# FUNCTIONS QUESTIONS

**Question 1 :** Write a Java method to compute the average of three numbers..

**Question 2 :** Write a method named isEven that accepts an int argument. The method should return true if the argument is even, or false otherwise. Also write a program to test your method.

**Question 3 :** Write a Java program to check if a number is a palindrome in Java? ( 121 is a palindrome, 321 is not)

A number is called a palindrome if the number is equal to the reverse of a number e.g., 121 is a palindrome because the reverse of 121 is 121 itself. On the other hand, 321 is not a palindrome because the reverse of 321 is 123, which is not equal to 321.

**Question 4 : READ & CODE EXERCISE**

Search about(Google) & use the following methods of the Math class in Java:

a. Math.min( )
b. Math.max( )
c. Math.sqrt( )
d. Math.pow( )
e. Math.avg( )
f. Math.abs( )

Free reading resource (https://www.javatpoint.com/java-math)

Please feel free to look for more resources/websites on your own.

**Question 5 :**

Write a Java method to compute the sum of the digits in an integer.

(**Hint** : Approach this question in the following way :

a. Take a variable sum = 0
b. Find the last digit of the number
c. Add it to the sum
d. Repeat a & b until the number becomes 0 )

# FUNCTIONS SOLUTIONS

## Solution 1:

```java
import java.util.Scanner;
public class Solution {
public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.print("Input the first number: ");
       double x = sc.nextDouble();
       System.out.print("Input the second number: ");
       double y = sc.nextDouble();
       System.out.print("Input the third number: ");
       double z = sc.nextDouble();
       System.out.print("The average value is " + average(x, y, z)+"\n" );
   }

 public static double average(double x, double y, double z) {
       return (x + y + z) / 3;
   }
}
```

## Solution 2:

```java
import java.util.*;

public class Solution {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int num;

       System.out.print("Enter an integer: ");
       num = sc.nextInt();

       if(isEven(num)) {
           System.out.println("Number is even");
       } else {
           System.out.println("Number is odd");
```

```
        }
    }

    public static boolean isEven(int number) {
        if(number % 2 == 0) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

## Solution 3:

```
import java.util.Scanner;

public class Solution {
    public static void main(String args[]) {
        System.out.println("Please Enter a number : ");
        Scanner sc = new Scanner(System.in);
        int palindrome = sc.nextInt();

        if(isPalindrome(palindrome)) {
            System.out.println("Number : " + palindrome + " is a palindrome");
        } else {
            System.out.println("Number : " + palindrome + " is not a palindrome");
        }

    }
    public static boolean isPalindrome(int number) {
        int palindrome = number; // copied number into variable
        int reverse = 0;

        while (palindrome != 0) {
            int remainder = palindrome % 10;
            reverse = reverse * 10 + remainder;
            palindrome = palindrome / 10;
        }
```

```
        // if original and the reverse of number is equal means
        // number is palindrome in Java
        if (number == reverse) {
            return true;
        }
        return false;
    }
}
```

Solution 4: This is a DIY question & should be solved on your own.

Solution 5:

```
import java.util.Scanner;
public class Solution {
 public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     System.out.print("Input an integer: ");
     int digits = sc.nextInt();
     System.out.println("The sum is " + sumDigits(digits));
    }

public static int sumDigits(int n) {
        int sumOfDigits = 0;
        while(n > 0) {
            int lastDigit = n % 10;
            sumOfDigits += lastDigit;
            n /= 10;
        }

        return sumOfDigits;
    }
}
```

# Operator Precedence

**Operator precedence** determines the order in which the operators in an expression are evaluated.

For eg –

*int x = 3 * 4 - 1;*

In the above example, the value of x will be 11, not 9. This happens because the precedence of * operator is higher than - operator. That is why the expression is evaluated as (3 * 4) - 1 and not 3 * (4 - 1).

## Operator Precedence Table

| Operators | Precedence |
|---|---|
| postfix increment and decrement | ++ -- |
| prefix increment and decrement, and unary | ++ -- + - ~ ! |
| multiplicative | * / % |
| additive | + - |
| shift | << >> >>> |
| relational | < > <= >= instanceof |
| equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | \| |
| logical AND | && |
| logical OR | \|\| |
| ternary | ? : |
| assignment | = += -= *= /= %= &= ^= \|= <<= >>= >>>= |

## Associativity of Operators

If an expression has two operators with similar precedence, the expression is evaluated according to its **associativity** (either left to right, or right to left).

| Operators | Precedence | Associativity |
|---|---|---|
| postfix increment and decrement | ++ -- | left to right |
| prefix increment and decrement, and unary | ++ -- + - ~ ! | right to left |
| multiplicative | * / % | left to right |
| additive | + - | left to right |
| shift | << >> >>> | left to right |
| relational | < > <= >= instanceof | left to right |
| equality | == != | left to right |
| bitwise AND | & | left to right |
| bitwise exclusive OR | ^ | left to right |
| bitwise inclusive OR | \| | left to right |
| logical AND | && | left to right |
| logical OR | \|\| | left to right |
| ternary | ? : | right to left |
| assignment | = += -= *= /= %= &= ^= \|= <<= >>= >>>= | right to left |

**Note -** These notes are just for a quick glance. We don't have to memorize them all at once. Most of these rules are very logical and we have been following them in a lot of instances already.

# OPERATORS SOLUTIONS

Solution 1: Output is : 5 , 4

Solution 2: Output is : Java

Solution 3: Output is : 4, 0, 0

Solution 4: Output is : 278

Solution 5: Output is : 20, 20

# OPERATORS QUESTIONS

**Question :** What will be the output of the following programs :

(i)

```
public class Test {
  public static void main(String[] args){
    int x = 2, y = 5;

    int exp1 = (x * y / x);
    int exp2 = (x * (y / x));

    System.out.print(exp1 + " , ");
    System.out.print(exp2);
  }
}
```

(ii)

```
public class Test {
  public static void main(String[] args) {
    int x = 200, y = 50, z = 100;
    if(x > y && y > z){
     System.out.println("Hello");
    }
   if(z > y && z < x){
     System.out.println("Java");
   }
   if((y+200) < x && (y+150) < z){
     System.out.println("Hello Java");
   }
  }
}
```

(iii)

```
public class Test {
  public static void main(String[] args){
     int x, y, z;
     x = y = z = 2;
     x += y;
     y -= z;
     z /= (x + y);
     System.out.println(x + " " + y + " " + z);
  }
}
```

(iv)

```
public class Test {
  public static void main(String[] args){
   int x = 9, y = 12;
   int a = 2, b = 4, c = 6;

   int exp = 4/3 * (x + 34) + 9 * (a + b * c) + (3 + y * (2 + a)) / (a + b*y);

   System.out.println(exp);
  }
}
```

(v)

```
public class Test {
  public static void main(String[] args){
    int x = 10, y = 5;

    int exp1 = (y * (x / y + x / y));
    int exp2 = (y * x / y + y * x / y);

    System.out.println(exp1);
    System.out.println(exp2);
  }
}
```

# Space & Time Complexity

NOTE : In some videos, you will hear about words like Space & Time Complexity of Algorithms, big Oh etc.
The concept of Space & Time Complexity is covered in the later part of the course(as an individual chapter).
To be able to understand it before that, please read this reading material.

---

## SCENARIO

Imagine you give 100 Rupees to one of your friends. Having them all together, you would like your pen back.

## SPACE COMPLEXITY

Space Complexity is represented as a function that portrays the amount of space necessary foran algorithm to run until complete.

In our scenario we are looking at you can think of space complexity as the number of rooms you need to figure out who has the pen. However, in computer science, this typically means how much memory does the processes and data structures in our codebase / functions take up to achieve their goal.

## TIME COMPLEXITY

Time Complexity is represented as a function that portrays the amount of time necessary for an algorithm to run until complete.

In our scenario we are looking at time complexity that can be represented in the approach youtake in finding out who has the pen. However, in computer science, this typically means how much time the processes and data structures in our codebase / functions take up to achieve their goal.

Time complexity however is an umbrella term for the different types of time complexities that we can calculate. From fastest to slowest they are:

Worst Case Time Complexity: The absolute most number of times an operation needs to be done before completed

Average Case Time Complexity: The average number of times it will take for the algorithm / code to complete

Amortized Running Time: Similar to average, it is the number of times the operation will take when run a sufficient amount of time consecutively

Best Case Time Complexity: The fastest number of times an operation needs to complete

NOTE - Read O(n) as Big Oh of n.

Example of Complexity Function

- $O(n^2)$: You ask one friend if they have the pen. You also ask them if the other 99 friends have the pen. Afterwards move on to the next friend and repeat the process.
- O(n): You ask each friend one by one if they have the pen.

- O(log n): You divide the friends into two rooms and ask if the person with the pen isin room 1 or room 2. Depending on which group has the pen, split them up into the two rooms and repeat until there is one person in the right room with the pen.
- O(1): You remember who has the pen and you go to them directly.

You Should Know : Something that is important to note is that the Time / Space Complexity of algorithm/code is not in fact actual time or space that is required to execute a particular code but the number of times a statement executes. Meaning the function is not a measure of time /or space but a measure of what the code is actually doing.

---

SUMMARY

There are multiple solutions that can be written for the same problem. We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem. While analyzing an algorithm, we mostly consider **time** complexity and **space** complexity.

Time complexity is the time taken by the algorithm to execute each set of instructions. It is always better to select the most efficient algorithm when a simple problem can be solved with different methods.

Similarly, Space complexity of an algorithm is the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Time and space complexity depends on lots of things like hardware, operating system, processors, etc. However, we don't consider any of these factors while analyzing the algorithm. We will only consider the **execution time** of an algorithm.

# Arrays - Solutions

**NOTE - Only functions will be given in the code. Please complete the entire code by writing themain function & class on your own.**

## Question 1 (DSA Sheet #4)

### Approach 1 - Brute Force (O(n^2))

```
public boolean containsDuplicate(int[] nums) {
    for(int i=0; i<nums.length-1 ; i++) {
        for(int j=i+1; j<nums.length ; j++ ) {
            if( nums[i] == nums[j] ) {
                return true ;
            }
        }
    }
    return false;
}
```

### Approach 2 - using Sets (O(n))

**/* You should have a basic knowledge about Java HashSets before following Approach 2. It willbe taught to you in later chapters. */**

```
public boolean containsDuplicate(int[] nums) {
    HashSet<Integer> set = new HashSet<>();
    for(int i=0; i<nums.length; i++) {
        if(set.contains(nums[i])) {
            return true;
        } else {
            set.add(nums[i]);
        }
    }

    return false;
}
```

## Question 2 (DSA Sheet #6)

### Approach - Based on Binary Search

```
public int search(int[] nums, int target) {
    //min will have index of  minimum element of nums
    int min = minSearch(nums);
    //find in sorted left
    if(nums[min] <= target && target <= nums[nums.length-1]){
         return search(nums,min,nums.length-1,target);
    }
    //find in sorted right
    else{
        return search(nums,0,min,target);
    }
}

//binary search to find target in left to right boundary
public int search(int[] nums,int left,int right,int target){
    int l = left;
    int r = right;
    // System.out.println(left+" "+right);
    while(l <= r){
        int mid = l + (r - l)/2;
        if(nums[mid] == target){
            return mid;
        }
        else if(nums[mid] > target){
            r = mid-1;
        }
        else{
            l = mid+1;
        }
    }
    return -1;
}

//smallest element index
public int minSearch(int[] nums){
    int left = 0;
    int right = nums.length-1;
```

```
    while(left < right){
        int mid = left + (right - left)/2;
        if(mid > 0 && nums[mid-1] > nums[mid]){
            return mid;
        }
        else if(nums[left] <= nums[mid] && nums[mid] > nums[right]){
            left = mid+1;
        }
        else{
            right = mid-1;
        }
    }

    return left;
}
```

---

## Question 3 (DSA Sheet #8)

### Approach

```
public int maxProfit(int[] prices) {
    int buy = prices[0];
    int profit = 0;

    for (int i=1; i<prices.length; i++) {
        if (buy < prices[i]) {
            profit = Math.max(prices[i] - buy, profit);
        }
        else {
            buy = prices[i];
        }
    }

    return profit;
}
```

---

**Question 4** (DSA Sheet #11)

**/* This problem can be a little difficult for beginners to solve. Please analyze the solution if youare not able to come up with the code. */**

**Approach**

```java
public int trap(int[] height) {
    int n = height.length;

    int res = 0, l = 0, r = n - 1;
    int rMax = height[r], lMax = height[l];

    while (l < r) {
        if (lMax < rMax) {
            l++;
            lMax = Math.max(lMax, height[l]);
            res += lMax - height[l];
        } else {
            r--;
            rMax = Math.max(rMax, height[r]);
            res += rMax - height[r];
        }
    }

    return res;
}
```

---

**Question 5** (DSA Sheet #16)

**/* This problem uses List to return the numbers & HashSets to store them. These are new data structures that we will study about in later chapters. */**

**Approach**

**Let us try to understand the problem statement. The first part of the problem statement is clear, we are asked to find out all the triplets in the given array whose sum is equal to zero. A triplet is nothing but a set of three numbers in the given array. For example, if nums=[1,2, 3,4] is the given array, [1,2,3] [2,3,4] [1,3,4] etc are its triplets.**

**What does the condition i != j, i != k, and j != k mean?**

It means that we are not allowed to reuse any number from the array within a triplet. Example, for the given array nums = [1,2,3,4], triplets [1,1,1] or [1,1,2] or [1,2,2] etc are not considered validtriplets.

The last condition that we need to consider is that we cannot have duplicate triplets in our final result. Example if [-2,-2,0,2] is the given array, we can only consider one of [-2,0,2] from indexes 0, 2, 3 and [-2,0,2] from indexes 1, 2, 3 in our final result.

BRUTE FORCE - The simple solution to this problem is to find every possible triplet from the given array, see if its sum is equal to zero and return the result (ensuring there are no duplicatetriplets in the result).

This algorithm involves the following steps:

1. Use three loops to generate all possible triplets for the given array, with each loop variable keeping track of 1 triplet element each.
2. Next we calculate the sum for each triplet generated in step 1.
3. If the sum is equal to 0 we need to check if it is a unique triplet (not already in our result set). We can ensure the triplets in our result set are unique by sorting the triplets and adding it to a hashmap (hashmap overwrites data if the same value iswritten to the same key multiple times thereby eliminating duplicates).
4. Once we have added all the triplets whose sum is equal to 0 into the hashmap, we iterate through the hashmap and add it to our result array.
5. Finally we return the result array.

```
public List<List<Integer>> threeSum(int nums[]) {
   List<List<Integer>> result = new ArrayList <List<Integer>> ();

   for(int i=0; i<nums.length; i++) {
       for(int j=i+1; j<nums.length; j++) {
           for(int k=j+1; k<nums.length; k++) {
               if(nums[i] + nums[j] + nums[k] == 0) {
```

```
                    List<Integer> triplet = new ArrayList < Integer > ();
                    triplet.add(nums[i]);
                    triplet.add(nums[j]);
                    triplet.add(nums[k]);
                    Collections.sort(triplet);
                    result.add(triplet);
                }
            }
        }
    }
    result = new ArrayList<List<Integer>> (new LinkedHashSet<List<Integer>> (result));
    return result;
}
```

# BASIC SORTING QUESTIONS

**Question :** Use the following sorting algorithms to sort an array in DESCENDING order :

a. Bubble Sort
b. Selection Sort
c. Insertion Sort
d. Counting Sort

You can use this array as an example : **[3, 6, 2, 1, 8, 7, 4, 5, 3, 1]**

# BASIC SORTING SOLUTIONS

## Solution (a): Bubble Sort

```
public static void bubbleSortDescending(int arr[]) {
        for(int turn=0; turn<arr.length-1; turn++) {
            for(int j=0; j<arr.length-1-turn; j++) {
                if(arr[j] < arr[j+1]) {
                    //swap
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
    }
```

## Solution (b): Selection Sort

```
public static void selectionSortDescending(int arr[]) {
        for(int turn=0; turn<arr.length; turn++) {
            int minPos = turn;
            for(int j=turn+1; j<arr.length; j++) {
                if(arr[minPos] < arr[j]) {
                    minPos = j;
                }
            }

            //swap
            int temp = arr[turn];
            arr[turn] = arr[minPos];
            arr[minPos] = temp;
        }
    }
```

## Solution (c) : Insertion Sort

```
public static void insertionSortDescending(int arr[]) {
```

```
        for(int i=1; i<arr.length; i++) {
            int curr = arr[i];
            int prev = i-1;
            //to find the index where curr is to be inserted
            while(prev >= 0 && arr[prev] < curr) {
                arr[prev+1] = arr[prev];
                prev--;
            }
            arr[prev+1] = curr;
        }
    }
```

## Solution (d): Counting Sort

```
public static void countingSortDescending(int arr[]) {
        int largest = Integer.MIN_VALUE;
        for(int i=0; i<arr.length; i++) {
            largest = Math.max(largest, arr[i]);
        }

        int count[] = new int[largest+1];
        for(int i=0; i<arr.length; i++) {
            count[arr[i]]++;
        }
        int j = 0;
        for(int i=count.length-1; i>=0; i--) {
            while(count[i] > 0) {
                arr[j] = i;
                j++;
                count[i]--;
            }
        }
    }
```

# 2D ARRAYS QUESTIONS

**Question 1 :** Print the number of 7's that are in the 2d array.

Example :

Input - int[][] array = { {4,7,8},{8,8,7} };

Output - 2

**Question 2 :** Print out the sum of the numbers in the second row of the "nums" array.

Example :

Input - int[][] nums = { {1,4,9},{11,4,3},{2,2,3} };

Output - 18

**Question 3 :** Write a program to Find **Transpose** of a Matrix.

What is Transpose?

Transpose of a matrix is the process of **swapping the rows to columns**. For a 2x3 matrix,

Matrix

a11 a12 a13

a21 a22 a23

Transposed Matrix

a11 a21

a12 a22

a13 a23

# 2D ARRAYS SOLUTIONS

## Solution 1:

```java
public class Solution {
    public static void main(String[] args) {
        int[][] array = { {4, 7, 8}, {8, 8, 7} };

        int countOf7 = 0;
        for(int i=0; i<array.length; i++) {
            for(int j=0; j<array[0].length; j++) {
                if(array[i][j] == 7) {
                    countOf7++;
                }
            }
        }

        System.out.println("count of 7 is : " + countOf7);
    }
}
```

## Solution 2:

```java
public class Solution {
    public static void main(String[] args) {
        int[][] nums = { {1,4,9},{11,4,3},{2,2,3} };
        int sum = 0;

        //sum of 2nd row elements
        for(int j=0; j<nums[0].length; j++) {
            sum += nums[1][j];
        }

        System.out.println("sum is : " + sum);
    }
}
```

Solution 3 :

```java
public class Solution {
    public static void main(String[] args) {
        int row = 2, column = 3;
        int[][] matrix = { {2, 3, 7}, {5, 6, 7} };

        // Display original matrix
        printMatrix(matrix);

        // Transpose the matrix
        int[][] transpose = new int[column][row];
        for(int i = 0; i<row; i++) {
            for (int j = 0; j<column; j++) {
                transpose[j][i] = matrix[i][j];
            }
        }

        // print the transposed matrix
        printMatrix(transpose);
    }

    public static void printMatrix(int[][] matrix) {
        System.out.println("The matrix is: ");
        for(int i=0; i<matrix.length; i++) {
            for (int j=0; j<matrix[0].length; j++) {
                System.out.print(matrix[i][j] + "    ");
            }
            System.out.println();
        }
    }
}
```

# STRINGS QUESTIONS

**Question 1 :** Count how many times lowercase vowels occurred in a String entered by the user.

**Question 2 :** What will be the output of the following code?

```
public class Solution {
   public static void main(String args[]) {
       String str = "ShradhaDidi";
       String str1 = "ApnaCollege";
       String str2 = "ShradhaDidi";
       System.out.println(str.equals(str1) + " " + str.equals(str2));
   }
}
```

**Question 3 :** What will be the output of the following code?

