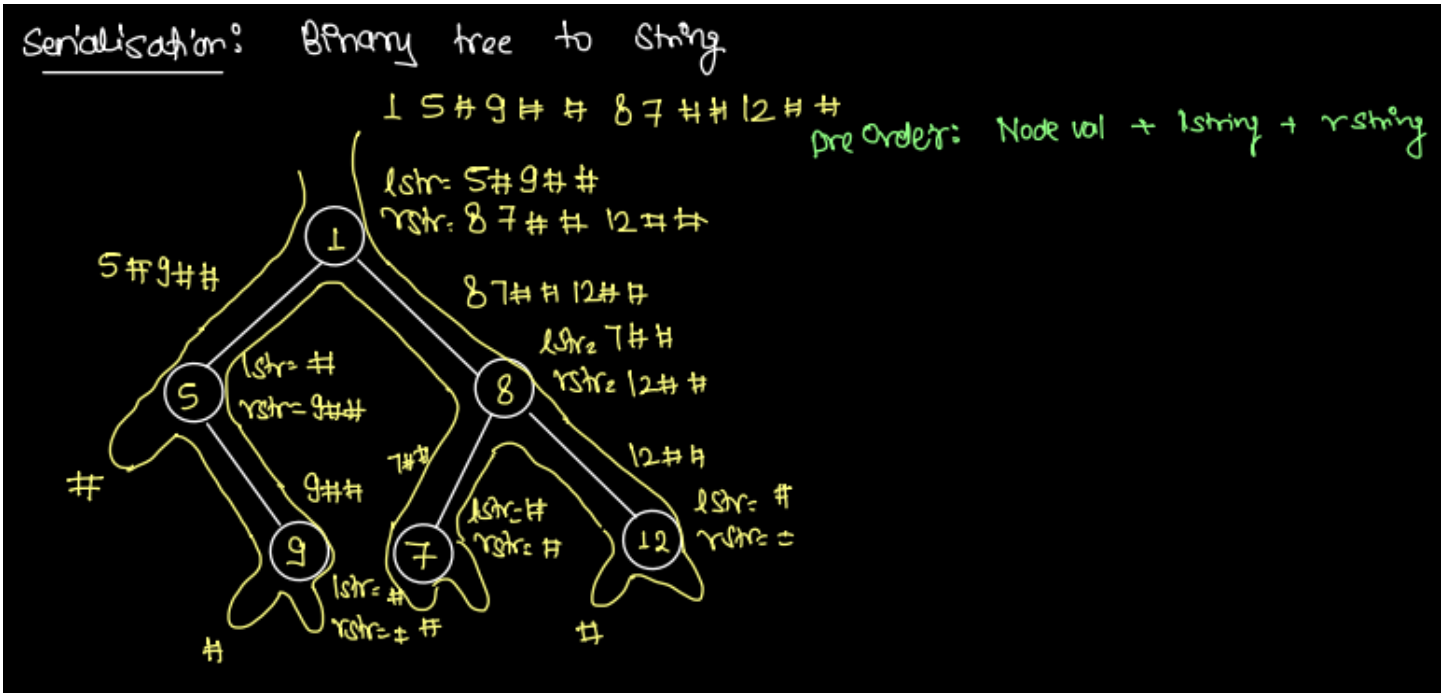
```java
// Diameter 1
public static int height(Node node) {
    if(node == null) {
        return -1;
    }

    int lh = height(node.left);
    int rh = height(node.right);

    return Math.max(lh, rh) + 1;
}

public static int diameter1(Node node) {
    if(node == null) {
        return 0;
    }

    int lh = height(node.left);
    int rh = height(node.right);
    int dia = lh + rh + 2;

    int ld = diameter1(node.left);
    int rd = diameter1(node.right);

    return Math.max(dia, Math.max(ld, rd));
}
```