```
public class Solution {
 public static void main(String args[]) {
   String str = "ApnaCollege".replace("l", "");
   System.out.println(str);
 }
}
```

**Question 4 :** Determine if 2 Strings are **anagrams** of each other.

What are anagrams?

If two strings contain the same characters but in a different order, they can be said to be anagrams. Consider race and care. In this case, race's characters can be formed into a study, or care's characters can be formed into race. Below is a java program to check if two strings are anagrams or not.

**Question 5 :** Search and read about

a. intern( ) method in String
b. StringBuffer

# STRINGS SOLUTIONS

## Solution 1:

```java
import java.util.*;

public class Solution {
   public static void main(String[] args) {
       String str = new Scanner(System.in).next();
       int count = 0;

       for(int i=0; i<str.length(); i++) {
           char ch = str.charAt(i);
           if(ch == 'a' || ch == 'e' ||ch == 'i' || ch == 'o' ||ch == 'u') {
               count++;
           }
       }
       System.out.println("count of vowels is :" + count);
   }
}
```

## Solution 2: Output will be :

false true

(If you need an explanation, please r-ewatch the video about how Strings work in memory?)

## Solution 3 : Output will be :

ApnaCoege

Following are some methods in Java which are used to replace characters:

| | |
|---|---|
| `String` | **`replace`**`(char oldChar, char newChar)`<br>Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`. |
| `String` | **`replace`**`(CharSequence target, CharSequence replacement)`<br>Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. |
| `String` | **`replaceAll`**`(String regex, String replacement)`<br>Replaces each substring of this string that matches the given **regular expression** with the given replacement. |
| `String` | **`replaceFirst`**`(String regex, String replacement)`<br>Replaces the first substring of this string that matches the given **regular expression** with the given replacement. |

Solution 4:

```
import java.util.Arrays;
public class Solution {
  public static void main(String[] args) {
       String str1 = "earth";
       String str2 = "heart";

       //Convert Strings to lowercase. Why? so that we don't have to check
separately for lower & uppercase.
       str1 = str1.toLowerCase();
       str2 = str2.toLowerCase();

       // First check - if the lengths are the same
       if(str1.length() == str2.length())  {
            // convert strings into char array
            char[] str1charArray = str1.toCharArray();
            char[] str2charArray = str2.toCharArray();
            // sort the char array
            Arrays.sort(str1charArray);
            Arrays.sort(str2charArray);
            // if the sorted char arrays are same or identical then the strings are
anagram
            boolean result = Arrays.equals(str1charArray, str2charArray);
            if(result) {
               System.out.println(str1 + " and " + str2 + " are anagrams of each
other.");
             } else {
               System.out.println(str1 + " and " + str2 + " are not anagrams of
each other.");
             }
       } else {
           // case when lengths are not equal
          System.out.println(str1 + " and " + str2 + " are not anagrams of each
other.");
       }
   }
}
```

# BIT MANIPULATION QUESTIONS

NOTE - Some of these questions will teach you something new as some of these might be useful hacks (which can be difficult to think of on your own). Even if you are not able to come up with a solution on your own. Try to learn something new from these questions & write down the logic of the solution in your notes.

Question 1 : What is the value of x^x for any value of x?

Question 2 : Swap two numbers without using any third variable.

Question 3 : Add 1 to an integer using Bit Manipulation.
(**Hint** : try using Bitwise NOT Operator)

Question 4 : This question is based on a trick, please directly look at the solution.
Convert uppercase characters to lowercase using bits.

Question 5 : A Good read of hacks using bits (you can check this out in your free time) :
http://graphics.stanford.edu/~seander/bithacks.html

# BIT MANIPULATION SOLUTIONS

**Solution 1:** The value of x^x = 0.

Think about it, xor gives 0 when the bits are the same. If we compare the same number to itself, the bits will always be the same. So, the answer of x^x will always be 0.

**Solution 2:** The idea is to use XOR operators to swap two numbers by their property x ^ x = 0

```java
public class Solution {
    public static void main(String[] args) {
        int x = 3, y = 4;
        System.out.println("Before swap: x = " + x + " and y = " + y);
        //swap using xor
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
        System.out.println("After swap: x = " + x + " and y = " + y);
    }
}
```

**Solution 3 :** The expression ~~x will add 1 to an integer x. We know that to get negative of a number, invert its bits and add 1 to it (Remember negative numbers are stored in 2's complement form), i.e.,

-x = ~x + 1;

-~x = x + 1 (by replacing x by ~x)

```java
public class Solution {
    public static void main(String[] args) {
        int x = 6;
        System.out.println(x + " " + 1 + " is " + -~x);
        x = -4;
        System.out.println(x + " " + 1 + " is " + -~x);
        x = 0;
        System.out.println(x + " " + 1 + " is " + -~x);
    }
}
```

Solution 4 :

```java
public class Solution {
   public static void main(String[] args) {
       // Convert uppercase character to lowercase
       for (char ch = 'A'; ch <= 'Z'; ch++) {
           System.out.println((char)(ch | ' '));
            // prints abcdefghijklmnopqrstuvwxyz
       }
   }
}
```

# OOPS QUESTIONS

**Question 1 :** Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by the user.

**Question 2 :** What is the output of the following program?

```
class Automobile {
    private String drive() {
        return "Driving vehicle";
    }
}

class Car extends Automobile {
    protected String drive() {
        return "Driving car";
    }
}

public class ElectricCar extends Car {

    @Override
    public final String drive() {
        return "Driving electric car";
    }

    public static void main(String[] wheels) {
        final Car car = new ElectricCar();
        System.out.print(car.drive());
    }
}
```



A. Driving vehicle
B. Driving electric car
C. Driving car
D. The code does not compile

Question 3 : Look at the following code and choose the right option for the word:

```
// Shape.java
public class Shape {
   protected void display() {
       System.out.println("Display-base");
   }
}
// Circle.java
public class Circle extends Shape { <
   < access - modifier > void display() {
       System.out.println("Display-derived");
   }
}
```

a. Only protected can be used.
B. public and protected both can be used.
C. public, protected, and private can be used.
d. Only public can be used.



Question 4 : What is the output of the following program?

```
abstract class Car {
   static {
       System.out.print("1");
   }

   public Car(String name) {
       super();
       System.out.print("2");
   }

   {
       System.out.print("3");
   }
}

public class BlueCar extends Car {
   {
       System.out.print("4");
```

```
    }

    public BlueCar() {
        super("blue");
        System.out.print("5");
    }

    public static void main(String[] gears) {
        new BlueCar();
    }
}
```

A. 23451

B. 12354

C. 13245

D. The code does not compile.

Question 5 : Read up about basics of exception handling from here : https://www.w3schools.com/java/java_try_catch.asp

# OOPS SOLUTIONS

## Solution 1:

```
import java.util.*;
class Complex{
 int real;
 int imag;
 public Complex(int r, int i){
   real = r;
   imag = i;
 }

 public static Complex add(Complex a, Complex b){
   return new Complex((a.real+b.real),(a.imag+b.imag));
 }

 public static Complex diff(Complex a, Complex b){
   return new Complex((a.real-b.real),(a.imag-b.imag));
 }

 public static Complex product(Complex a, Complex b){
                                                    return                    new
Complex(((a.real*b.real)-(a.imag*b.imag)),((a.real*b.imag)+(a.imag*b.real)));
 }

 public void printComplex(){
   if(real == 0 && imag!=0){
     System.out.println(imag+"i");
   }
   else if(imag == 0 && real!=0){
     System.out.println(real);
   }
   else{
     System.out.println(real+"+"+imag+"i");
   }
 }

}
```

```
class Solution {
 public static void main(String[] args){
   Complex c = new Complex(4,5);
   Complex d = new Complex(9,4);

   Complex e = Complex.add(c,d);
   Complex f = Complex.diff(c,d);
   Complex g = Complex.product(c,d);
   e.printComplex();
   f.printComplex();
   g.printComplex();
 }
}
```

## Solution 2: B. Driving electric car

The drive() method in the Car class does not override the version in the Automobile class since the method is not visible to the Car class.

The drive() method in the ElectricCar class is a valid override of the method in the Car class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect.

In the main() method, the object created is an ElectricCar, even if it is assigned to a Car reference. Due to polymorphism, the method from the ElectricCar will be invoked, making Option B the correct answer.

## Solution 3 : B. public and protected both can be used.

You can provide only a less restrictive or same-access modifier when overriding a method.

## Solution 4 : C. 13245

The class is loaded first, with the static initialization block called and 1 is outputted first. When the BlueCar is created in the main() method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next. Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making Option C the correct answer.

# RECURSION (Basics) QUESTIONS

Question 1 : For a given integer array of size N. You have to find all the occurrences (indices) of a given element (Key) and print them. Use a recursive function to solve this problem.

**Sample Input** : arr[ ] = {3, 2, 4, 5, 6, 2, 7, 2, 2}, key = 2
**Sample Output** : 1 5 7 8

Question 2 :

You are given a number (eg - 2019), convert it into a String of english like "two zero one nine". *Use a recursive function to solve this problem.*

**NOTE -** The digits of the number will only be in the range 0-9 and the last digit of a number can't be 0.

**Sample Input** : 1947
**Sample Output** : "one nine four seven"



Question 3 : Write a program to find **Length of a String** using Recursion.

Question 4 : We are given a string S, we need to find the count of all contiguous substrings starting and ending with the same character.

**Sample Input 1** : S = "abcab"
**Sample Output 1** : 7
There are 15 substrings of "abcab" : a, ab, abc, abca, abcab, b, bc, bca, bcab, c, ca, cab, a, ab, b
Out of the above substrings, there are 7 substrings : a, abca, b, bcab, c, a and b. So, only 7 contiguous substrings start and end with the same character.

**Sample Input 2** : S = "aba"
**Sample Output 2** : 4
The substrings are a, b, a and aba.

Question 5 : **TOWER OF HANOI** (Important!)

You have 3 towers and N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sits on top of an even larger one).

You have the following constraints:

(1) Only one disk can be moved at a time.
(2) A disk is slid off the top of one tower onto another tower.
(3) A disk cannot be placed on top of a smaller disk. Write a program to move the disks from the first tower to the
last using Stacks.

Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.
An **example** with 2 disks i.e. N=2:
Step 1 : Shift the first disk from 'A' to 'B'.

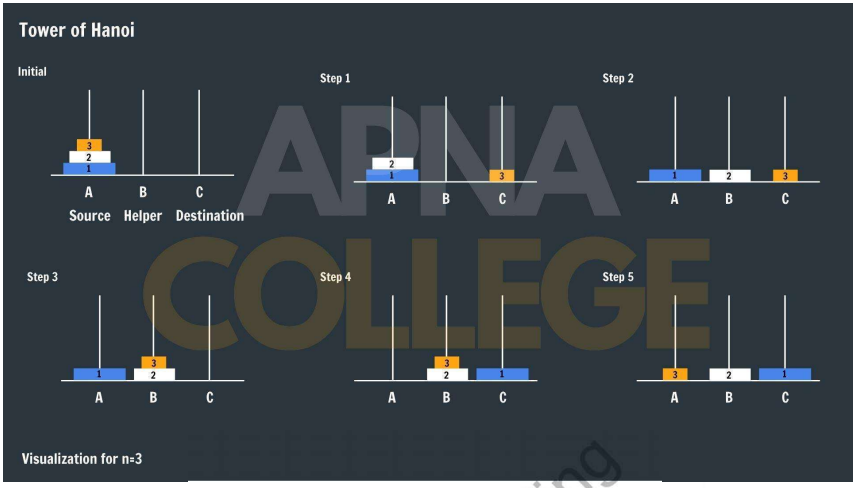
Step 2 : Shift the second disk from 'A' to 'C'.

Step 3 : Shift the first disk from 'B' to 'C'.

An **example** with 3 disks i.e. N=3 :
Step 1 : Shift the first disk from 'A' to 'C'.
Step 2 : Shift second disk from 'A' to 'B'.
Step 3 : Shift the first disk from 'C' to 'B'.

Step 4 : Shift the third disk from 'A' to 'C'.

Step 5 : Shift the first disk from 'B' to 'A'.
Step 6 : Shift second disk from 'B' to 'C'.
Step 7 : Shift the first disk from 'A' to 'C'.





The Approach here is :

- Shift 'n-1' disks from 'A' to 'B', using C.
- Shift the last disk from 'A' to 'C'.
- Shift 'n-1' disks from 'B' to 'C', using A.

# RECURSION (Basics) SOLUTIONS

Solution 1:

```java
public class Solution {
    public static void allOccurences(int arr[], int key, int i) {
        if(i == arr.length) {
            return;
        }

        if(arr[i] == key) {
            System.out.print(i+" ");
        }
        allOccurences(arr, key, i+1);
    }
    public static void main(String[] args) {
        int arr[] = {3, 2, 4, 5, 6, 2, 7, 2, 2};
        int key = 2;
        allOccurences(arr, key, 0);
        System.out.println();
    }
}
```



Solution 2:

```java
public class Solution {
    static String digits[] = {"zero", "one", "two", "three", "four", "five", "six",
"seven", "eight", "nine"};

    public static void printDigits(int number) {
        if(number == 0) {
            return;
        }

        int lastDigit = number%10;
        printDigits(number/10);
        System.out.print(digits[lastDigit]+" ");
    }
    public static void main(String[] args) {
        printDigits(1234);
```

```
        System.out.println();
    }
}
```

Solution 3 :

```
public class Solution {
    public static int length(String str) {
        if(str.length() == 0) {
            return 0;
        }

        return length(str.substring(1)) + 1;
    }
    public static void main(String[] args) {
        String str = "abcde";
        System.out.println(length(str));
    }
}
```

Solution 4 :

```
public class Solution {
    public static int countSubstrs(String str, int i, int j, int n) {
        if (n == 1) {
            return 1;
        }
        if (n <= 0) {
            return 0;
        }

        int res = countSubstrs(str, i + 1, j, n - 1) +
                countSubstrs(str, i, j - 1, n - 1) -
                countSubstrs(str, i + 1, j - 1, n - 2);

        if (str.charAt(i) == str.charAt(j)) {
            res++;
        }
```

```
        return res;
    }
    public static void main(String[] args) {
        String str = "abcab";
        int n = str.length();
        System.out.print(countSubstrs(str, 0, n-1, n));
    }
}
```

Solution 5 :

```
public class Solution {
    public static void towerOfHanoi(int n, String src, String helper, String dest) {
        if(n == 1) {
        System.out.println("transfer disk " + n + " from " + src + " to " + dest);
        return;
        }

        //transfer top n-1 from src to helper using dest as 'helper'
        towerOfHanoi(n-1, src, dest, helper);
        //transfer nth from src to dest
        System.out.println("transfer disk " + n + " from " + src + " to " + helper);
        //transfer n-1 from helper to dest using src as 'helper'
        towerOfHanoi(n-1, helper, src, dest);
    }

    public static void main(String args[]) {
        int n = 4;
        towerOfHanoi(n, "A", "B", "C");
    }
}
```



The Solution for this particular question has also been discussed here : https://www.youtube.com/watch?v=u-HgzgYe8KA

At timestamp : 00:05

# Divide & Conquer QUESTIONS

Question 1 : Apply Merge sort to sort an array of Strings. (Assume that all the characters in all the Strings are in lowercase). (**EASY**)

**Sample Input 1** : arr = { "sun", "earth", "mars", "mercury" }
**Sample Output 1** : arr = { "earth", "mars", "mercury", "sun"}

Question 2 : Given an array nums of size n, return the majority element. (**MEDIUM**)

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

**Sample Input 1** : nums = [3,2,3]
**Sample Output 1** : 3

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

**Constraints** (extra Conditions):

- n == nums.length
- 1 <= n <= 5 * 104
- -109 <= nums[i] <= 109



Question 3 : Given an array of integers. Find the Inversion Count in the array. (**HARD**)

*Inversion Count:* For an array, inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum.
Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

**Sample Input 1** : N = 5, arr[ ] = {2, 4, 1, 3, 5}
**Sample Output 1** : 3, because it has 3 inversions - (2, 1), (4, 1), (4, 3).

**Sample Input 2** : N = 5, arr[ ] = {2, 3, 4, 5, 6}
**Sample Output 2** : 0, because the array is already sorted

**Sample Input 3** : N = 3, arr[] = {5, 5, 5}
**Sample Output 3** : 0, because all the elements of the array are the same & already in a sorted manner.

(**Hint** : A sorting algorithm will be used to solve this question.)
Note - This question is important. Even if you are not able to come up with the approach, please understand the solution.

# Divide & Conquer SOLUTIONS

Solution 1:

```
class Solution {
   //function to mergeSort 2 arrays
   public static String[] mergeSort(String[] arr, int lo, int hi) {
       if (lo == hi) {
           String[] A = { arr[lo] };
           return A;
       }
       int mid = lo + (hi - lo) / 2;
       String[] arr1 = mergeSort(arr, lo, mid);
       String[] arr2 = mergeSort(arr, mid + 1, hi);

       String[] arr3 = merge(arr1, arr2);
       return arr3;
   }

   static String[] merge(String[] arr1, String[] arr2) {
       int m = arr1.length;
       int n = arr2.length;
       String[] arr3 = new String[m + n];

       int idx = 0;

       int i = 0;
       int j = 0;

       while (i < m && j < n) {
           if (isAlphabeticallySmaller(arr1[i], arr2[j])) {
               arr3[idx] = arr1[i];
               i++;
               idx++;
           }
           else {
               arr3[idx] = arr2[j];
               j++;
               idx++;
           }
```

```
        }
        while (i < m) {
            arr3[idx] = arr1[i];
            i++;
            idx++;
        }
        while (j < n) {
            arr3[idx] = arr2[j];
            j++;
            idx++;
        }
        return arr3;
    }

    // Return true if str1 appears before str2 in alphabetical order
    static boolean isAlphabeticallySmaller(String str1, String str2) {
        if (str1.compareTo(str2) < 0) {
            return true;
        }
        return false;
    }
    public static void main(String[] args) {
        String[] arr = { "sun", "earth", "mars", "mercury" };
        String[] a = mergeSort(arr, 0, arr.length - 1);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
}
```



## Solution 2:

Approach 1 - Brute Force Approach

Idea : Count the number of times each number occurs in the array & find the largest count.

Time complexity - O(n^2)

```
class Solution {
    public static int majorityElement(int[] nums) {
        int majorityCount = nums.length/2;
```

```
        for (int i=0; i<nums.length; i++) {
            int count = 0;
            for (int j=0; j<nums.length; j++) {
                if (nums[j] == nums[i]) {
                    count += 1;
                }
            }
            if (count > majorityCount) {
                return nums[i];
            }
        }
        return -1;
    }
    public static void main(String args[]) {
        int nums[] = {2,2,1,1,1,2,2};
        System.out.println(majorityElement(nums));
    }
}
```

Approach 2 - Divide & Conquer

Idea : If we know the majority element in the left and right halves of an array, we can determine which is the global majority element in linear time.

Here, we apply a classical divide & conquer approach that recurses on the left and right halves of an array until an answer can be trivially achieved for a length-1 array. Note that because actually passing copies of subarrays costs time and space, we instead pass lo and hi indices that describe the relevant slice of the overall array. In this case, the majority element for a length-1 slice is trivially its only element, so the recursion stops there. If the current slice is longer than length-1, we must combine the answers for the slice's left and right halves. If they agree on the majority element, then the majority element for the overall slice is obviously the same[1]. If they disagree, only one of them can be "right", so we need to count the occurrences of the left and right majority elements to determine which subslice's answer is globally correct. The overall answer for the array is thus the majority element between indices 0 and n.

Time complexity - O(n*logn)

```
class Solution {
    private static int countInRange(int[] nums, int num, int lo, int hi) {
        int count = 0;
        for (int i = lo; i <= hi; i++) {
            if (nums[i] == num) {
```

```
                count++;
            }
        }
        return count;
    }

    private static int majorityElementRec(int[] nums, int lo, int hi) {
        // base case; the only element in an array of size 1 is the majority
        // element.
        if (lo == hi) {
            return nums[lo];
        }

        // recurse on left and right halves of this slice.
        int mid = (hi-lo)/2 + lo;
        int left = majorityElementRec(nums, lo, mid);
        int right = majorityElementRec(nums, mid+1, hi);

        // if the two halves agree on the majority element, return it.
        if (left == right) {
            return left;
        }

        // otherwise, count each element and return the "winner".
        int leftCount = countInRange(nums, left, lo, hi);
        int rightCount = countInRange(nums, right, lo, hi);

        return leftCount > rightCount ? left : right;
    }

    public static int majorityElement(int[] nums) {
        return majorityElementRec(nums, 0, nums.length-1);
    }

    public static void main(String args[]) {
        int nums[] = {2,2,1,1,1,2,2};
        System.out.println(majorityElement(nums));
    }
}
```

(You can also find this problem at - https://leetcode.com/problems/majority-element/)

## Solution 3 :

Approach 1 - Brute Force Approach

Idea : Traverse through the array, and for every index, find the number of smaller elements on its right side of the array. This can be done using a nested loop. Sum up the counts for all indexes in the array and print the sum.

- Traverse through the array from start to end
- For every element, find the count of elements smaller than the current number up to that index using another loop.
- Sum up the count of inversion for every index.
- Print the count of inversions.

Time complexity - O(n^2)

```
class Solution {
    public static int getInvCount(int arr[]) {
        int n = arr.length;
        int invCount = 0;
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (arr[i] > arr[j]) {
                    invCount++;
                }
            }
        }
        return invCount;
    }
    public static void main(String[] args) {
        int arr[] = {1, 20, 6, 4, 5};
        System.out.println("Inversion Count = "+ getInvCount(arr));
    }
}
```

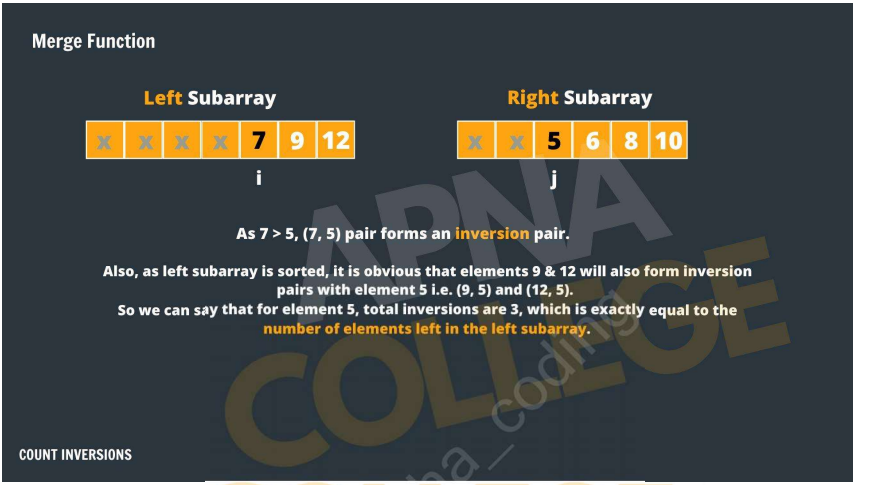


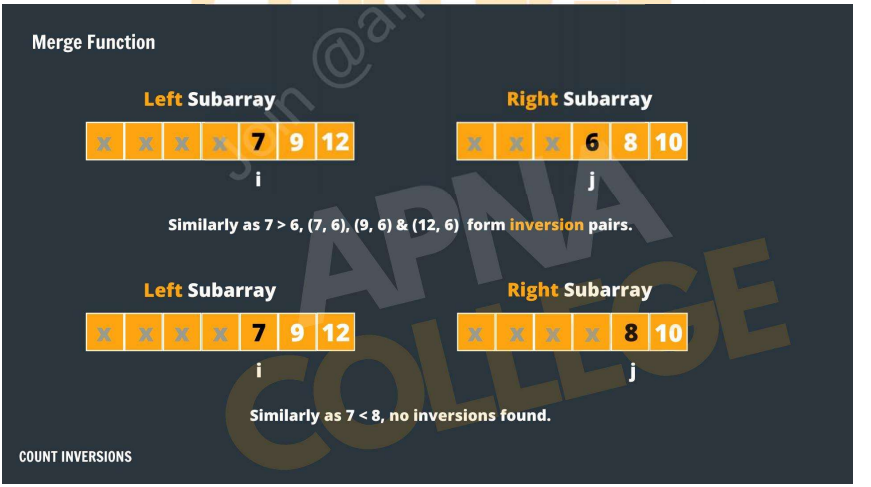
Approach 2 - Modified Merge Sort

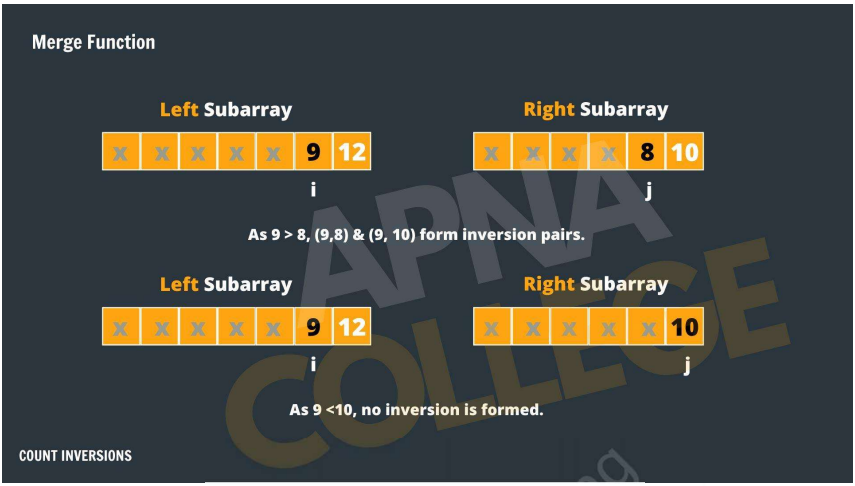
Idea : Suppose the number of inversions in the left half and right half of the array (let be inv1 and inv2); what kinds of inversions are not accounted for in Inv1 + Inv2? The answer is – the inversions that need to be counted during the merge step. Therefore, to get the total number of inversions that need to be added are the number of inversions in the left subarray, right subarray, and merge().

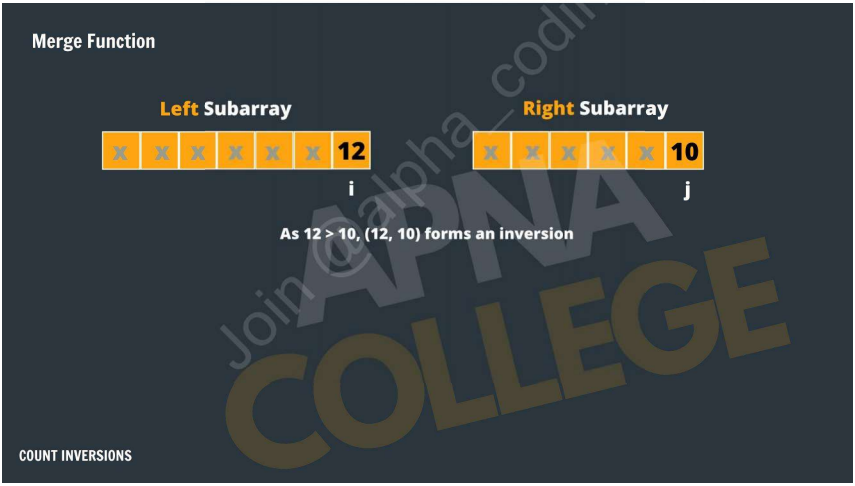
Basically, for each array element, count all elements more than it to its left and add the count to the output. This whole magic happens inside the merge function of merge sort.

Let's consider two subarrays involved in the merge process:






Algorithm:

- Split the given input array into two halves, left and right similar to merge sort recursively.
- Count the number of inversions in the left half and right half along with the inversions found during the merging of the two halves.
- Stop the recursion, only when 1 element is left in both halves.
- To count the number of inversions, we will use a two pointers approach. Let us consider two pointers i and j, one pointing to the left half and the other pointing towards the right half.

- While iterating through both the halves, if the current element A[i] is less than A[j], add it to the sorted list, else increment the count by mid – i.

Time complexity - O(n* logn)

```
public class Solution {
   public static int merge(int arr[], int left, int mid, int right) {
       int i = left, j = mid, k = 0;
       int invCount = 0;
       int temp[] = new int[(right - left + 1)];

       while ((i < mid) && (j <= right)) {
           if (arr[i] <= arr[j]) {
               temp[k] = arr[i];
               k++;
               i++;
           }
           else {
               temp[k] = arr[j];
               invCount += (mid - i);
               k++;
               j++;
           }
       }

       while (i < mid) {
           temp[k] = arr[i];
           k++;
           i++;
       }

       while (j <= right) {
           temp[k] = arr[j];
           k++;
           j++;
       }

       for (i = left, k = 0; i <= right; i++, k++) {
           arr[i] = temp[k];
       }
```

```
        return invCount;
    }

    private static int mergeSort(int arr[], int left, int right) {
        int invCount = 0;

        if (right > left) {
            int mid = (right + left) / 2;

            invCount = mergeSort(arr, left, mid);
            invCount += mergeSort(arr, mid + 1, right);
            invCount += merge(arr, left, mid + 1, right);
        }
        return invCount;
    }

    public static int getInversions(int arr[]) {
        int n = arr.length;
        return mergeSort(arr, 0, n - 1);
    }

    public static void main(String args[]) {
        int arr[] = {1, 20, 6, 4, 5};
        System.out.println("Inversion Count = " + getInversions(arr));
    }
}
```

# BACKTRACKING QUESTIONS

Note - These are classical questions. Please positively solve them.

## Question 1 :

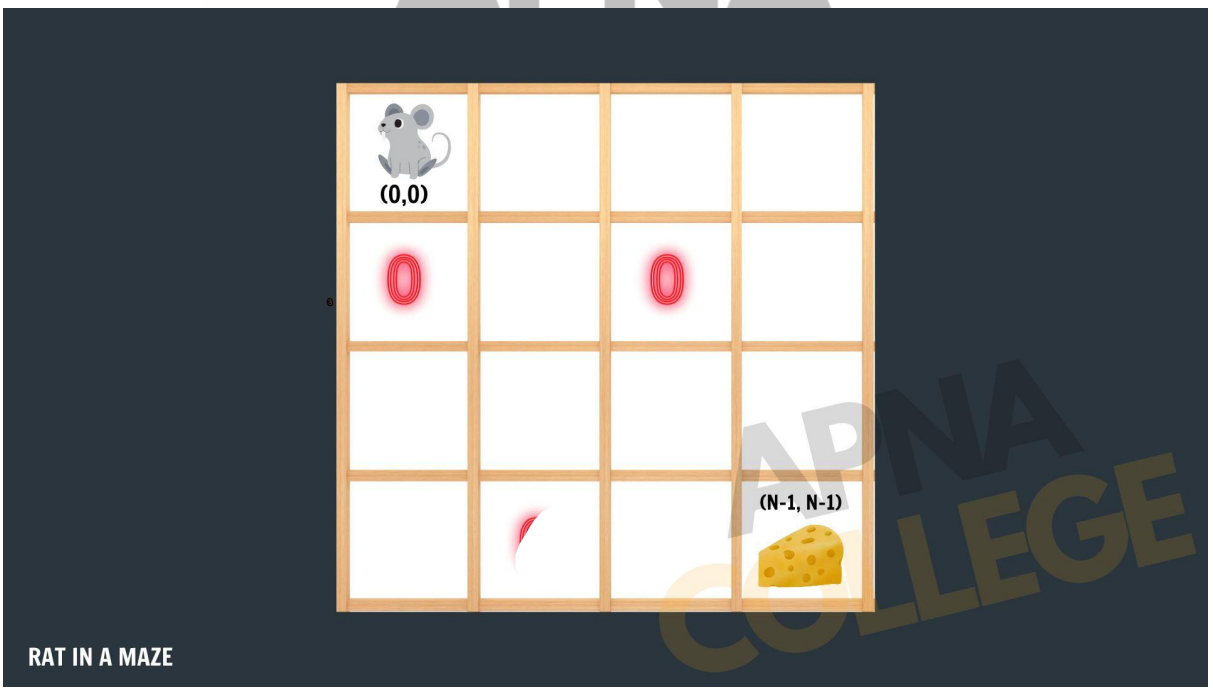
**Rat in a Maze**

You are given a starting position for a rat which is stuck in a maze at an initial point (0, 0) (the maze can be thought of as a 2-dimensional plane). The maze would be given in the form of a square matrix of order N * N where the cells with value 0 represent the maze's blocked locations while value 1 is the open/available path that the rat can take to reach its destination. The rat's destination is at (N - 1, N - 1).

Your task is to find all the possible paths that the rat can take to reach from source to destination in the maze.

The possible directions that it can take to move in the maze are 'U'(up) i.e. (x, y - 1) , 'D'(down) i.e. (x, y + 1) , 'L' (left) i.e. (x - 1, y), 'R' (right) i.e. (x + 1, y).

(This problem is similar to Grid ways.)




**Sample Input** : int maze[][] = { { 1, 0, 0, 0 },
{ 1, 1, 0, 1 },
{ 0, 1, 0, 0 },
{ 1, 1, 1, 1 } };

**Sample Output** : 1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1

## Question 2 :

**Keypad Combinations**

Given a string containing digits from 2-9 inclusive, print all possible letter combinations that the number could represent. You can print the answer in any order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.


**Sample Input 1** : digits = "23"
**Sample Output 1** : "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"

**Sample Input 2** : digits = "2"
**Sample Output 2** : "a", "b", "c"

**Sample Input 3** : digits = ""
**Sample Output 3** : ""

## Question 3 :

**Knight's Tour**

Given a N*N board with the Knight placed on the first block of an empty board. Moving according to the rules of chess, knights must visit each square exactly once. Print the order of each cell in which they are visited.

**Sample Input 1** : N = 8

**Sample Output 1** :

```
0  59  38  33  30  17   8  63
37  34  31  60   9  62  29  16
58   1  36  39  32  27  18   7
35  48  41  26  61  10  15  28
42  57   2  49  40  23   6  19
47  50  45  54  25  20  11  14
56  43  52   3  22  13  24   5
51  46  55  44  53   4  21  12
```

(**Hint** : Similar to N Queens)

# BACKTRACKING SOLUTIONS

## Solution 1:

Algorithm -

1. Create a solution matrix, initially filled with 0's.
2. Create a recursive function, which takes the initial matrix, output matrix and position of rat (i, j).
3. if the position is out of the matrix or the position is not valid then return.
4. Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
5. Recursively call for position (i+1, j) and (i, j+1).
6. Unmark position (i, j), i.e output[i][j] = 0.

```
public class Solution {
    public static void printSolution(int sol[][]) {
        for (int i = 0; i<sol.length; i++) {
            for (int j = 0; j<sol.length; j++) {
                System.out.print(" " + sol[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static boolean isSafe(int maze[][], int x, int y) {
        // if (x, y outside maze) return false
        return (x >= 0 && x < maze.length
            && y >= 0 && y < maze.length && maze[x][y] == 1);
    }

    public static boolean solveMaze(int maze[][]) {
        int N = maze.length;
        int sol[][] = new int[N][N];
        if (solveMazeUtil(maze, 0, 0, sol) == false) {
            System.out.print("Solution doesn't exist");
            return false;
        }
        printSolution(sol);
        return true;
```

```
    }

    public static boolean solveMazeUtil(int maze[][], int x, int y, int sol[][]) {
        if (x == maze.length - 1 && y == maze.length - 1 && maze[x][y] == 1) {
            sol[x][y] = 1;
            return true;
        }

        // Check if maze[x][y] is valid
        if (isSafe(maze, x, y) == true) {
            if (sol[x][y] == 1)
                return false;
            sol[x][y] = 1;
            if (solveMazeUtil(maze, x + 1, y, sol))
                return true;
            if (solveMazeUtil(maze, x, y + 1, sol))
                return true;
            sol[x][y] = 0;
            return false;
        }

        return false;
    }

    public static void main(String args[]){
        int maze[][] = { { 1, 0, 0, 0 },
                         { 1, 1, 0, 1 },
                         { 0, 1, 0, 0 },
                         { 1, 1, 1, 1 } };

        solveMaze(maze);
    }
}
```

## Solution 2:

```
public class Solution {
   final static char[][] L = {{},{},{'a','b','c'},{'d','e','f'},{'g','h','i'},
                       {'j','k','l'},{'m','n','o'},{'p','q','r','s'},
                       {'t','u','v'},{'w','x','y','z'}};

   public static void letterCombinations(String D) {
       int len = D.length();
       if (len == 0) {
           System.out.println("");
           return;
       }
       bfs(0, len, new StringBuilder(), D);
   }

   public static void bfs(int pos, int len, StringBuilder sb, String D) {
       if (pos == len){
           System.out.println(sb.toString());
       }
       else {
           char[] letters = L[Character.getNumericValue(D.charAt(pos))];
           for (int i = 0; i < letters.length; i++)
               bfs(pos+1, len, new StringBuilder(sb).append(letters[i]), D);
       }
   }
   public static void main(String args[]){
       letterCombinations("2");
   }
}
```

## Solution 3 :

```
public class Solution {
   static int N = 8;

   public static boolean isSafe(int x, int y, int sol[][]){
```

```
    return (x >= 0 && x < N && y >= 0 && y < N
            && sol[x][y] == -1);
}

public static void printSolution(int sol[][]) {
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++)
            System.out.print(sol[x][y] + " ");
        System.out.println();
    }
}

public static boolean solveKT() {
    int sol[][] = new int[8][8];
    for (int x = 0; x < N; x++)
        for (int y = 0; y < N; y++)
            sol[x][y] = -1;

    int xMove[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
    int yMove[] = { 1, 2, 2, 1, -1, -2, -2, -1 };

    //As the Knight starts from cell(0,0)
    sol[0][0] = 0;

    if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
        System.out.println("Solution does not exist");
        return false;
    }
    else
        printSolution(sol);

    return true;
}

public static boolean solveKTUtil(int x, int y, int movei, int sol[][],
                                  int xMove[],int yMove[]) {
    int k, next_x, next_y;
    if (movei == N * N)
        return true;
```

```
        for (k = 0; k < 8; k++) {
            next_x = x + xMove[k];
            next_y = y + yMove[k];
            if (isSafe(next_x, next_y, sol)) {
                sol[next_x][next_y] = movei;
                if (solveKTUtil(next_x, next_y, movei + 1,
                                sol, xMove, yMove))
                    return true;
                else
                    sol[next_x][next_y]
                        = -1; // backtracking
            }
        }
        return false;
    }

    public static void main(String args[]){
        solveKT();
    }
}
```

# ARRAYLIST QUESTIONS

## Question 1 :

**Monotonic ArrayList (EASY)**

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

An Arraylist nums is monotone increasing if for all i <= j, nums.get(i) <= nums.get(j). An Arraylist nums is monotone decreasing if for all i <= j, nums.get(i) >= nums.get(j).

Given an integer Arraylist nums, return true if the given list is monotonic, or false otherwise.

**Sample Input 1** : nums = [1,2,2,3]
**Sample Output 1** : true

**Sample Input 2** : nums = [6,5,4,4]
**Sample Output 2** : true

**Sample Input 3** : nums = [1,3,2]
**Sample Output 3** : false

**Constraints** :

- 1 <= nums.size() <= 105
- -105 <= nums.get(i) <= 105

## Question 2 :

**Lonely Numbers in ArrayList (MEDIUM)**

You are given an integer arraylist nums. A number x is lonely when it appears only once, and no adjacent numbers (i.e. x + 1 and x - 1) appear in the arraylist.

Return all lonely numbers in nums. You may return the answer in any order.

**Sample Input 1** : nums = [10,6,5,8]
**Sample Output 1** : [10,8]
Explanation :
- 10 is a lonely number since it appears exactly once and 9 and 11 does not appear in nums.
- 8 is a lonely number since it appears exactly once and 7 and 9 does not appear in nums.

- 5 is not a lonely number since 6 appears in nums and vice versa.
Hence, the lonely numbers in nums are [10, 8].
Note that [8, 10] may also be returned.

**Sample Input 2** : nums = [1,3,5,3]
**Sample Output 2** : [1,5]
Explanation :
- 1 is a lonely number since it appears exactly once and 0 and 2 does not appear in nums.
- 5 is a lonely number since it appears exactly once and 4 and 6 does not appear in nums.
- 3 is not a lonely number since it appears twice.
Hence, the lonely numbers in nums are [1, 5].
Note that [5, 1] may also be returned.

**Constraints** :

- 1 <= nums.size() <= 105
- 0 <= nums.get(i) <= 106

## Question 3 :

**Most Frequent Number following Key (EASY)**

You are given an integer Arraylist nums. You are also given an integer key, which is present in nums.

For every unique integer target in nums, count the number of times target immediately follows an occurrence of key in nums. In other words, count the number of indices i such that:

0 <= i <= nums.size() - 2,
nums.get(i) == key and,
nums.get(i+1) == target.
Return the target with the maximum count.

(Assumption - that the target with maximum count is unique.)

**Sample Input 1** :nums = [1,100,200,1,100], key = 1
**Sample Output 1** : 100
Explanation :
For target = 100, there are 2 occurrences at indices 1 and 4 which follow an occurrence of key.
No other integers follow an occurrence of key, so we return 100.

**Sample Input 2** : nums = [2,2,2,2,3], key = 2
**Sample Output 2** : 2
Explanation :
For target = 2, there are 3 occurrences at indices 1, 2, and 3 which follow an occurrence of key
For target = 3, there is only one occurrence at index 4 which follows an occurrence of key.
target = 2 has the maximum number of occurrences following an occurrence of key, so we return 2.

**Constraints** :

- 2 <= nums.size() <= 1000
- 1 <= nums.get(i) <= 1000
- Assume that the answer is unique.

**Hints** : Count the number of times each target value follows the key in the arraylist.
Choose the target with the maximum count and return it.

## Question 4 :

**Beautiful ArrayList (MEDIUM)**
An Arraylist nums of size n is beautiful if:

nums is a permutation of the integers in the range [1, n].
For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums.get(k) == nums.get(i) + nums.get(j).
Given the integer n, return any beautiful arraylist nums of size n. There will be at least one valid answer for the given n.

**Sample Input 1** : n = 4
**Sample Output 1** : [2,1,4,3]

**Sample Input 2** : n = 5
**Sample Output 2** : [3,1,2,5,4]

**Constraints** :

- 1 <= n <= 1000

# ARRAYLIST SOLUTIONS

## Solution 1:

```
public boolean isMonotonic(ArrayList<Integer> A) {
        boolean inc = true;
        boolean dec = true;
        for (int i=0; i<A.size()-1; i++) {
            if (A.get(i) > A.get(i+1))
                inc = false;
            if (A.get(i) < A.get(i+1))
                dec = false;
        }

        return inc || dec;
    }
```

## Solution 2:

```
public ArrayList<Integer> findLonely(ArrayList<Integer> nums) {
        Collections.sort(nums);
        ArrayList<Integer> list = new ArrayList<>();
        for (int i=1; i < nums.size()-1; i++) {
              if (nums.get(i-1) + 1 < nums.get(i) && nums.get(i) + 1 < nums.get(i+1))
{
                 list.add(nums.get(i));
            }
        }
        if (nums.size() == 1) {
            list.add(nums.get(0));
        }
        if (nums.size() > 1) {
            if (nums.get(0) + 1 < nums.get(1)) {
                list.add(nums.get(0));
            }
            if (nums.get(nums.size()-2) + 1 < nums.get(nums.size()-1)) {
```

```
                list.add(nums.get(nums.size()-1));
            }
        }
        return list;
    }
```

## Solution 3 :

```
public int mostFrequent(ArrayList<Integer> nums, int key) {
        int[]result = new int[1000];

        for(int i=0; i<nums.size()-1; i++){
            if(nums.get(i) == key){
                result[nums.get(i+1)-1]++;
            }
        }

        int max = Integer.MIN_VALUE;
        int ans = 0;

        for(int i=0; i<1000; i++){
            if(result[i] > max){
                max = result[i];
                ans = i+1;
            }
        }

        return ans;
    }
```

## Solution 4 :

Approach 1 (Iterative)

We can see that if we separate odd and even numbers then there is no possibility that even numbers will violate the rule with odd numbers and vice versa. Now we have to arrange even and odd numbers in such a way that they do not violate rules with themselves. For doing so

first let's say we have a beautiful arraylist of size n and we want to make n+1 size of arraylist so what we do is first put all odd numbers that lie within 1 to n+1 and then even(WE can do even then odd also) . Beautiful arraylist has the property that if we multiply any number with arraylist then it still remains beautiful or if we add or subtract any number from arraylist then it still remains beautiful. For only obtaining even number from n size arraylist we do 2* num and for obtaining odd size arraylist we do 2 * n-1 .

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> ans = new ArrayList<>();
        ans.add(1);

        for(int i=2;i<=n;i++){
            ArrayList<Integer>temp=new ArrayList<>();
            for(Integer e:ans){
                if(2*e<=n)temp.add(e*2);
            }
            for(Integer e:ans){
                if(2*e-1<=n)temp.add(e*2-1);
            }

            ans=temp;
        }

        return ans;
    }
```

Approach 2 (Divide & Conquer)

Let's start from a simple 3 numbers case: (1, 2, 3) -> the only thing we need to do is move 2 out of 1 and 3 -> (1, 3, 2).