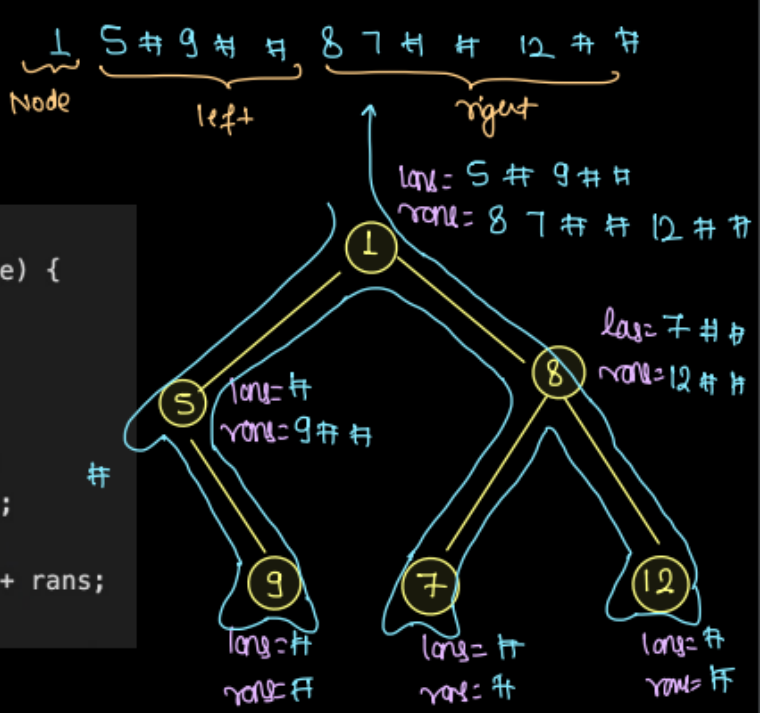
Serialize and Deserialize binary tree
For given root node: Serialize and deserialized of a binary tree.

https://www.interviewbit.com/snippet/c6d225fae2a8bec9b6f0/





```java
// Serialise and deserialise
public static String serialise(Node node) {
    if(node == null) {
        return "#";
    }

    String lans = serialise(node.left);
    String rans = serialise(node.right);

    return node.val + " " + lans + " " + rans;
}
```



Serialised String :- "1 5 # 9 # # 8 7 # # 12 # #"

If we found a data ;
- node creation
- left answer
- right answ.

else
- #
- → null is answ.

Serialised String :- "1 5 # 9 # # 8 7 # # 12 # #"

Convert this single string into array of String, on the basis of space split the entire string.

String[]: [1, 5, #, 9, #, #, 8, 7, #, #, 12, #, #]

How? String[] arr = str.split(" ");

index

String[]: [1, 5, #, 9, #, #, 8, 7, #, #, 12, #, #]

```
int n = Integer.parseInt(arr[indx]);
Node nn = new Node(n);
indx++;
nn.left = prepare left ans;
nn.right = prepare right ans;
return nn;
```

} Recursive call

```java
static int indx;

public static Node helperDS(String[] arr) {
    if(arr[indx].equals("#")) {
        indx++;
        return null;
    } else {
        int n = Integer.parseInt(arr[indx]);
        indx++;
        Node nn = new Node(n);

        nn.left = helperDS(arr);
        nn.right = helperDS(arr);

        return nn;
    }
}
public static Node deserialisation(String str) {
    String[] arr = str.split(" ");
    indx = 0;
    Node root = helperDS(arr);
    // helperDS -> recursive helper function
    //              for deserialistion
    return root;
}
```

Treemap

Introduction of TreeMap

TreeMap is Sorted HashMap
- HashMap → key + value
- Sorted → treemap is sorted on the basis of key.

Syntax:

```
TreeMap<String, Integer> tm = new TreeMap<>();
tm.put("India", 244);
tm.put("Australia", 444);
tm.put("China", 439);
tm.put("Bharat", 416);
tm.put("Pakistan", 389);
SOP(tm); → [Aus-444, Bharat-416, China-439, India-244, Pak-389]
```

NOTE: Sorted on the basis of key

```
SOP(tm.containsKey("Eg")); → false
SOP(tm.containsKey("Bharat")); → true → 416
```

Hash map	Tree map
put, containsKey, get remove → O(1)	put, containskey, get, remove, ceilingKey, floorKey → logN.