Then what if the case is (1, 5, 9 ) which has increment = 4? It's the same thing -> move 3 out of 1 and 5 -> (1, 9, 5).

Now, what if the case is (1, 3, 5, 7, 9) ? With the odd + even or divide + conquer idea in mind, we can simply divide it to (1, 5, 9) and (3, 7). Since no change is needed for the 2 numbers case, after following the above steps, we can conquer them to (1, 9, 5, 3, 7).

```
public ArrayList<Integer> beautifulArray(int n) {
        ArrayList<Integer> res = new ArrayList<>();
        divideConque(1, 1, res, n);
        return res;
    }
```

```
    private void divideConque(int start, int increment, ArrayList<Integer> res, int
n) {
        if (start + increment > n) {
            res.add(start);
            return;
        }
        divideConque(start, 2 * increment, res, n);
        divideConque(start + increment, 2 * increment, res, n);
    }
```

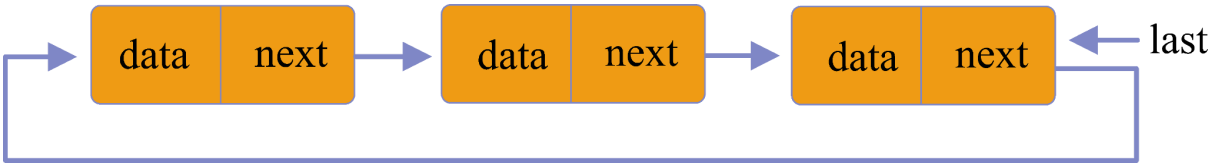
# Circular Linked List

**What is Circular Linked List?**

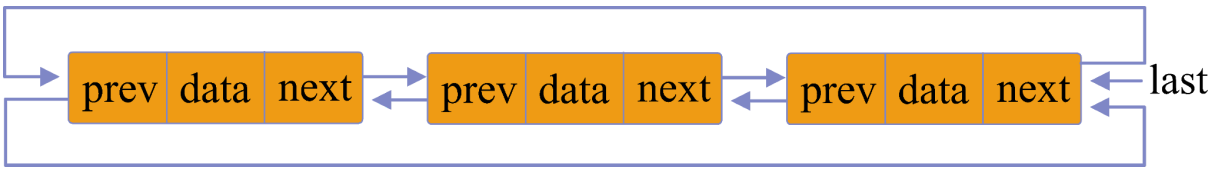
The circular linked list is a linked list in which all nodes form a circle. The initial and last nodes in a circular linked list are linked to each other, forming a circle. There is no NULL at the end.

There are two types of Circular Linked List.

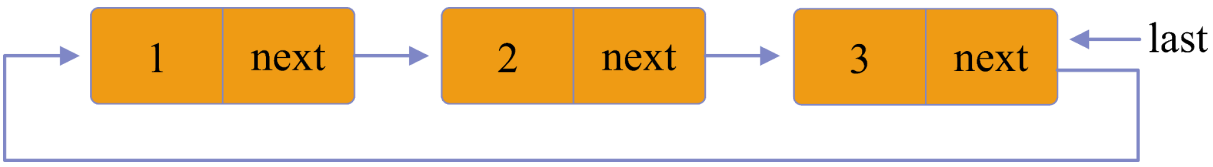
- **Singly Circular Linked List**: The last node of a circular Singly linked list contains a pointer to the beginning node of the list. We go around the circular singly linked list till we get back to where we started. There is no beginning or finish to the circular singly linked list. There is no null value in the next section of any of the nodes.




- **Doubly Linked List**: Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous. Next, pointer and the last node points to the first node by the next pointer and the first node points to the last node by the previous pointer.




**Representation of Circular Linked List**




```
public class Node{
```

```
    int data;
    Node next;
    public Node(int data) {
       this.data = data;
    }
}
```

**Operations on the circular linked list:**

We can do some operations on the circular linked list similar to the singly linked list which are:

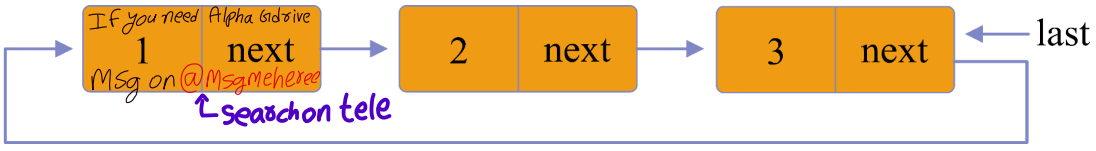
1. Insertion
2. Deletion

## Insertion into a Circular Linked List

We can introduce elements into a circular linked list at three different positions:

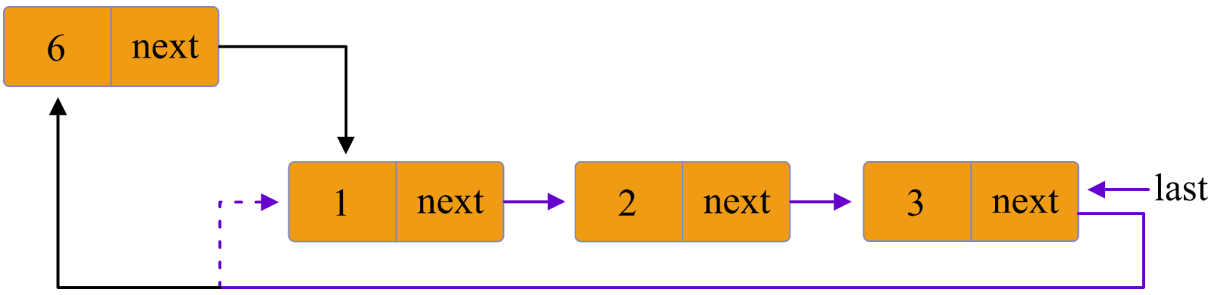
- Insertion in the beginning
- Insertion between nodes
- Insertion toward the end

Assume we have a circular linked list with the elements 1, 2, and 3.



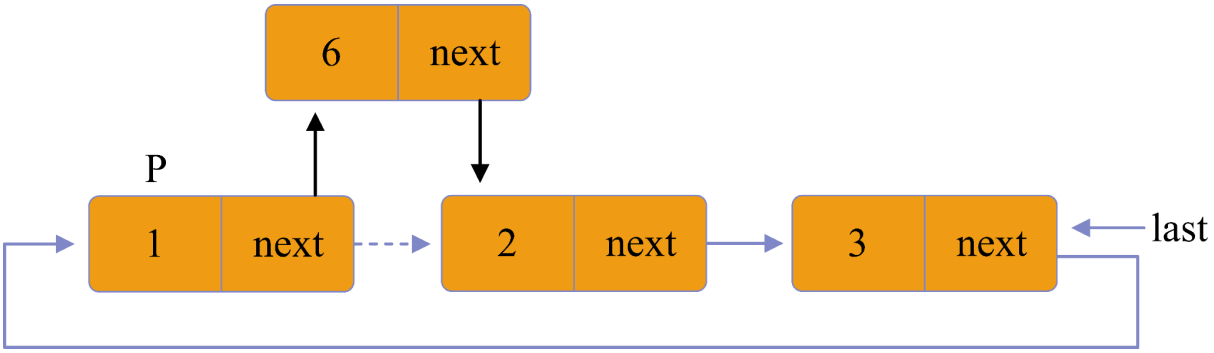

### 1. Insertion at the Start

=> Store the address of the current first node in the newNode (i.e. directing the newNode to the current first node) point the last node to the newNode (i.e making newNode as head)



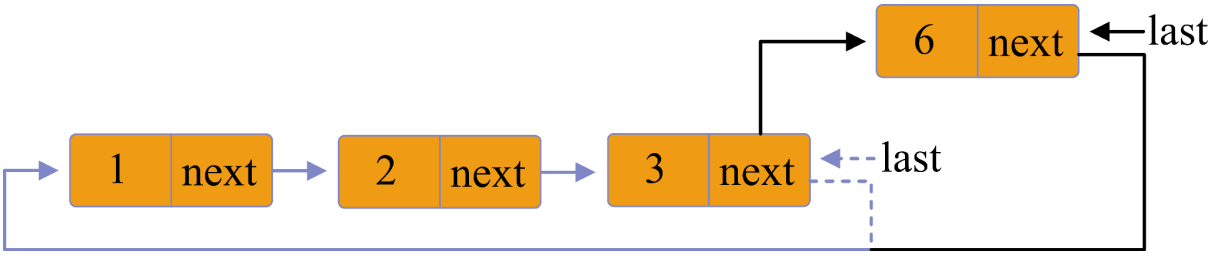

2. **Insertion between two nodes**

=> Insert newNode after the first node.travel to the supplied node (let this node be p) direct the next of newNode to the node next to p put the address of newNode at next of p.



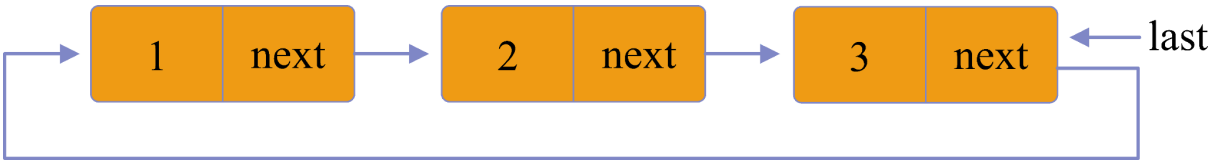

3. **Insert at the End**

=> Insert the address of the head node to the next of newNode (making newNode the last node) make newNode the final node by pointing the current last node to newNode.



## Deletion in a Circular Linked List

Let's say we have a singly circular linked list.

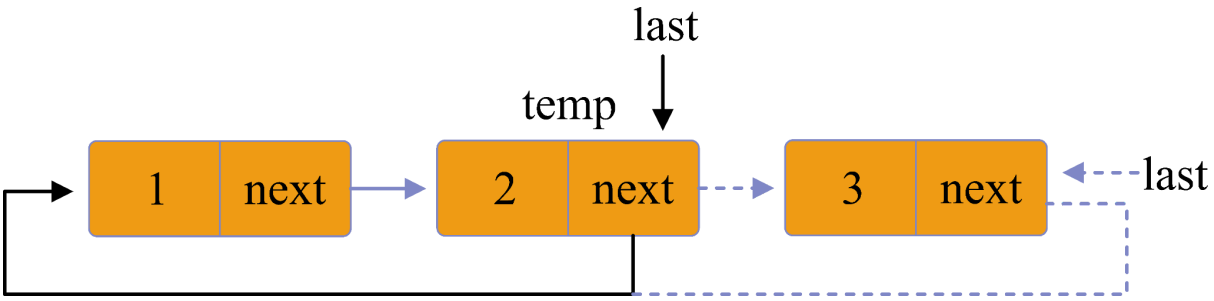



1. **Delete Single Node**

=> If the node to be destroyed is the sole node, free the node's memory and store NULL in the final node.

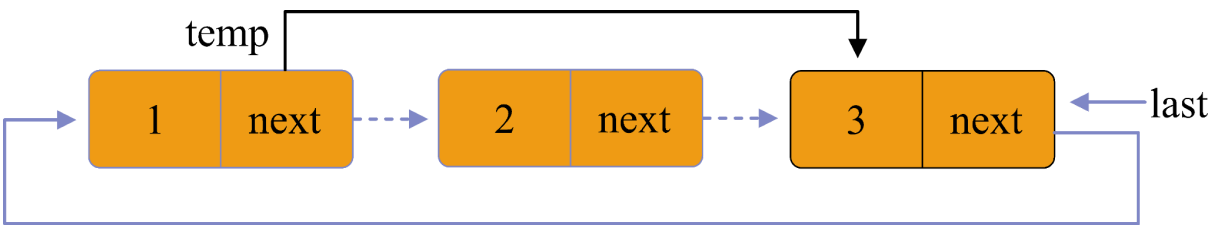
2. **Delete Last Node**

=> If the last node is to be eliminated, locate the node preceding the last node (let it be temp), save the address of the node following the last node in temp free the memory of the last make temp as the last node




3. **Delete Any Other Node**

=> If any additional nodes are to be removed, go to the node to be deleted (in this case, node 2), let the node before node 2 be temp store the address of the node next to 2 in temp free the memory of 2



**Implementing Insertion, Deletion and Printing in Circular Linked List**

```
class Solution {
  static class Node {
    int data;
    Node next;
  };

  static Node addToEmpty(Node last, int data) {
    if (last != null)
      return last;
    Node newNode = new Node();
    newNode.data = data;
    last = newNode;
    newNode.next = last;
    return last;
  }
  static Node addFront(Node last, int data) {
    if (last == null)
      return addToEmpty(last, data);
    Node newNode = new Node();
    newNode.data = data;
    newNode.next = last.next;
    last.next = newNode;
    return last;
  }

  static Node addEnd(Node last, int data) {
    if (last == null)
      return addToEmpty(last, data);
    Node newNode = new Node();
    newNode.data = data;
    newNode.next = last.next;
    last.next = newNode;
    last = newNode;
    return last;
```

```
}

static Node addAfter(Node last, int data, int item) {
  if (last == null)
    return null;

  Node newNode, p;
  p = last.next;
  do {
    if (p.data == item) {
      newNode = new Node();
      newNode.data = data;
      newNode.next = p.next;
      p.next = newNode;
      if (p == last)
        last = newNode;
      return last;
    }
    p = p.next;
  } while (p != last.next);
  System.out.println(item + "The given node is not present in the list");
  return last;
}

static Node deleteNode(Node last, int key) {
  if (last == null)
    return null;
  if (last.data == key && last.next == last) {
    last = null;
    return last;
  }

  Node temp = last, d = new Node();
  if (last.data == key) {
    while (temp.next != last) {
      temp = temp.next;
    }
```

```
    temp.next = last.next;
    last = temp.next;
  }
  while (temp.next != last && temp.next.data != key) {
    temp = temp.next;
  }
  if (temp.next.data == key) {
    d = temp.next;
    temp.next = d.next;
  }
  return last;
}

static void traverse(Node last) {
  Node p;
  if (last == null) {
    System.out.println("List is empty.");
    return;
  }
  p = last.next;
  do {
    System.out.print(p.data + " ");
    p = p.next;
  }
  while (p != last.next);
}

public static void main(String[] args) {
  Node last = null;
  last = addToEmpty(last, 6);
  last = addEnd(last, 8);
  last = addFront(last, 2);
  last = addAfter(last, 10, 2);
  traverse(last);
  deleteNode(last, 8);
  traverse(last);
}
```

```
}
```

Thanks for reading this article till the end.

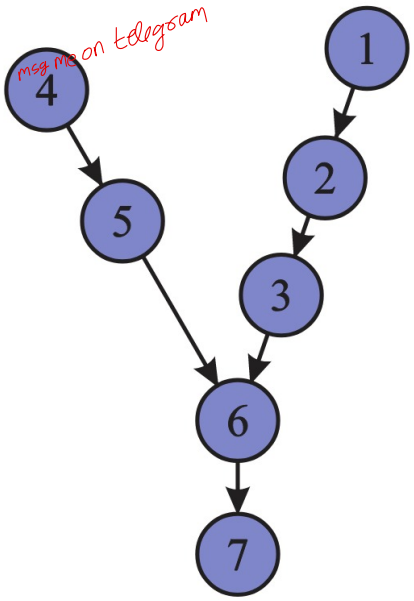
# Linked List Questions

Note - These are classical questions. Please positively solve them.

## Question 1 :

**Intersection of Two Linked Lists**

In a system there are two singly linked list. By some programming error, the end node of one of the linked lists got linked to the second list, forming an inverted Y-shaped list. Write a program to get the point where two linked lists merge.




We have to find the intersection part in this system.

## Question 2 :

**Delete N Nodes After M Nodes of a Linked List**

We have a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list. Difficulty Level: Rookie.

**Sample Input 1** : M=2 N=2 LL: 1->2->3->4->5->6->7->8
**Sample Output 1** : 1->2->5->6

**Sample Input 2** : M=3 N=2 LL: 1->2->3->4->5->6->7->8->9->10
**Sample Output 2** : 1->2->3->6->7->8

## Question 3 :

**Swapping Nodes in a Linked List**

We have a linked list and two keys in it, swap nodes for two given keys. Nodes should be swapped by changing links. Swapping data of nodes may be expensive in many situations when data contains many fields. It may be assumed that all keys in the linked list are distinct.

**Sample Input 1** : 1->2->3->4, x = 2, y = 4
**Sample Output 1** : 1->4->3->2

## Question 4 :

**Odd Even Linked List**

We have a Linked List of integers, write a function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers same.

**Sample Input 1** : 8->12->10->5->4->1->6->NULL
**Sample Output 1** : 8->12->10->4->6->5->1->NULL

**Sample Input 2** : 1->3->5->7->NULL
**Sample Output 2** : `1->3->5->7->NULL`

## Question 5 :

**Merge k Sorted Lists**

We have K sorted linked lists of size N each, merge them and print the sorted output.

**Sample Input 1** : `k = 2, n = 2`
`l1 = 1->3->NULL`
`l2 = 6->8->NULL`
`l3 = 9->10->NULL`

**Sample Output 1** : 1>3->6->8->9->10->NULL

# Linked List Solutions

## Solution 1:

Time Complexity : o(m*n)
Space Complexity: o(1)

```
class Solution {

    static class Node {
        int data;
        Node next;
        Node(int d){
            data = d;
            next = null;
        }
    }

    public Node getIntersectionNode(Node head1, Node head2)
    {
        while (head2 != null) {
            Node temp = head1;
            while (temp != null) {
                if (temp == head2) {
                    return head2;
                }
                temp = temp.next;
            }
            head2 = head2.next;
        }
        return null;
    }

    public static void main(String[] args){
        Solution list = new Solution();

        Node head1, head2;
        head1 = new Node(10);
        head2 = new Node(3);
```

```
        Node newNode = new Node(6);
        head2.next = newNode;

        newNode = new Node(9);
        head2.next.next = newNode;

        newNode = new Node(15);
        head1.next = newNode;
        head2.next.next.next = newNode;

        newNode = new Node(30);
        head1.next.next = newNode;

        head1.next.next.next = null;

        Node intersectionPoint
                = list.getIntersectionNode(head1, head2);

        if (intersectionPoint == null) {
                System.out.print(" No Intersection Point \n");
        }
        else {
                System.out.print("Intersection Point: "
                                                + intersectionPoint.data);
        }
    }
}
```

## Solution 2:

Time Complexity  : o(n)
Space Complexity: o(1)

```
import java.util.*;
class Solution{

static class Node{
        int data;
        Node next;
```

```
};

static Node push( Node head_ref, int new_data){
        Node new_node = new Node();
        new_node.data = new_data;
        new_node.next = (head_ref);
        (head_ref) = new_node;
        return head_ref;
}

static void printList( Node head){
        Node temp = head;
        while (temp != null){
                System.out.printf("%d ", temp.data);
                temp = temp.next;
        }
        System.out.printf("\n");
}

static void skipMdeleteN( Node head, int M, int N){
        Node curr = head, t;
        int count;

        while (curr!=null){
                for (count = 1; count < M && curr != null; count++)
                        curr = curr.next;

                if (curr == null)
                        return;
                t = curr.next;
                for (count = 1; count <= N && t != null; count++){
                        Node temp = t;
                        t = t.next;
                }

                curr.next = t;
                curr = t;
        }
}
```

```
public static void main(String args[]){
        Node head = null;
        int M=2, N=3;
        head=push(head, 10);
        head=push(head, 9);
        head=push(head, 8);
        head=push(head, 7);
        head=push(head, 6);
        head=push(head, 5);
        head=push(head, 4);
        head=push(head, 3);
        head=push(head, 2);
        head=push(head, 1);

        System.out.printf("M = %d, N = %d \n" +
                                        "Linked list we have is :\n", M, N);
        printList(head);

        skipMdeleteN(head, M, N);

        System.out.printf("\nLinked list on deletion is :\n");
        printList(head);
}
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Node {
        int data;
        Node next;
        Node(int d){
                data = d;
                next = null;
        }
```

```
}

class Solution {
    Node head;

    public void swapNodes(int x, int y){
        if (x == y)
            return;

        Node prevX = null, currX = head;
        while (currX != null && currX.data != x) {
            prevX = currX;
            currX = currX.next;
        }

        Node prevY = null, currY = head;
        while (currY != null && currY.data != y) {
            prevY = currY;
            currY = currY.next;
        }

        if (currX == null || currY == null)
            return;

        if (prevX != null)
            prevX.next = currY;
        else
            head = currY;

        if (prevY != null)
            prevY.next = currX;
        else
            head = currX;

        Node temp = currX.next;
        currX.next = currY.next;
        currY.next = temp;
    }

    public void push(int new_data){
```

```
        Node new_Node = new Node(new_data);
        new_Node.next = head;
        head = new_Node;
    }

    public void printList(){
        Node tNode = head;
        while (tNode != null) {
            System.out.print(tNode.data + " ");
            tNode = tNode.next;
        }
    }

    public static void main(String[] args){
        Solution llist = new Solution();

        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.print(
            "\n Linked list before ");
        llist.printList();

        llist.swapNodes(4, 3);

        System.out.print(
            "\n Linked list after ");
        llist.printList();
    }
}
```

## Solution 4 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Solution{
    Node head;
    class Node{
        int data;
        Node next;
        Node(int d){
            data = d;
            next = null;
        }
    }

    void segregateEvenOdd(){
        Node end = head;
        Node prev = null;
        Node curr = head;

        while (end.next != null)
            end = end.next;

        Node new_end = end;

        while (curr.data %2 !=0 && curr != end){
            new_end.next = curr;
            curr = curr.next;
            new_end.next.next = null;
            new_end = new_end.next;
        }

        if (curr.data %2 ==0){
            head = curr;
            while (curr != end){
                if (curr.data % 2 == 0){
                    prev = curr;
                    curr = curr.next;
                }
                else{
                    prev.next = curr.next;
                    curr.next = null;
                    new_end.next = curr;
```

```
                    new_end = curr;
                    curr = prev.next;
                }
            }
        }
        else prev = curr;
        if (new_end != end && end.data %2 != 0){
            prev.next = end.next;
            end.next = null;
            new_end.next = end;
        }
    }
    void push(int new_data){
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }

    void printList(){
        Node temp = head;
        while(temp != null){
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    public static void main(String args[]){
        Solution llist = new Solution();
        llist.push(11);
        llist.push(10);
        llist.push(8);
        llist.push(6);
        llist.push(4);
        llist.push(2);
        llist.push(0);
        System.out.println("Linked List");
        llist.printList();

        llist.segregateEvenOdd();
```

```
            System.out.println("updated Linked List");
            llist.printList();
        }
}
```

## Solution 5 :

Time Complexity : o(n*logk)
Space Complexity: o(n)

```
public class Solution {
    public static Node SortedMerge(Node a, Node b){
        Node result = null;
        if (a == null)
            return b;
        else if (b == null)
            return a;
        if (a.data <= b.data) {
            result = a;
            result.next = SortedMerge(a.next, b);
        }
        else {
            result = b;
            result.next = SortedMerge(a, b.next);
        }

        return result;
    }

    public static Node mergeKLists(Node arr[], int last)
    {
        while (last != 0) {
            int i = 0, j = last;
            while (i < j) {
                arr[i] = SortedMerge(arr[i], arr[j]);
                i++;
                j--;
                if (i >= j)
                    last = j;
            }
```

```
            }

            return arr[0];
        }
        public static void printList(Node node){
            while (node != null) {
                System.out.print(node.data + " ");
                node = node.next;
            }
        }

        public static void main(String args[]){
            int k = 3;
            int n = 4;
            Node arr[] = new Node[k];

            arr[0] = new Node(1);
            arr[0].next = new Node(3);
            arr[0].next.next = new Node(5);
            arr[0].next.next.next = new Node(7);

            arr[1] = new Node(2);
            arr[1].next = new Node(4);
            arr[1].next.next = new Node(6);
            arr[1].next.next.next = new Node(8);

            arr[2] = new Node(0);
            arr[2].next = new Node(9);
            arr[2].next.next = new Node(10);
            arr[2].next.next.next = new Node(11);

            Node head = mergeKLists(arr, k - 1);
            printList(head);
        }
}

class Node {
        int data;
        Node next;
        Node(int data){
```

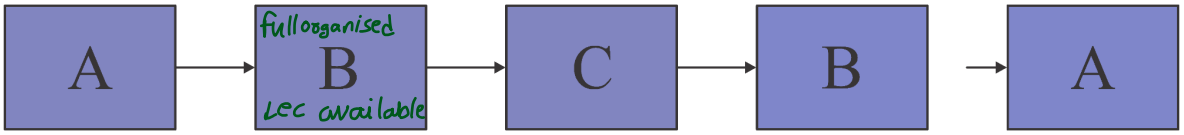
```
        this.data = data;
    }
}
```

# Stack Questions

## Question 1 :

**Palindrome Linked List**

We have a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.




**Input** : A->B->C->B->A

**Output** : Yes It is Palindrome

## Question 2 :

**Simplify Path**

We hava an absolute path for a file (Unix-style), simplify it. Note that absolute path always begin with '/' ( root directory ), a dot in path represent current directory and double dot represents parent directory.

**Sample Input 1** : /apnacollege/

**Sample Output 1** : /apnacollege

**Sample Input 1** : /a/..

**Sample Output 1** : /

## Question 3 :

**Decode a string**

We have an encoded string s and the task is to decode it. The pattern in which the strings are encoded is as follows.

**Sample Input 1** : 2[cv]

**Sample Output 1** : cvcv

**Sample Input 2** : 3[b2[v]]L

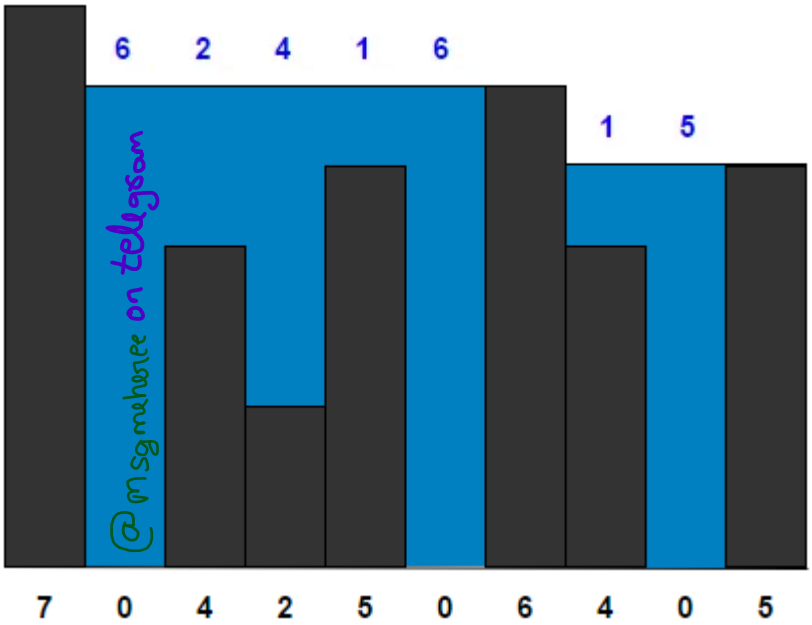
**Sample Output 2** : bvvbvvbvv

## Question 4 :

**Trapping Rain Water**

We have an array of N non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

Note: We have already solved this Question using Arrays but you have to now solve this using a Stack.




**Sample Input 1** : [7 0 4 2 5 0 6 4 0 6]

**Sample Output 1** : 25

# Stack Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

class Solution {
	public static void main(String args[]){
		Node one = new Node(1);
		Node two = new Node(2);
		Node three = new Node(3);
		Node four = new Node(4);
		Node five = new Node(3);
		Node six = new Node(2);
		Node seven = new Node(1);
		one.ptr = two;
		two.ptr = three;
		three.ptr = four;
		four.ptr = five;
		five.ptr = six;
		six.ptr = seven;
		boolean condition = isPalindrome(one);
		System.out.println("Palindrome :" + condition);
	}
	static boolean isPalindrome(Node head){

		Node slow = head;
		boolean ispalin = true;
		Stack<Integer> stack = new Stack<Integer>();

		while (slow != null) {
			stack.push(slow.data);
			slow = slow.ptr;
		}

		while (head != null) {
```

```
                int i = stack.pop();
                if (head.data == i) {
                        ispalin = true;
                }
                else {
                        ispalin = false;
                        break;
                }
                head = head.ptr;
            }
            return ispalin;
        }
}

class Node {
        int data;
        Node ptr;
        Node(int d){
            ptr = null;
            data = d;
        }
}
```

## Solution 2 :

Time Complexity : o(n)
Space Complexity: o(1)

```
import java.io.*;
import java.util.*;

class Solution{
        public static void main(String []args){
            String str = new String("/a/./b/../../c/");
            String res = simplify(str);
            System.out.println(res);
        }
```

```
static String simplify(String A){
        Stack<String> st = new Stack<String>();
        String res = "";
        res += "/";
        int len_A = A.length();

        for (int i = 0; i < len_A; i++){
                String dir = "";
                while (i < len_A && A.charAt(i) == '/')
                        i++;

                while (i < len_A && A.charAt(i) != '/'){
                        dir += A.charAt(i);
                        i++;
                }

                if (dir.equals("..") == true){
                        if (!st.empty())
                                st.pop();
                }

                else if (dir.equals(".") == true)
                        continue;

                else if (dir.length() != 0)
                        st.push(dir);
        }

        Stack<String> st1 = new Stack<String>();
        while (!st.empty()){

                st1.push(st.pop());
        }

        while (!st1.empty()){
                if (st1.size() != 1)
                        res += (st1.pop() + "/");
                else
                        res += st1.pop();
        }
```

```
                return res;
        }

}
```

## Solution 3 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.util.Stack;

class Solution{
        static String decode(String str){
                Stack<Integer> integerstack = new Stack<>();
                Stack<Character> stringstack = new Stack<>();
                String temp = "", result = "";
                for (int i = 0; i < str.length(); i++){
                        int count = 0;
                        if (Character.isDigit(str.charAt(i))){
                                while (Character.isDigit(str.charAt(i))){
                                        count = count * 10 + str.charAt(i) - '0';
                                        i++;
                                }

                                i--;
                                integerstack.push(count);
                        }

                        else if (str.charAt(i) == ']'){
                                temp = "";
                                count = 0;

                                if (!integerstack.isEmpty()){
                                        count = integerstack.peek();
                                        integerstack.pop();
                                }

                                while (!stringstack.isEmpty() && stringstack.peek()!='[' ){
```

```
                    temp = stringstack.peek() + temp;
                    stringstack.pop();
                }

                if (!stringstack.empty() && stringstack.peek() == '[')
                    stringstack.pop();

                for (int j = 0; j < count; j++)
                    result = result + temp;

                for (int j = 0; j < result.length(); j++)
                    stringstack.push(result.charAt(j));

                result = "";
            }

            else if (str.charAt(i) == '['){
                if (Character.isDigit(str.charAt(i-1)))
                    stringstack.push(str.charAt(i));

                else{
                    stringstack.push(str.charAt(i));
                    integerstack.push(1);
                }
            }

            else
                stringstack.push(str.charAt(i));
        }

        while (!stringstack.isEmpty()){
            result = stringstack.peek() + result;
            stringstack.pop();
        }
        return result;
}

public static void main(String args[]){
        String str = "3[b2[ca]]";
        System.out.println(decode(str));
```

```
        }
}
```

## Solution 4 :

Time Complexity  : o(n)
Space Complexity: o(n)

```
import java.io.*;
import java.util.*;

class Solution{

        public static int maxWater(int[] height){
                Stack<Integer> stack = new Stack<>();
                int n = height.length;
                int ans = 0;
                for (int i = 0; i < n; i++) {
                        while ((!stack.isEmpty())
                                && (height[stack.peek()] < height[i])) {
                                int pop_height = height[stack.peek()];
                                stack.pop();
                                if (stack.isEmpty())
                                        break;
                                int distance = i - stack.peek() - 1;
                                int min_height
                                        = Math.min(height[stack.peek()],
                                                        height[i])
                                        - pop_height;

                                ans += distance * min_height;
                        }
                        stack.push(i);
                }

                return ans;
        }

        public static void main(String[] args){
                int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
```

```
        System.out.print(maxWater(arr));
    }
}
```

# Queue Questions

## Question 1 :

**Generate Binary Numbers**

Given a number N. The task is to generate and print all binary numbers with decimal values from 1 to N.



**Sample Input 1** : N =2
**Sample Output 1** : 1 10

**Sample Input 2** : 5.
**Sample Output 2** : 1 10 11 100 101

## Question 2 :

**Connect n ropes with minimum cost**

Given are N ropes of different lengths, the task is to connect these ropes into one rope with minimum cost, such that the cost to connect two ropes is equal to the sum of their lengths.

**Sample Input 1** : N = 4, arr = [4 3 2 6]
**Sample Output 1** : 29

**Sample Input 2** : N = 2, arr = [1 2 3]
**Sample Output 2** : 9

## Question 3 :

**Job Sequencing Problem**

We have an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. Maximize the total profit if only one job can be scheduled at a time.

**Sample Input 1** :

| JobID | Deadline | Profit |
|---|---|---|
| a | 4 | 20 |
| b | 1 | 10 |
| c | 1 | 40 |
| d | 1 | 30 |

**Sample Output 1** : c, a

## Question 4 :

**Reversing the first K elements of a Queue**

We have an integer k and a queue of integers, we need to reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

**Sample Input 1** : `Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100] ,k=5`

**Sample Output 1** : Q = [50, 40, 30, 20, 10, 60, 70, 80, 90, 100]

## Question 5 :

**Maximum of all subarrays of size k**

We have an array arr[] of size N and an integer K. Find the maximum for each and ever contiguous subarray of size K.

**Sample Input 1** : `N=9, K=3 arr= 1 2 3 1 4 5 2 3 6`

**Sample Output 1** : `3 3 4 5 5 5 6`

# Queue Solutions

## Solution 1 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.LinkedList;
import java.util.Queue;

public class Solution {
	static void generatePrintBinary(int n){
		Queue<String> q = new LinkedList<String>();
		q.add("1");
		while (n-- > 0) {
			String s1 = q.peek();
			q.remove();
			System.out.println(s1);
			String s2 = s1;
			q.add(s1 + "0");
			q.add(s2 + "1");
		}
	}

	public static void main(String[] args){
		int n = 10;
		generatePrintBinary(n);
	}
}
```

## Solution 2 :

Time Complexity : o(n)
Space Complexity: o(n)

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

```
class Solution{
	static int minCost(int arr[], int n){
		PriorityQueue<Integer> pq
			= new PriorityQueue<Integer>();

		for (int i = 0; i < n; i++) {
			pq.add(arr[i]);
		}

		int res = 0;
		while (pq.size() > 1) {
			int first = pq.poll();
			int second = pq.poll();
			res += first + second;
			pq.add(first + second);
		}
		return res;
	}

	public static void main(String args[]){
		int len[] = { 4, 3, 2, 6 };
		int size = len.length;
		System.out.println("Total cost for connecting"
						+ " ropes is "
						+ minCost(len, size));
	}
}
```

## Solution 3 :

Time Complexity : o(nlogn)
Space Complexity: o(n)

```
import java.util.*;
class Solution {
	static class Job {
		char job_id;
		int deadline;
		int profit;
```

```
        Job(char job_id, int deadline, int profit){
            this.deadline = deadline;
            this.job_id = job_id;
            this.profit = profit;
        }
}

static void printJobScheduling(ArrayList<Job> arr){
        int n = arr.size();
        Collections.sort(arr, (a, b) -> {
            return a.deadline - b.deadline;
        });
        ArrayList<Job> result = new ArrayList<>();
        PriorityQueue<Job> maxHeap = new PriorityQueue<>(
            (a, b) -> { return b.profit - a.profit; });
        for (int i = n - 1; i > -1; i--) {
            int slot_available;
            if (i == 0) {
                slot_available = arr.get(i).deadline;
            }
            else {
                slot_available = arr.get(i).deadline
                                 - arr.get(i - 1).deadline;
            }
            maxHeap.add(arr.get(i));
            while (slot_available > 0
                && maxHeap.size() > 0) {
                Job job = maxHeap.remove();
                slot_available--;
                result.add(job);
            }
        }

        Collections.sort(result, (a, b) -> {
            return a.deadline - b.deadline;
        });

        for (Job job : result) {
            System.out.print(job.job_id + " ");
        }
```

```
            System.out.println();
        }

        public static void main(String[] args){
            ArrayList<Job> arr = new ArrayList<Job>();

            arr.add(new Job('a', 2, 100));
            arr.add(new Job('b', 1, 19));
            arr.add(new Job('c', 2, 27));
            arr.add(new Job('d', 1, 25));
            arr.add(new Job('e', 3, 15));

            System.out.println("Following is maximum "
                                    + "profit sequence of jobs");
            printJobScheduling(arr);
        }
}
```

## Solution 4 :

Time Complexity : o(n+k)
Space Complexity: o(k)

```
import java.io.*;
import java.util.*;

import java.util.*;

class Solution {

    static class cell {
        int x, y;
        int dis;
        public cell(int x, int y, int dis){
            this.x = x;
            this.y = y;
            this.dis = dis;
```

```
        }
}

static boolean isInside(int x, int y, int N){
        if (x >= 1 && x <= N && y >= 1 && y <= N)
                return true;
        return false;
}

static int minStepToReachTarget(
        int knightPos[], int targetPos[],
        int N){
        int dx[] = { -2, -1, 1, 2, -2, -1, 1, 2 };
        int dy[] = { -1, -2, -2, -1, 1, 2, 2, 1 };

        Vector<cell> q = new Vector<>();
        q.add(new cell(knightPos[0], knightPos[1], 0));
        cell t;
        int x, y;
        boolean visit[][] = new boolean[N + 1][N + 1];
        visit[knightPos[0]][knightPos[1]] = true;
        while (!q.isEmpty()) {
                t = q.firstElement();
                q.remove(0);
                if (t.x == targetPos[0] && t.y == targetPos[1])
                        return t.dis;
                for (int i = 0; i < 8; i++) {
                        x = t.x + dx[i];
                        y = t.y + dy[i];
                        if (isInside(x, y, N) && !visit[x][y]) {
                                visit[x][y] = true;
                                q.add(new cell(x, y, t.dis + 1));
                        }
                }
        }
        return Integer.MAX_VALUE;
}

public static void main(String[] args){
        int N = 30;
```

```
                int knightPos[] = { 1, 1 };
                int targetPos[] = { 30, 30 };
                System.out.println(
                        minStepToReachTarget(
                                knightPos, targetPos, N));
        }
}
```

## Solution 5 :

Time Complexity  : o(n)
Space Complexity: o(k)

```
import java.util.Deque;
import java.util.LinkedList;

public class Solution {

   static void printMax(int arr[], int n, int k){

      Deque<Integer> Qi = new LinkedList<Integer>();
      int i;
      for (i = 0; i < k; ++i) {
         while (!Qi.isEmpty() && arr[i] >=
                  arr[Qi.peekLast()])
            Qi.removeLast();
         Qi.addLast(i);
      }
      for (; i < n; ++i) {
         System.out.print(arr[Qi.peek()] + " ");
         while ((!Qi.isEmpty()) && Qi.peek() <=
                              i - k)
            Qi.removeFirst();
         while ((!Qi.isEmpty()) && arr[i] >=
                     arr[Qi.peekLast()])
            Qi.removeLast();
         Qi.addLast(i);
      }
      System.out.print(arr[Qi.peek()]);
   }
```

```
    public static void main(String[] args){
        int arr[] = { 12, 1, 78, 90, 57, 89, 56 };
        int k = 3;
        printMax(arr, arr.length, k);
    }
}
```

# Comparators in Java

A **Comparator** in Java is an interface that is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class.

Let's take an example of a function that compares obj1 with obj2 :
*public int compare(Object obj1, Object obj2):*

There are 2 ways of implementing this sort method -

a. We write a custom function on our own, where we define the sorting logic from scratch.
b. Using the Comparator interface. (Better!)

**How to use sort?**

Comparator interface is used to order/sort the objects of a user-defined class.

This interface is present in the java.util package and contains 2 methods compare(Object obj1, Object obj2) and equals(Object element).

Using a comparator, we can sort the elements based on data members/properties. For instance, it may be on the basis of name, age, height etc.

Method of Collections class for sorting List elements is used to sort the elements of List by the given comparator.
*public void sort(List list, ComparatorClass c)*

**Internal working of the sort( ) method of Collections class**

sort( ) method calls the Compare method of the classes it is sorting.

To compare 2 objects, it asks "Which is greater?"

Compare method returns one of the 3 values : -1, 0, or 1.

-1 -> obj1 is less than obj2

0 -> obj1 is equal to obj2

1 -> obj1 is greater than obj2

It uses this result to then determine if they(obj1 & obj2) should be swapped for their sort.

To define a **Customized Sorting Order**

```
import java.util.ArrayList;
import java.util.Collections;

public class Solution {
   public static void main (String[] args){
       ArrayList<Person> list = new ArrayList<Person>();
       list.add (new Person("Aman", 34));
       list.add (new Person("Akbar", 42));
       list.add (new Person("Anthony", 28));

       Collections.sort (list);
       System.out.println (list);
   }
}

class Person implements Comparable<Person>{
   String name;
   int age;

   Person(String name, int age){
       this.name = name;
       this.age = age;
   }

   /**
    * This function compares 2 objects
    * A person object is passed as an argument
    * and returns negative integer, zero, or a positive integer
    * as this object is less than, equal to, or greater than the specified object.
    */
   @Override
```

```
    public int compareTo(Person person){
        if(this.age == person.age)
            return 0;
        else
            return (this.age < person.age) ? -1 : 1;
    }

    @Override
    public String toString(){
        return this.name + ":" + this.age;
    }
}
```

**Lambda Expressions** in Java

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Format:

*Comparator<ClassName> comparator = Comparator.comparing(o -> o.property);*

Example:

*Comparator<Student> comparator = Comparator.comparing(o -> o.age);*
*Collections.sort(students, comparator);*

# Greedy Questions

## Question 1 :

**Maximum Balanced String Partitions**

We have balanced string str of size N with an equal number of L and R, the task is to find a maximum number X, such that a given string can be partitioned into X balanced substring. A string is called to be balanced if the number of 'L's in the string equals the number of 'R's.

**Input** : "LRRRRLLRLLRL"
**Output** : 3

## Question 2 :

**Kth largest odd number in a given range**

We have two variables L and R, indicating a range of integers from L to R inclusive, and a number K, the task is to find Kth largest odd number. If K > number of odd numbers in the range L to R then return 0.

**Sample Input 1** : L = -3, R = 3, K = 1
**Sample Output 1** : 3

## Question 3 :

**Lexicographically smallest string of length N and sum K**

We have two integers N and K. The task is to print the lexicographically smallest string of length N consisting of lower-case English alphabets such that the sum of the characters of the string equals to K where 'a' = 1, 'b' = 2, 'c' = 3, ..... and 'z' = 26.

**Sample Input 1** : N = 5, K = 42
**Sample Output 1** : aamz

**Sample Input 2** : N = 3, K = 25
**Sample Output 2** : aaw

## Question 4 :

**Best Time to Buy and Sell Stock**

Given an array prices[] of length N, representing the prices of the stocks on different days, the task is to find the maximum profit possible for buying and selling the stocks on different days using transactions where at most one transaction is allowed.

Note: Stock must be bought before being sold.

**Sample Input 1** : `prices[] = {7, 6, 4, 3, 1}`
**Sample Output 1** : 0

**Sample Input 2** : `prices[] = {7, 1, 5, 3, 6, 4]`
**Sample Output 2** : 5

## Question 5 :

**Split the given array into K sub-arrays**

We have an Array[] of N elements and a number K. ( 1 <= K <= N ) . Split the given array into K subarrays (they must cover all the elements). The maximum subarray sum achievable out of K subarrays formed must be the minimum possible. Find that possible subarray sum.

**Sample Input 1** :Array[] = {1, 1, 2} K = 2
**Sample Output 1** : 2

**Sample Input 2** : `Array[] = {1, 2, 3, 4}, K = 3`
**Sample Output 2** : 4

# Greedy Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(1)

```
import java.util.*;

class Solution {

static int BalancedPartition(String str, int n){

        if (n == 0)
                return 0;

        int r = 0, l = 0;
        int ans = 0;
        for(int i = 0; i < n; i++)  {
        if (str.charAt(i) == 'R'){
                r++;
        }

        else if (str.charAt(i) == 'L'){
                l++;
        }

        if (r == l){
                ans++;
        }
        }

        return ans;
}

public static void main(String[] args){
        String str = "LLRRRLLRRL";
        int n = str.length();

        System.out.print(BalancedPartition(str, n) + "\n");
}
}
```

## Solution 2 :

Time Complexity : o(1)
Space Complexity: o(1)

```
class Solution {

	public static int kthOdd(int[] range, int K) {

		if (K <= 0)
			return 0;

		int L = range[0];
		int R = range[1];

		if ((R & 1) > 0) {
			int Count = (int) Math.ceil((R - L + 1) / 2);
			if (K > Count)
				return 0;
			else
				return (R - 2 * K + 2);
		} else {
			int Count = (R - L + 1) / 2;
			if (K > Count)
				return 0;
			else
				return (R - 2 * K + 1);
		}
	}

	public static void main(String args[]){
		int[] p = { -10, 10 };
		int k = 8;
		System.out.println(kthOdd(p, k));
	}
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.Arrays;
public class Main {
	public static char[] lexo_small(int n, int k){
		char arr[] = new char[n];
		Arrays.fill(arr, 'a');
		for (int i = n - 1; i >= 0; i--) {
			k -= i;
			if (k >= 0) {
				if (k >= 26) {
					arr[i] = 'z';
					k -= 26;
				}

				else {
					arr[i] = (char)(k + 97 - 1);
					k -= arr[i] - 'a' + 1;
				}
			}

			else
				break;

			k += i;
		}
		return arr;
	}

	public static void main(String[] args){
		int n = 5, k = 42;

		char arr[] = lexo_small(n, k);

		System.out.print(new String(arr));
	}
}
```

## Solution 4 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Solution {
static int maxProfit(int prices[], int n)
{
        int buy = prices[0], max_profit = 0;
        for (int i = 1; i < n; i++) {
        if (buy > prices[i])
                buy = prices[i];
        else if (prices[i] - buy > max_profit)
                max_profit = prices[i] - buy;
        }
        return max_profit;
}

public static void main(String args[]){
        int prices[] = { 7, 1, 5, 6, 4 };
        int n = prices.length;
        int max_profit = maxProfit(prices, n);
        System.out.println(max_profit);
}
}
```

## Solution 5 :

Time Complexity :O((N−1)c(K−1))
(Here 'c' here depicts combinations i.e. ((n-1)!/((n-k)!*(k-1)!) Where N is the number of elements of the array and K is the number of divisions that we are having)
Space Complexity: o(n)

```
class Solution {
        public static int ans = 10000000;
        public static void solve(int a[], int n, int k,
                                                        int index, int sum, int maxsum){
```

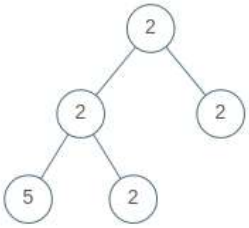
```
            if (k == 1) {
                maxsum = Math.max(maxsum, sum);
                sum = 0;
                for (int i = index; i < n; i++) {
                    sum += a[i];
                }
                maxsum = Math.max(maxsum, sum);
                ans = Math.min(ans, maxsum);
                return;
            }
            sum = 0;
            for (int i = index; i < n; i++) {
                sum += a[i];
                maxsum = Math.max(maxsum, sum);
                solve(a, n, k - 1, i + 1, sum, maxsum);
            }
        }
        public static void main(String[] args){
            int arr[] = { 1, 2, 3, 4 };
            int k = 3; // K divisions
            int n = 4; // Size of Array
            solve(arr, n, k, 0, 0, 0);
            System.out.println(ans + "\n");
        }
}
```

# Binary Tree Questions

## Question 1 :

**Check if a Binary Tree is univalued or not**

We have a binary tree, the task is to check if the binary tree is univalued or not. If found to be true, then print "YES". Otherwise, print "NO".



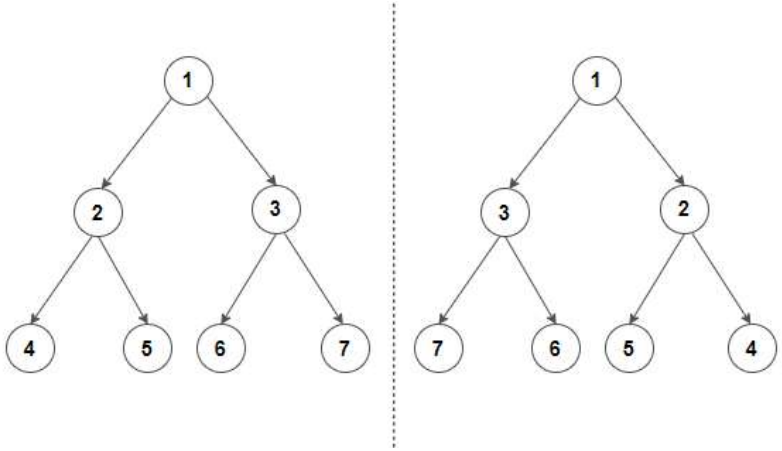

**Sample Input 1** :

**Sample Output 1** : false

## Question 2 :

**Invert Binary Tree**

Mirror of a Tree: Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.



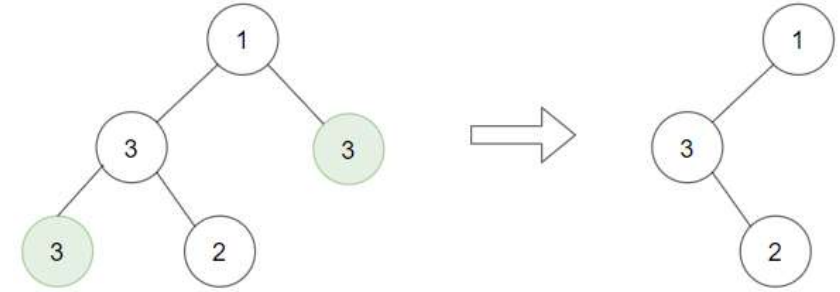

**Input1** : fig. above

**Output1** : Yes

## Question 3 :

**Delete leaf nodes with values as x**

We have a binary tree and a target integer x, delete all the leaf nodes having value as x. Also, delete the newly formed leaves with the target value as x.




:

## Question 4 :

**Find All Duplicate Subtrees**

We have a binary tree, find all duplicate subtrees. For each duplicate subtree, we only need to return the root node of any one of them. Two trees are duplicates if they have the same structure with the same node values.

**Input1:**

```
     1
    / \
   4   3
  /   / \
 3   4   3
    /
   3
```

**Output1**: 4-3 , 3

## Question 5 :

**Maximum Path Sum in a Binary Tree**

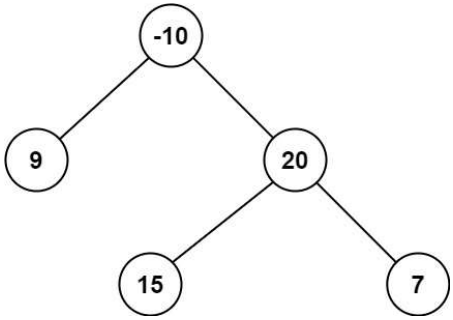
We have a binary tree, find the maximum path sum. The path may start and end at any node in the tree.

**Input1:**

```
  4
 / \
2   7
```

**Output1**: 13

**Input2**:




;

**Output2**: 42

# Binary Tree Solutions

## Solution 1:

Time Complexity : o(h)
Space Complexity: o(1)

```
import java.util.*;
class Solution{

static class Node{
        int data;
        Node left;
        Node right;
};

static Node newNode(int data){
        Node temp = new Node();
        temp.data = data;
        temp.left = temp.right = null;
        return (temp);
}

static boolean isUnivalTree(Node root){

        if (root == null){
                return true;
        }

        if (root.left != null
                && root.data != root.left.data)
                return false;

        if (root.right != null
                && root.data != root.right.data)
                return false;

        return isUnivalTree(root.left)
                && isUnivalTree(root.right);
```

```
}

public static void main(String[] args){

        Node root = newNode(1);
        root.left = newNode(1);
        root.right = newNode(1);
        root.left.left = newNode(1);
        root.left.right = newNode(1);
        root.right.right = newNode(1);

        if (isUnivalTree(root)) {
                System.out.print("YES");
        }
        else{
                System.out.print("NO");
        }
}
}
```



## Solution 2 :

Time Complexity : o(n)
Space Complexity: o(n)

```
class Node{
        int data;
        Node left, right;

        public Node(int item)
        {
                data = item;
                left = right = null;
        }
}

class Solution{
        Node root;

        void mirror(){
                root = mirror(root);
        }
```

```
Node mirror(Node node){
        if (node == null)
                return node;

        /* do the subtrees */
        Node left = mirror(node.left);
        Node right = mirror(node.right);

        /* swap the left and right pointers */
        node.left = right;
        node.right = left;

        return node;
}

void inOrder(){
        inOrder(root);
}

void inOrder(Node node){
        if (node == null)
                return;

        inOrder(node.left);
        System.out.print(node.data + " ");

        inOrder(node.right);
}

public static void main(String args[]){
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("Inorder traversal of input tree is :");
        tree.inOrder();
        System.out.println("");
        tree.mirror();
        System.out.println("Inorder traversal of binary tree is : ");
        tree.inOrder();

}
```

```
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Solution {

static class Node {
        int data;
        Node left, right;
}

static Node newNode(int data){
        Node newNode = new Node();
        newNode.data = data;
        newNode.left = null;
        newNode.right = null;
        return (newNode);
}

static Node deleteLeaves(Node root, int x){
        if (root == null)
                return null;
        root.left = deleteLeaves(root.left, x);
        root.right = deleteLeaves(root.right, x);

        if (root.data == x && root.left == null && root.right == null) {

                return null;
        }
        return root;
}

static void inorder(Node root){
        if (root == null)
                return;
        inorder(root.left);
```

```
        System.out.print(root.data + " ");
        inorder(root.right);
}

public static void main(String[] args){
        Node root = newNode(10);
        root.left = newNode(3);
        root.right = newNode(10);
        root.left.left = newNode(3);
        root.left.right = newNode(1);
        root.right.right = newNode(3);
        root.right.right.left = newNode(3);
        root.right.right.right = newNode(3);
        deleteLeaves(root, 3);
        System.out.print("Inorder traversal after deletion : ");
        inorder(root);
}
}
```



## Solution 4 :

Time Complexity : o(n*n)
Space Complexity: o(n*n)

```
import java.util.HashMap;
public class Solution {

        static HashMap<String, Integer> m;
        static class Node {
                int data;
                Node left;
                Node right;
                Node(int data){
                        this.data = data;
                        left = null;
                        right = null;
                }
        }
```

```
    static String inorder(Node node){
        if (node == null)
            return "";

        String str = "(";
        str += inorder(node.left);
        str += Integer.toString(node.data);
        str += inorder(node.right);
        str += ")";

        if (m.get(str) != null && m.get(str)==1 )
            System.out.print( node.data + " ");

        if (m.containsKey(str))
            m.put(str, m.get(str) + 1);
        else
            m.put(str, 1);

        return str;
    }

    static void printAllDups(Node root){
        m = new HashMap<>();
        inorder(root);
    }

    public static void main(String args[]){
        Node root = null;
        root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.right.left = new Node(2);
        root.right.left.left = new Node(4);
        root.right.right = new Node(4);
        printAllDups(root);
    }
}
```

Solution 5 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Node {

        int data;
        Node left, right;

        public Node(int item) {
                data = item;
                left = right = null;
        }
}

class Res {
        public int val;
}

class Solution {
        Node root;
        int findMaxUtil(Node node, Res res)  {

                if (node == null)
                        return 0;

                int l = findMaxUtil(node.left, res);
                int r = findMaxUtil(node.right, res);

                int max_single = Math.max(Math.max(l, r) + node.data,
                                                        node.data);

                int max_top = Math.max(max_single, l + r + node.data);

                res.val = Math.max(res.val, max_top);

                return max_single;
        }

        int findMaxSum() {
```

```
        return findMaxSum(root);
    }

    int findMaxSum(Node node) {

        Res res = new Res();
        res.val = Integer.MIN_VALUE;

        findMaxUtil(node, res);
        return res.val;
    }

    public static void main(String args[]) {
        Solution tree = new Solution();
        tree.root = new Node(10);
        tree.root.left = new Node(2);
        tree.root.right = new Node(10);
        tree.root.left.left = new Node(20);
        tree.root.left.right = new Node(1);
        tree.root.right.right = new Node(-25);
        tree.root.right.right.left = new Node(3);
        tree.root.right.right.right = new Node(4);
        System.out.println("maximum path sum is : " +
                                                tree.findMaxSum());
    }
}
```

# AVL Trees

**AVL** Tree is a self-balancing BST.

**Balance Factor** in AVL tree = height(left Subtree) - height(right Subtree) and always equal to {-1, 0, 1}

There are 4 cases of rotation in an AVL Tree:

a. LL Case
   - Right Rotate

b. RR Case
   - Left Rotate

c. LR Case
   - Left Rotate
   - Right Rotate

d. RL Case
   - Right Rotate
   - Left Rotate



## AVL Tree Code (Insertion)

```
public class AVLTrees {
    static class Node {
        int data, height;
        Node left, right;

        Node(int data) {
            this.data = data;
            height = 1;
        }
    }

    public static Node root;

    public static int height(Node root) {
        if (root == null)
            return 0;
```

```
        return root.height;
    }

    // Right rotate subtree rooted with y
    public static Node rightRotate(Node y) {
        Node x = y.left;
        Node T2 = x.right;

        // rotation using 3 nodes
        x.right = y;
        y.left = T2;

        // update heights
        y.height = Math.max(height(y.left), height(y.right)) + 1;
        x.height = Math.max(height(x.left), height(x.right)) + 1;

        // x is new root
        return x;
    }

    // Left rotate subtree rooted with x
    public static Node leftRotate(Node x) {
        Node y = x.right;
        Node T2 = y.left;

        // rotation using 3 nodes
        y.left = x;
        x.right = T2;

        // update heights
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        y.height = Math.max(height(y.left), height(y.right)) + 1;

        // y is new root
        return y;
    }

    // Get Balance factor of node
    public static int getBalance(Node root) {
        if (root == null)
```

```
            return 0;

        return height(root.left) - height(root.right);
    }

    public static Node insert(Node root, int key) {
        if (root == null)
            return new Node(key);

        if (key < root.data)
            root.left = insert(root.left, key);
        else if (key > root.data)
            root.right = insert(root.right, key);
        else
            return root; // Duplicate keys not allowed

        // Update root height
        root.height = 1 + Math.max(height(root.left), height(root.right));

        // Get root's balance factor
        int bf = getBalance(root);

        // Left Left Case
        if (bf > 1 && key < root.left.data)
            return rightRotate(root);

        // Right Right Case
        if (bf < -1 && key > root.right.data)
            return leftRotate(root);

        // Left Right Case
        if (bf > 1 && key > root.left.data) {
            root.left = leftRotate(root.left);
            return rightRotate(root);
        }

        // Right Left Case
        if (bf < -1 && key < root.right.data) {
            root.right = rightRotate(root.right);
            return leftRotate(root);
```

```
        }

        return root; //returned if AVL balanced
    }


    public static void preorder(Node root) {
        if(root == null) {
            return;
        }

        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }

    public static void main(String[] args) {
        root = insert(root, 10);
        root = insert(root, 20);
        root = insert(root, 30);
        root = insert(root, 40);
        root = insert(root, 50);
        root = insert(root, 25);

        /*          AVL Tree

                     30
                     / \
                   20 40
                   / \  \
                 10 25 50

        */

        preorder(root);
    }
}
```

## AVL Tree Deletion

*BST delete is a recursive function in which, after deletion, we get pointers to all ancestors one by one in a bottom up manner. So we don't need a parent pointer to travel up. The recursive code itself travels up and visits all the ancestors of the deleted node.*

a. *Perform the normal BST deletion.*
b. *The current node must be one of the ancestors of the deleted node. Update the height of the current node.*
c. *Get the balance factor (left subtree height – right subtree height) of the current node.*
d. *If the balance factor is greater than 1, then the current node is unbalanced and we are either in the Left Left case or Left Right case. To check whether it is Left Left case or Left Right case, get the balance factor of the left subtree. If the balance factor of the left subtree is greater than or equal to 0, then it is Left Left case, else Left Right case.*
e. *If the balance factor is less than -1, then the current node is unbalanced and we are either in the Right Right case or Right Left case. To check whether it is a Right Right case or Right Left case, get the balance factor of the right subtree. If the balance factor of the right subtree is smaller than or equal to 0, then it is Right Right case, else Right Left case.*

```
//for non-empty BST, return the node with MIN data
   public static Node getMinNode(Node root) {
       Node curr = root;
       //MIN data is at left-most node
       while (curr.left != null)
       curr = curr.left;
       return curr;
   }
   public static Node deleteNode(Node root, int key) {
       // perform usual BST delete
       if (root == null) {
           return root;
       }

       // key < root's data => then it lies in left subtree
       if (key < root.data) {
           root.left = deleteNode(root.left, key);
       }
       // key > root's data => then it lies in right subtree
       else if (key > root.data) {
           root.right = deleteNode(root.right, key);
       }
       // key = root's data => then this is the node to be deleted
       else {
            // node with only one child or no child
```

```
        if ((root.left == null) || (root.right == null)) {
            Node temp = null;
            if (temp == root.left)
                temp = root.right;
            else
                temp = root.left;
            // No child case
            if (temp == null)
            {
                temp = root;
                root = null;
            }
            else // One child case
                root = temp; // Copy the contents of
                             // the non-empty child
        }
        else {
            // node with two children: Get the inorder
            // successor (smallest in the right subtree)
            Node temp = getMinNode(root.right);
            // Copy the inorder successor's data to this node
            root.data = temp.data;
            // Delete the inorder successor
            root.right = deleteNode(root.right, temp.data);
        }
    }
    // If the tree had only one node then return
    if (root == null)
        return root;
    // update height of curr node
    root.height = Math.max(height(root.left), height(root.right)) + 1;
    // get balance factor of this node (to check for unbalanced)
    int bf = getBalance(root);
    // If this node becomes unbalanced, then there are 4 cases
    // Left Left Case
    if (bf > 1 && getBalance(root.left) >= 0)
        return rightRotate(root);
    // Left Right Case
    if (bf > 1 && getBalance(root.left) < 0)
    {
```

```
            root.left = leftRotate(root.left);
            return rightRotate(root);
        }
        // Right Right Case
        if (bf < -1 && getBalance(root.right) <= 0)
            return leftRotate(root);
        // Right Left Case
        if (bf < -1 && getBalance(root.right) > 0)
        {
            root.right = rightRotate(root.right);
            return leftRotate(root);
        }
        return root;
    }
```

# BST Questions

## Question 1 :

**Range Sum of BST**

We have a Binary Search Tree consisting of N nodes and two positive integers L and R, the task is to find the sum of values of all the nodes that lie in the range [L, R]..

**Sample Input1:**:

**Sample Output1** : 11

## Question 2 :

**Find the closest element in Binary Search Tree**

We have a binary search tree and a target node K**.** The task is to find the node with minimum absolute difference with given target value K**.**

**Sample Input 1** : 5
**Sample Output 1** : 5 (difference is 0)

**Sample Input 2** : .19
**Sample Output 2** : 20 (difference is 1)

## Question 3 :

**Find k-th smallest element in BST**

We have the root of a binary search tree and K as input, find Kth smallest element in BST.

**Sample Input 1** : k=3
**Sample Output 1** : 8

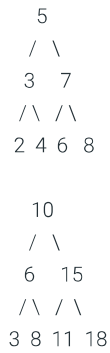
**Sample Input 2** : k=5
**Sample Output 2** : 5

## Question 4 :

**Two Sum BSTs**

Given a BST, transform it into a greater sum tree where each node contains sum of all nodes greater than that node.

**Sample Input 1** :



x = 16

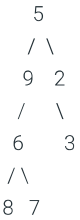
**Sample Output 1** : 3
The pairs are:
(5, 11), (6, 10) and (8, 8)

## Question 5 :

**Maximum Sum BST in Binary Tree**

We have a binary tree, the task is to print the maximum sum of nodes of a sub-tree which is also a Binary Search Tree.

**Sample Input 1** :



**Sample Output 1** : 8

# BST Solutions

## Solution 1:

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

class Solution{

    static class Node{
    int val;
    Node left, right;
    };

    static Node newNode(int item){
    Node temp = new Node();
    temp.val = item;
    temp.left = temp.right = null;
    return temp;
    }


    static int sum = 0;

    static int rangeSumBST(Node root, int low,
                                        int high){
        if (root == null)
        return 0;

        Queue<Node> q = new LinkedList<Node>();
        q.add(root);

        while (q.isEmpty() == false){

            Node curr = q.peek();
            q.remove();
            if (curr.val >= low &&
                    curr.val <= high){
                    sum += curr.val;
            }
```

```
                if (curr.left != null &&
                        curr.val > low) q.add(curr.left);
                if (curr.right != null &&
                        curr.val < high)
                        q.add(curr.right);
        }
        return sum;
}

static Node insert(Node node, int data){

        if (node == null)
                return newNode(data);
        if (data <= node.val)
                node.left = insert(node.left,
                                                data);
        else
                node.right = insert(node.right,
                                        data);
        return node;
}

public static void main(String[] args){

        Node root = null;
        root = insert(root, 10);
        insert(root, 5);
        insert(root, 15);
        insert(root, 3);
        insert(root, 7);
        insert(root, 18);

        int L = 7, R = 15;
        System.out.print(rangeSumBST(root, L, R));
}
}
```

Solution 2:

Time Complexity : o(h)
Space Complexity: o(1)

```
class Solution{
```

```
        static int min_diff, min_diff_key; static class Node{
        int key;
        Node left, right;
};

static Node newnode(int key){
        Node node = new Node();
        node.key = key;
        node.left = node.right = null;
        return (node);
}

         static void
  maxDiffUtil(Node ptr, int
      k){ if (ptr == null)
                return ;

        if (ptr.key == k){
                min_diff_key = k;
                return;
        }

        if (min_diff > Math.abs(ptr.key - k)){
                min_diff = Math.abs(ptr.key - k);
                min_diff_key = ptr.key;
        }

        if (k < ptr.key)
                maxDiffUtil(ptr.left, k);
        else
                maxDiffUtil(ptr.right, k);
}

static int maxDiff(Node root, int k){
        min_diff = 999999999;
        min_diff_key = -1;
        maxDiffUtil(root, k);
        return min_diff_key;
}
```

```
public static void main(String args[]){
        Node root = newnode(9); root.left = newnode(4);
        root.right = newnode(17);
        root.left.left = newnode(3);
        root.left.right = newnode(6);
        root.left.right.left = newnode(5);
        root.left.right.right = newnode(7);
        root.right.right = newnode(22);
        root.right.right.left = newnode(20);
        int k = 18;
        System.out.println( maxDiff(root, k));

}
}
```

## Solution 3 :

Time Complexity : o(n)
Space Complexity: o(h)

```
import java.io.*;

class Node {
        int data;
        Node left, right;
        Node(int x)
        {
                data = x;
                left = right = null;
        }
}

class Solution {

        static int count = 0;
        public static Node insert(Node
                root, int x) { if (root ==
                null)
                        return new Node(x);
```

```
        if (x < root.data)
            root.left = insert(root.left, x);
        else if (x > root.data) root.right = insert(root.right, x);
        return root;
    }

    public static Node kthSmallest(Node root, int k){

        if (root == null)
            return null;

        Node left = kthSmallest(root.left, k);
        if (left != null)
            return left;
        count++;
        if (count == k)
            return root;
        return kthSmallest(root.right, k);
    }

    public static void printKthSmallest(Node root, int k){
        Node res = kthSmallest(root, k);
        if (res == null)
          System.out.println("There are less than k nodes in the BST");
        else
            System.out.println("K-th Smallest Element is " + res.data);
    }

    public static void main(String[] args){
        Node root = null;
        int keys[] = { 20, 8, 22, 4, 12, 10, 14 };
        for (int x : keys)
            root = insert(root, x);
        int k = 3;
        printKthSmallest(root, k);
    }
}
```

Solution 4 :

Time Complexity : o(n1+n2)
Space Complexity: o(h1+h2)

```
import java.util.Stack;
public class Solution {

	static class Node {
		int data;
		Node left, right;

		public Node(int data) {
			this.data = data;
			left = null;
			right = null;
		}
	}

	static Node root1;
	static Node root2;
	static int countPairs(Node root1, Node root2,
	                                                    int x)
	{
		if (root1 == null || root2 == null)
			return 0;
		Stack<Node> st1 = new Stack<>();
		Stack<Node> st2 = new Stack<>();
		Node top1, top2;

		int count = 0;
		while (true) {
			while (root1 != null) {
				st1.push(root1);
				root1 = root1.left;
			}
			while (root2 != null) {
				st2.push(root2);
				root2 = root2.right;
```

```
                }
                if (st1.empty() || st2.empty()) break;

                top1 = st1.peek();
                top2 = st2.peek();
                if ((top1.data + top2.data) == x) {
                        count++;
                        st1.pop();
                        st2.pop();
                        root1 = top1.right;
                        root2 = top2.left;
                }

                else if ((top1.data + top2.data) < x) {
                        st1.pop();
                        root1 = top1.right;
                }

                else {
                        st2.pop();
                        root2 = top2.left;
                }
        }
        return count;
}

public static void main(String args[])
{
        root1 = new Node(5);
        root1.left = new Node(3);
        root1.right = new Node(7);
        root1.left.left = new Node(2);
        root1.left.right = new Node(4);
        root1.right.left = new Node(6);
        root1.right.right = new Node(8);

        root2 = new Node(10);
        root2.left = new Node(6);
        root2.right = new Node(15);
        root2.left.left = new Node(3);
```

```
        root2.left.right = new Node(8); root2.right.left = new Node(11);
        root2.right.right = new Node(18);

        int x = 16;
        System.out.println("Pairs = "
                + countPairs(root1, root2, x));
    }
}
```

## Solution 5 :

Time Complexity : o(n)
Space Complexity: o(n)

```
class Solution {

static class Node{
    Node left;
    Node right;
    int data;

    Node(int data){
        this.data = data;
        this.left = null;
        this.right = null;
    }
};

static class Info{
    int max;
    int min;
    boolean isBST;
    int sum;
    int currmax;

    Info(int m,int mi, boolean is,
        int su, int cur) {
```

```
        max = m;
                min = mi;
                isBST = is;
                sum = su;
                currmax = cur;
        }
        Info(){}
};

static class INT{
        int a;
}

static Info MaxSumBSTUtil( Node root, INT maxsum){
        if (root == null)
                return new Info( Integer.MIN_VALUE,
                                Integer.MAX_VALUE, true, 0, 0 );
        if (root.left == null && root.right == null){
                maxsum.a = Math.max(maxsum.a, root.data);
                return new Info( root.data, root.data,
                                true, root.data, maxsum.a );
        }

        Info L = MaxSumBSTUtil(root.left, maxsum);
        Info R = MaxSumBSTUtil(root.right, maxsum);

        Info BST=new Info();
        if (L.isBST && R.isBST && L.max < root.data &&
                                        R.min > root.data) {

                BST.max = Math.max(root.data, Math.max(L.max, R.max));
                BST.min = Math.min(root.data, Math.min(L.min, R.min));

                maxsum.a = Math.max(maxsum.a, R.sum +
                root.data + L.sum); BST.sum = R.sum + root.data +
                L.sum;
                BST.currmax = maxsum.a;

                BST.isBST = true;
                return BST;
```

```
        }
        BST.isBST = false;
        BST.currmax = maxsum.a;
        BST.sum = R.sum + root.data + L.sum;
                                                    return BST;
}


static int MaxSumBST( Node root){
        INT maxsum = new INT();
        maxsum.a = Integer.MIN_VALUE;
        return MaxSumBSTUtil(root,
maxsum).currmax; }

public static void main(String args[]){
        Node root = new Node(5);
        root.left = new Node(14);
        root.right = new Node(3);
        root.left.left = new Node(6);
        root.right.right = new Node(7);
        root.left.left.left = new Node(9);
        root.left.left.right = new Node(1);

        System.out.println( MaxSumBST(root));
}
}
```

# Red Black Trees

A **Red-Black** Tree is a self-balancing binary search tree in which each node has an extra bit, which represents its color (red or black).

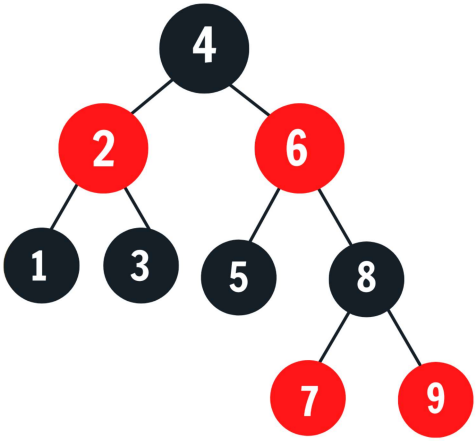
(Every Red Black Tree is a binary search tree but all the Binary Search Trees need not to be Red Black trees.)

There are few properties associated with Red-Black Trees:

- The root is black. (Root Property)
- Every external node is black. (External Property)
- The children of a red node are black. (Red Property)
- All external nodes have the same black depth.(Depth Property)
- Every New node must be inserted with Red color.
- Every leaf (i.e. NULL node) must be colored Black.

**Example**

The following is a Red Black Tree which is created by inserting numbers from 1 to 9 and every node is satisfying all the properties of the Red Black Tree.



Most of the Binary Search Tree operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the Binary Search Tree. The cost of these operations may become O(n) for a skewed Binary tree. If we make sure that height of the tree remains O(Logn) after every insertion and deletion, then we can guarantee an upper bound of O(Logn)for all these operations. The height of a Red-Black tree is always O(Logn) where n is the number of nodes in the tree.

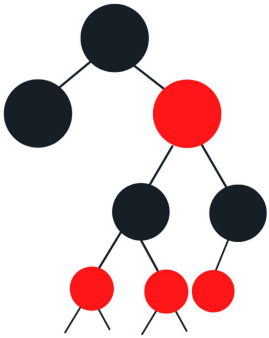
**Height of a Red-Black Tree**

Proving the height of the red-black tree is especially important because the height of the red-black tree is what allows us to calculate it asymptotic complexity and performance. This is one method of doing so.

First imagine a red-black tree with height. Now, we merge all red nodes into their black parents. A given black node can either have:

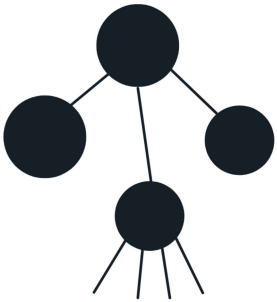
1. 2 black children, in which case the black parent still has 2 children.
2. 1 black child and 1 red child, in which case the black parent now has 3 children.
3. 2 red children, in which case the black parent now has 4 children.

Here is a graphical example of that merging process (assume any stray arrow points to a black node).



We merge all the red nodes into their parents nodes.

If rc = number of red children a red node has then each black node will now have 2*rc pointers coming out of it.



As you can see, every black node has either 2, 3, or 4 children.

This new tree has a height, h1. Because any given path in the original red-black tree had at most half its nodes red, we know that this new height is at least half the original height. So,

**H1 >= H/2**

The number of leaves in a tree is exactly equal to n+1, so

**n+1 > 2^(H1)**

**log(N+1) >= H1 >= H/2**

**H < 2log(n+1)**

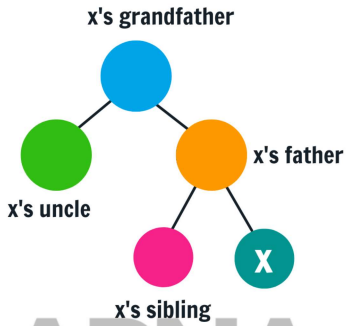
**Insertion in Red Black Trees**

In the Red-Black tree, we use two steps to do the balancing:

1. Recoloring
2. Rotation

Recolouring is the change in color of the node i.e. if it is red then change it to black and vice versa. It must be noted that the color of the NULL node is always black. Moreover, we always try recolouring first, if recolouring doesn't work, then we go for rotation. Following is a detailed algorithm. The

algorithms have mainly two cases depending upon the color of the uncle. If the uncle is red, we do recolour. If the uncle is black, we do rotations and/or recolouring.






**Process**:

Let x be the newly inserted node.

Perform standard BST insertion and make the color of newly inserted nodes as RED.

If x is the root, change the color of x as BLACK (Black height of complete tree increases by 1).

Do the following if the color of x's parent is not BLACK and x is not the root.

a) If x's uncle is RED (Grandparent must have been black from property 4)

(i) Change the color of parent and uncle as BLACK.

(ii) Color of a grandparent as RED.

(iii) Change x = x's grandparent, repeat steps 2 and 3 for new x.

b) If x's uncle is BLACK, then there can be four configurations for x, x's parent (p) and x's grandparent (g) (This is similar to AVL Tree)

(i) Left Left Case (p is left child of g and x is left child of p)

(ii) Left Right Case (p is left child of g and x is the right child of p)
(iii) Right Right Case (Mirror of case i)
(iv) Right Left Case (Mirror of case ii)
Recoloring after rotations:
For Left Left Case [3.b (i)] and Right Right case [3.b (iii)], swap colors of grandparent and parent after rotations
For Left Right Case [3.b (ii)]and Right Left Case [3.b (iv)], swap colors of grandparent and inserted node after rotations

**Code**

```
import java.io.*;

public class RedBlackTree {
   public Node root;//root node
   public RedBlackTree() {
       super();
       root = null;
   }

   class Node {
       int data;
       Node left;
       Node right;
       char colour;
       Node parent;

       Node(int data) {
           super();
           this.data = data; // only including data. not key
           this.left = null; // left subtree
           this.right = null; // right subtree
           this.colour = 'R'; // colour . either 'R' or 'B'
           this.parent = null; // required at time of rechecking.
       }
   }
   // this function performs left rotation
```

```
  Node rotateLeft(Node node) {
      Node x = node.right;
      Node y = x.left;
      x.left = node;
      node.right = y;
      node.parent = x; // parent resetting is also important.
      if(y!=null)
          y.parent = node;
      return(x);
  }
  //this function performs right rotation
  Node rotateRight(Node node) {
      Node x = node.left;
      Node y = x.right;
      x.right = node;
      node.left = y;
      node.parent = x;
      if(y!=null)
          y.parent = node;
      return(x);
  }



  // these are some flags.
  // Respective rotations are performed during traceback.
  // rotations are done if flags are true.
  boolean ll = false;
  boolean rr = false;
  boolean lr = false;
  boolean rl = false;
  // helper function for insertion. Actually this function performs all tasks in
single pass only.
  Node insertHelp(Node root, int data) {
      // f is true when RED RED conflict is there.
      boolean f=false;


      //recursive calls to insert at proper position according to BST properties.
      if(root==null)
          return(new Node(data));
      else if(data<root.data) {
```

```
            root.left = insertHelp(root.left, data);
            root.left.parent = root;
            if(root!=this.root) {
                if(root.colour=='R' && root.left.colour=='R')
                    f = true;
            }
        }
        else {
            root.right = insertHelp(root.right,data);
            root.right.parent = root;
            if(root!=this.root) {
                if(root.colour=='R' && root.right.colour=='R')
                    f = true;
            }
        // at the same time of insertion, we are also assigning parent nodes
        // also we are checking for RED RED conflicts
        }

        // now lets rotate.
        if(this.ll) {
            root = rotateLeft(root);
            root.colour = 'B';
            root.left.colour = 'R';
            this.ll = false;
        }
        else if(this.rr) {
            root = rotateRight(root);
            root.colour = 'B';
            root.right.colour = 'R';
            this.rr = false;
        }
        else if(this.rl) {
            root.right = rotateRight(root.right);
            root.right.parent = root;
            root = rotateLeft(root);
            root.colour = 'B';
            root.left.colour = 'R';

            this.rl = false;
        }
```

```
        else if(this.lr) {
            root.left = rotateLeft(root.left);
            root.left.parent = root;
            root = rotateRight(root);
            root.colour = 'B';
            root.right.colour = 'R';
            this.lr = false;
        }
        // when rotation and recolouring is done flags are reset.
        // Now lets take care of RED RED conflict
        if(f) {
            // to check which child is the current node of its parent
            if(root.parent.right == root) {
                // case when parent's sibling is black
                if(root.parent.left==null || root.parent.left.colour=='B') {//
perform certaing rotation and recolouring. This will be done while backtracking.
Hence setting up respective flags.
                    if(root.left!=null && root.left.colour=='R')
                        this.rl = true;
                    else if(root.right!=null && root.right.colour=='R')
                        this.ll = true;
                }
                // case when parent's sibling is red
                else {
                    root.parent.left.colour = 'B';
                    root.colour = 'B';
                    if(root.parent!=this.root)
                        root.parent.colour = 'R';
                }
            }
            else {
                if(root.parent.right==null || root.parent.right.colour=='B') {
                    if(root.left!=null && root.left.colour=='R')
                        this.rr = true;
                    else if(root.right!=null && root.right.colour=='R')
                        this.lr = true;
                }
                else {
                    root.parent.right.colour = 'B';
                    root.colour = 'B';
```

```
                if(root.parent!=this.root)
                    root.parent.colour = 'R';
            }
        }
        f = false;
    }
    return(root);
}

// function to insert data into tree.
public void insert(int data) {
    if(this.root==null) {
        this.root = new Node(data);
        this.root.colour = 'B';
    }
    else
        this.root = insertHelp(this.root,data);
}
// helper function to print inorder traversal
void inorderTraversalHelper(Node node) {
    if(node!=null) {
        inorderTraversalHelper(node.left);
        System.out.printf("%d ", node.data);
        inorderTraversalHelper(node.right);
    }
}
//function to print inorder traversal
public void inorderTraversal() {
    inorderTraversalHelper(this.root);
}
// helper function to print the tree.
void printTreeHelper(Node root, int space) {
    int i;
    if(root != null) {
        space = space + 10;
        printTreeHelper(root.right, space);
        System.out.printf("\n");
        for ( i = 10; i < space; i++) {
            System.out.printf(" ");
        }
```

```
            System.out.printf("%d", root.data);
            System.out.printf("\n");
            printTreeHelper(root.left, space);
        }
    }
    // function to print the tree.
    public void printTree() {
        printTreeHelper(this.root, 0);
    }
    public static void main(String[] args) {
        RedBlackTree t = new RedBlackTree();
        int[] arr = {1, 4, 6, 3, 5, 7, 8, 2, 9};

        for(int i=0; i<9; i++) {
            t.insert(arr[i]);
            System.out.println();
            t.inorderTraversal();
        }

        t.printTree();
    }
}
```

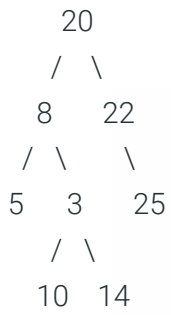
# HASHING QUESTIONS

## Question 1 :

**Bottom View of a Binary Tree**

The top view of a binary tree is the set of nodes visible when the tree is viewed from the top. Given a binary tree, print the top view of it. The output nodes can be printed in any order.

**Sample Input** :




**Sample Output** : 5 10 3 14 25

**Hint** : Use the concept of Vertical Order

## Question 2 :

**Two Sum**

Given an array of integers arr[ ]and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

**Sample Input 1** : arr = [2, 7, 11, 15], target = 9

**Sample Output 1** : [0, 1]

As arr[0] + arr[1] == 9, we return [0, 1].

**Sample Input 2** : arr = [3,2,4], target = 6

**Sample Output 2** : [1, 2]

## Question 3 :

**Sort by Frequency**

Given a string s, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string.

Return the sorted string. If there are multiple answers, return any of them.

**Sample Input 1** : s = "cccaaa"

**Sample Output 1** : "aaaccc"

Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.

Note that "cacaca" is incorrect, as the same characters must be together.

**Sample Input 2** : s = "tree"

**Sample Output 2** :"eert"

'e' appears twice while 'r' and 't' both appear once.

So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

**BONUS (LRU Cache)** **IMPORTANT**

Please go on the platform and solve this question : https://leetcode.com/problems/lru-cache/

# HASHING SOLUTIONS

## Solution 1:

Using the concept of Horizontal Distance (HD), as discussed in Top View of a Binary Tree Question in the Trees chapter.

```
import java.util.*;

class Solution {
   static class Node {
       int data;
       int hd;
       Node left, right;

       public Node(int key) {
           this.data = key;
           this.hd = Integer.MAX_VALUE;
           this.left = this.right = null;
       }
   }

       public static void bottomViewHelper(Node root, int curr, int hd,
TreeMap<Integer, int[]> m) {
       if (root == null)
           return;

       // If node for a particular HD is not present, add to the map.
       if (!m.containsKey(hd)) {
           m.put(hd, new int[]{ root.data, curr });
       }

       // Compare height for already
       // present node at similar horizontal
       // distance
       else {
           int[] p = m.get(hd);
           if (p[1] <= curr) {
               p[1] = curr;
```

```
                p[0] = root.data;
            }
            m.put(hd, p);
        }

        // call for left subtree
        bottomViewHelper(root.left, curr + 1, hd - 1, m);

        // call for right subtree
        bottomViewHelper(root.right, curr + 1, hd + 1, m);
    }

    public static void printBottomView(Node root) {
        // Map to store Horizontal Distance, Height and Data.
        TreeMap<Integer, int[]> m = new TreeMap<>();

        bottomViewHelper(root, 0, 0, m);

        // Prints the values stored by printBottomViewUtil()
        for(int val[] : m.values())
        {
            System.out.print(val[0] + " ");
        }
    }

    public static void main(String[] args) {
        Node root = new Node(20);
        root.left = new Node(8);
        root.right = new Node(22);
        root.left.left = new Node(5);
        root.left.right = new Node(3);
        root.right.left = new Node(4);
        root.right.right = new Node(25);
        root.left.right.left = new Node(10);
        root.left.right.right = new Node(14);

        System.out.println(
            "Bottom view of the given binary tree:");

        printBottomView(root);
```

```
    }
}
```

Solution 2:

```
public int[] twoSum(int[] arr, int target) {
        Map<Integer, Integer> visited = new HashMap<>();
        for(int i = 0; i<arr.length; i++) {
            //diff = given target - number given at ith index
            int diff = target - arr[i];

            // check if found difference is present in the MAP list
            if(visited.containsKey(diff)) {
                //if difference in map matches with the ith index element in array
                return new int[] { i, visited.get(diff) };
            }
            //add arr element in map to match with future element if forms a pair
            visited.put(arr[i],i);
        }
        //if no matches are found
        return new int[] {0, 0};
    }
```



Solution 3 :

```
public String frequencySort(String s) {
        HashMap<Character , Integer>map = new HashMap<>();
        for(int i=0;i<s.length();++i)
            map.put(s.charAt(i),map.getOrDefault(s.charAt(i),0)+1);

      PriorityQueue<Map.Entry<Character,Integer>> pq =
          new PriorityQueue<>((a,b)->a.getValue()== b.getValue()?
            a.getKey()-b.getKey(): b.getValue()-a.getValue());

        for(Map.Entry<Character,Integer>e:map.entrySet())  pq.add(e);
```

```
        StringBuilder res = new StringBuilder();
        while(pq.size()!=0){
            char ch = pq.poll().getKey();
            int val = map.get(ch);
           while(val!=0){
               res.append(ch);
               val--;
           }
        }
        return res.toString();
    }
```

# DSA by Shradha Didi & Aman Bhaiya

Meet us on Youtube (Apna College)

How to solve this sheet?

| | | |
|---|---|---|
| Easy | Ideal Time : 5-10 mins | |
| Medium | Ideal Time : 15-20 mins | 5 Questions each Day |
| Hard | Ideal Time : 40-60 mins (based on Qs) \| 88 Qs | |

| Topics | Question (375) | Companies | Remarks |
|---|---|---|---|
| Arrays | Maximum and Minimum Element in an Array | ABCO Accolite Amazon Cisco Hike Microsoft Snapdeal VMWare Google Adobe | |
| Arrays | Reverse the Array | Infosys Moonfrog Labs | |
| Arrays | Maximum-Subarray | Microsoft + Facebook Interview Qs | use Kadane's Algorithm |
| Arrays | Contains Duplicate | Amazon Interview Qs | |
| Arrays | Chocolate Distribution Problem | Amazon Interview Qs | |
| Arrays | Search in Rotated Sorted Array | Microsoft Google Adobe Amazon D-E-Shaw Flipkart Hike Intuit MakeMyTrip Pay | |
| Arrays | Next Permutation | Uber + Goldman Sachs + Adobe Interview Qs | |
| Arrays | Best time to Buy and Sell Stock | Amazon D-E-Shaw Directi Flipkart Goldman Sachs Intuit MakeMyTrip Microsoft ( | |
| Arrays | Repeat and Missing Number Array | Amazon Interview Qs | |
| Arrays | Kth-Largest Element in an Array | Amazon Microsoft Walmart Adobe | |
| Arrays | Trapping Rain Water | Samsung Interview Qs | |
| Arrays | Product of Array Except Self | Microsoft + Facebook Interview Qs | |
| Arrays | Maximum Product Subarray | Amazon D-E-Shaw Microsoft Morgan Stanley OYO Rooms Google | |
| Arrays | Find Minimum in Rotated Sorted Array | Adobe Amazon Microsoft Morgan Stanley Samsung Snapdeal Times Internet | |
| Arrays | Find Pair with Sum in Sorted & Rotated Array | Microsoft + Google + Apple Interview Qs | |
| Arrays | 3Sum | Adobe Amazon Microsoft Morgan Stanley Samsung Snapdeal Times Internet | |
| Arrays | Container With Most Water | Flipkart + Dunzo Interview Qs | |
| Arrays | Given Sum Pair | Infosys + Amazon + Flipkart Interview Qs | |
| Arrays | Kth - Smallest Element | ABCO Accolite Amazon Cisco Hike Microsoft Snapdeal VMWare Google Adobe | |
| Arrays | Merge Overlapping Intervals | Google Interview Qs | |
| Arrays | Find Minimum Number of Merge Operations to Make an Array Palindrome | Amazon | |
| Arrays | Given an Array of Numbers Arrange the Numbers to Form the Biggest Number | Barclays Interview Qs | |
| Arrays | Space Optimization Using Bit Manipulations | Amazon | |
| Arrays | Subarray Sum Divisible K | Snapdeal Microsoft | |
| Arrays | Print all Possible Combinations of r Elements in a Given Array of Size n | Amazon | |
| Arrays | Mo's Algorithm | Microsoft | |
| | | | |
| Strings | Valid Palindrome | Amazon Cisco D-E-Shaw Facebook FactSet Morgan Stanley Paytm Zoho | |
| Strings | Valid Anagram | Nagarro Media.net Directi Google Adobe Flipkart | |
| Strings | Valid parentheses | Google Interview Qs | use Stacks (if possible) |
| Strings | Remove Consecutive Characters | Samsung + Adobe | |
| Strings | Longest Common Prefix | Adobe + Grofers + Dunzo Interview Qs | |
| Strings | Convert a Sentence into its Equivalent Mobile Numeric Keypad Sequence | Adobe | |
| Strings | Print all the Duplicates in the Input String | Ola + Amdocs IQ | |
| Strings | Longest Substring without Repeating Characters | Morgan Stanley + Amazon IQ | |
| Strings | Longest Repeating Character Replacement | Amazon Google | |
| Strings | Group Anagrams | Samsung + Adobe + Amazon Interview Qs | |
| Strings | Longest Palindromic Substring | Microsoft + Google + Samsung + Visa IQ | |
| Strings | Palindromic Substrings | Microsoft IQ | |
| Strings | Next Permutation | Adobe + Goldman Sachs + Uber | |
| Strings | Count Palindromic Subsequences | Myntra Interview Qs | |
| Strings | Smallest Window in a String Containing all the Characters of Another String | Microsoft + Amazon IQ | |
| Strings | Wildcard String Matching | Microsoft + Amazon + Ola IQ | |
| Strings | Longest Prefix Suffix | Flipkart + Swiggy IQ | |
| Strings | Rabin-Karp Algorithm for Pattern Searching | Microsoft | |
| Strings | Transform One String to Another using Minimum Number of Given Operation | Directi | |
| Strings | Minimum Window Substring | Amazon Google MakeMyTrip Streamoid Technologies Microsoft Media.net Atla | |
| Strings | Boyer Moore Algorithm for Pattern Searching | Amdocs | |
| Strings | Word Wrap | Microsoft | use Dynaming Programming |
| | | | |
| 2D Arrays | Zigzag (or diagonal) Traversal of Matrix | Amazon | |
| 2D Arrays | Set Matrix Zeroes | Amazon Microsoft | |
| 2D Arrays | Spiral Matrix | Flipkart + Apple + Societe Generale IQ | |
| 2D Arrays | Rotate Image | Microsoft Paytm Samsung Adobe | |
| 2D Arrays | Word Search | Google + Ola + Goldman Sachs IQ | |
| 2D Arrays | Find the Number of Islands \| Set 1 (Using DFS) | Microsoft + Uber + Apple + Amazon IQ | Read about DFS |
| 2D Arrays | Given a Matrix of 'O' and 'X', Replace 'O' with 'X' if Surrounded by 'X' | Google | |
| 2D Arrays | Find a Common Element in all Rows of a Given Row-Wise Sorted Matrix | MAQ Software Microsoft VMWare | |
| 2D Arrays | Create a Matrix with Alternating Rectangles of O and X | MAQ VMWare | |
| 2D Arrays | Maximum Size Rectangle of all 1s | Amazon Microsoft | |
| | | | |
| Searching & Sorting | Permute Two Arrays such that Sum of Every Pair is Greater or Equal to K | Samsung | |
| Searching & Sorting | counting sort | Samsung+ Morgan Stanley+ Snapdeal + EPAM Systems | |
| Searching & Sorting | find common elements three sorted arrays | MAQ Software Microsoft VMWare | |
| Searching & Sorting | Searching in an array where adjacent differ by at most k | TCS Amazon | |
| Searching & Sorting | ceiling in a sorted array | TCS | |
| Searching & Sorting | Piar with given difference | Amazon Visa | |
| Searching & Sorting | majority element | Amazon+ Google | |
| Searching & Sorting | count triplets with sum smaller that a given value | Amazon SAP Labs | |
| Searching & Sorting | Maximum Sum Subsequence with no adjacent elements | Amazon FactSet Oxigen Wallet OYO Rooms Paytm Walmart Yahoo Adobe Flipka | |
| Searching & Sorting | Merge Sorted Arrays using O(1) Space | Amdocs Brocade Goldman Sachs Juniper Networks Linkedin Microsoft Quikr Sr | |
| Searching & Sorting | Inversion of Array | Adobe Amazon BankBazaar Flipkart Microsoft Myntra MakeMyTrip | |
| Searching & Sorting | Find Duplicates in O(n) Time and O(1) Extra Space | Amazon D-E-Shaw Flipkart Paytm Qualcomm Zoho | |
| Searching & Sorting | Radix Sort | Amazon+ Microsoft | |
| Searching & Sorting | Product of Array except itself | Accolite Amazon D-E-Shaw Intuit Morgan Stanley Opera Microsoft Flipkart | |
| Searching & Sorting | Make all Array Elements Equal | Amazon | |
| Searching & Sorting | Check if Reversing a Sub Array Make the Array Sorted | Amazon | |
| Searching & Sorting | Find Four Elements that Sum to a Given Value | Adobe Amazon Google Microsoft OYO Rooms | |
| Searching & Sorting | Median of Two Sorted Array with Different Size | Amazon Samsung Microsoft Google | |
| Searching & Sorting | Median of Stream of Integers Running Integers | Amazon + Google | |
| Searching & Sorting | Print Subarrays with 0 Sum | Paytm Adobe | |
| Searching & Sorting | Aggressive Cows | Adobe | |
| Searching & Sorting | Allocate Minimum number of Pages | Google Infosys Codenation Amazon Microsoft | |
| Searching & Sorting | Minimum Swaps to Sort | Amazon + Google | |

| | | | |
|---|---|---|---|
| **Backtracking** | Backtracking Set 2 Rat in a Maze | Microsoft Amazon | |
| **Backtracking** | Combinational Sum | Adobe Amazon Microsoft | |
| **Backtracking** | Crossword-Puzzle | Microsoft | |
| **Backtracking** | Longest Possible Route in a Matrix with Hurdles | Microsoft | |
| **Backtracking** | Printing all solutions in N-Queen Problem | Accolite Amazon Amdocs D-E-Shaw MAQ Software Twitter Visa Microsoft | |
| **Backtracking** | Solve the Sudoku | Amazon Directi Flipkart MakeMyTrip MAQ Software Microsoft Ola Cabs Oracle P | |
| **Backtracking** | Partition Equal Subset Sum | Amazon + Adobe + Accolite + Traveloka | |
| **Backtracking** | M Coloring Problem | Amazon | |
| **Backtracking** | Knight Tour | IBM | |
| **Backtracking** | Soduko | Amazon + Adobe + Accolite + Traveloka | |
| **Backtracking** | Remove Invalid Parentheses | Uber | |
| **Backtracking** | Word Break Problem using Backtracking | | |
| **Backtracking** | Print all Palindromic Partitions of a String | Facebook Amazon Microsoft | |
| **Backtracking** | Find Shortest Safe Route in a Path with Landmines | Facebook Amazon Microsoft | |
| **Backtracking** | Partition of Set into K Subsets with Equal Sum | Amazon | |
| **Backtracking** | Backtracking set-7 hamiltonian cycle | Amazon | |
| **Backtracking** | tug-of-war | Google | |
| **Backtracking** | Maximum Possible Number by doing at most K swaps | Amazon + Adobe + Accolite + Traveloka | |
| **Backtracking** | Backtracking set-8 solving cryptarithmetic puzzles | Goldman Sachs | |
| **Backtracking** | Find paths from corner cell to middle cell in maze | Meta | |
| **Backtracking** | Arithmetic Expressions | Flipkart | |
| | | | |
| | | | |
| **Linked List** | Reverse Linked List | Sprinklr | |
| **Linked List** | Linked List Cycle | Accolite Amazon D-E-Shaw Hike Lybrate Mahindra Comviva MakeMyTrip MAQ S | |
| **Linked List** | Merge Two Sorted Lists | Accolite Amazon Belzabar Brocade FactSet Flipkart MakeMyTrip Microsoft OATS | |
| **Linked List** | Delete without Head node | Amazon Goldman Sachs Kritikal Solutions Microsoft Samsung Visa | |
| **Linked List** | Remove duplicates from an unsorted linked list | Amazon Intuit | |
| **Linked List** | Sort a linked list of 0s-1s-or-2s | Microsoft Amazon MakeMyTrip | |
| **Linked List** | Multiply two numbers represented linked lists | Amazon | |
| **Linked List** | Remove nth node from end of list | Accolite Adobe Amazon Citicorp Epic Systems FactSet Hike MAQ Software Mono | |
| **Linked List** | Reorder List | Amazon Microsoft OYO Rooms Intuit | |
| **Linked List** | Detect and remove loop in a linked list | Accolite Amazon D-E-Shaw Hike Lybrate Mahindra Comviva MakeMyTrip MAQ S | |
| **Linked List** | Write a Function to get the Intersection Point of two Linked Lists | Amazon | |
| **Linked List** | Flatten a linked list with next and child pointers | Google | |
| **Linked List** | Linked list in zig-zag fashion | Micorsoft | |
| **Linked List** | Reverse a doubly linked list | Walmart | |
| **Linked List** | Delete nodes which have a greater value on right side | Amazon | |
| **Linked List** | Segregate even and odd Elements in a Linked List | Walmart | |
| **Linked List** | Point to next higher value node in a linked list with an Arbitrary Pointer | GeekyAnts | |
| **Linked List** | Rearrange a given linked list in place | Ola Uber | |
| **Linked List** | Sort Biotonic Doubly Linked Lists | Morgan Stanley | |
| **Linked List** | Merge K Sorted Lists | Microsoft+ Ola+ eBay | |
| **Linked List** | Merge sort for linked list | Accolite Adobe Amazon MAQ Software Microsoft Paytm Veritas | Important |
| **Linked List** | Quicksort on singly-linked list | Paytm | Important |
| **Linked List** | Sum of two linked lists | Accolite Amazon Flipkart MakeMyTrip Microsoft Morgan Stanley Qualcomm Sna | |
| **Linked List** | Flattening a linked list | 24*7 Innovation Labs Amazon Drishti-Soft Flipkart Goldman Sachs Microsoft Pa | |
| **Linked List** | Clone a linked list with next and random Pointer | Triology | |
| **Linked List** | Subtract two numbers represented as linked lists | Amazon Goldman Sachs | |
| | | | |
| | | | |
| **Stacks & Queues** | Implement two stacks in an Array | 24*7 Innovation Labs Microsoft Samsung Snapdeal | |
| **Stacks & Queues** | Evaluation of Postfix Expression | Amazon + Google + Facebook | |
| **Stacks & Queues** | Implement Stack using Queues | Facebook | |
| **Stacks & Queues** | Queue Reversal | Amazon + Morgain Stanley | |
| **Stacks & Queues** | Implement Stack Queue using Deque | Microsoft +Atlassian | |
| **Stacks & Queues** | Reverse first k elements of queue | Microsoft + Amdocs | |
| **Stacks & Queues** | Design Stack with Middle Operation | MaQ Software | |
| **Stacks & Queues** | Infix to Postfix | Amazon + Samsung + Paytm + Vmware inc | |
| **Stacks & Queues** | Design and Implement Special stack | Amazon Google Microsoft Visa Goldman Sachs | |
| **Stacks & Queues** | Longest Valid String | Google Microsoft | |
| **Stacks & Queues** | Find if an expression has duplicate parenthesis or not | Flipkart Oracle OYO Rooms Snapdeal Walmart Yatra.com Microsoft Google | |
| **Stacks & Queues** | Stack permutations check if an array is stack permutation of other | Visa | |
| **Stacks & Queues** | Count natural numbers whose permutation greater number | Amazon | |
| **Stacks & Queues** | Sort a stack using Recursion | Amazon Goldman Sachs IBM Intuit Kuliza Yahoo Microsoft | |
| **Stacks & Queues** | Queue based approach for first non repeating character in a stream | Microsoft Flipkart | |
| **Stacks & Queues** | The Celebrity Problem | Google + Visa + Apple | |
| **Stacks & Queues** | Next larger Element | Visa | |
| **Stacks & Queues** | Distance of nearest cell | Flipkar + Facebook | |
| **Stacks & Queues** | Rotten-oranges | Facebook | |
| **Stacks & Queues** | Next smaller element | Codenation | |
| **Stacks & Queues** | Circular-tour | Codenation Flipkart | |
| **Stacks & Queues** | Efficiently implement k-stacks single array | Flipkart | |
| **Stacks & Queues** | The celebrity problem | Google + Visa + Apple | |
| **Stacks & Queues** | Iterative tower of hanoi | Microsoft Flipkart | |
| **Stacks & Queues** | Find the maximum of minimums for every window size in a given array | Amazon Microsoft Flipkart | |
| **Stacks & Queues** | lru cache implementation | Microsoft + Uber + Alibaba | |
| **Stacks & Queues** | Find a tour that visits all stations | Uber | |
| | | | |
| | | | |
| **Greedy** | Activity selection problem greedy algo | Facebook Morgan Stanley Flipkart | |
| **Greedy** | Greedy algorithm to find minimum number of coins | Accolite Amazon Morgan Stanley Oracle Paytm Samsung Snapdeal Synopsys Vi | |
| **Greedy** | Minimum sum two numbers formed digits array-2 | Google | |
| **Greedy** | Minimum sum absolute difference pairs two arrays | Amazon | |
| **Greedy** | Find maximum height pyramid from the given array of objects | Flipkart Amazon | |
| **Greedy** | Minimum cost for acquiring all coins with k extra coins allowed with every coin | | |
| **Greedy** | Find maximum equal sum of every three stacks | Microsoft Amazon Flipkart | |
| **Greedy** | Job sequencing problem | Microsoft + Acolite | |
| **Greedy** | Greedy algorithm egyptian fraction | | |
| **Greedy** | Fractional knapsack problem | Microsoft | |
| **Greedy** | Maximum length chain of pairs | Amazon Microsoft | |
| **Greedy** | Find smallest number with given number of digits and digit sum | MAQ Software OYO Rooms | |
| **Greedy** | Maximize sum of consecutive differences circular-array | Maccafe | |
| **Greedy** | paper-cut minimum number squares | Google | |
| **Greedy** | Lexicographically smallest array-k consecutive swaps | Amazon | |
| **Greedy** | Problems-CHOCOLA | Flipkart | |
| **Greedy** | Find minimum time to finish all jobs with given constraints | | |
| **Greedy** | Job sequencing using disjoint set union | Samsung | |
| **Greedy** | Rearrange characters string such that no two adjacent are same | Amazon Microsoft | |

| | | | |
|---|---|---|---|
| Greedy | Minimum edges to reverse to make path from a source to a destination | | |
| Greedy | Minimize Cash Flow among a given set of friends who have borrowed money from each other | | |
| Greedy | Minimum Cost to cut a board into squares | Maccafe | |
| | | | |
| | | | |
| Binary Trees | Maximum Depth of Binary Tree | Amazon Cadence India CouponDunia D-E-Shaw FactSet FreeCharge MakeMyTri | |
| Binary Trees | Reverse Level Order Traversal | Amazon + Microsoft + flipkart + Adobe | |
| Binary Trees | Subtree of Another Tree | Amazon + Microsoft + Facebook | |
| Binary Trees | Invert Binary Tree | Amazon Hike | |
| Binary Trees | Binary Tree Level Order Traversal | Accolite Adobe Amazon Cisco D-E-Shaw Flipkart | |
| Binary Trees | Left View of Binary Tree | Microsoft + Adobe + Cisco Networking Academy | |
| Binary Trees | Right View of Binary Tree | Amdocs | |
| Binary Trees | ZigZag Tree Traversal | Amazon Cisco FactSet Hike Snapdeal Walmart Microsoft Flipkart | |
| Binary Trees | Create a mirror tree from the given binary tree | Accolite Adobe Amazon Belzabar EBay Goldman Sachs Microsoft Morgan Stanl | |
| Binary Trees | Leaf at same level | Amazon | |
| Binary Trees | Check for Balanced Tree | Amazon Walmart Microsoft | |
| Binary Trees | Transform to Sum Tree | Amazon FactSet Microsoft Samsung Walmart | |
| Binary Trees | Check if Tree is Isomorphic | Amazon Microsoft | |
| Binary Trees | Same Tree | Amazon Microsoft Flipkart | |
| Binary Trees | Construct Binary Tree from Preorder and Inorder Traversal | Accolite Amazon Microsoft | |
| Binary Trees | Height of Binary Tree | Amazon Cadence India CouponDunia D-E-Shaw FactSet FreeCharge MakeMyTri | |
| Binary Trees | Diameter of a Binary Tree | Amazon Microsoft OYO Rooms | |
| Binary Trees | Top View of Binary Tree | Microsoft + Adobe + Expedia Group | |
| Binary Trees | Bottom View of Binary Tree | DE Shaw India | |
| Binary Trees | Diagonal Traversal of Binary Tree | Amazon Microsoft | |
| Binary Trees | Boundary Traversal of binary tree | Accolite Amazon FactSet Hike Kritikal Solutions | |
| Binary Trees | Construct Binary Tree from String with Brackets | Microsoft Morgan Stanley OYO Rooms Payu Samsung Snapdeal Flipkart | |
| Binary Trees | Minimum swap required to convert binary tree to binary search tree | Adobe Amazon | |
| Binary Trees | Duplicate subtree in Binary Tree | Google | |
| Binary Trees | Check if a given graph is tree or not | Microsoft Amazon | |
| Binary Trees | Lowest Common Ancestor in a Binary Tree | Accolite Amazon American Express Cisco Expedia Flipkart MakeMyTrip Microso | |
| Binary Trees | Min distance between two given nodes of a Binary Tree | Amazon Linkedin MakeMyTrip Ola Cabs Qualcomm Samsung | |
| Binary Trees | Duplicate Subtrees | Ola | |
| Binary Trees | Kth ancestor of a node in binary tree | Josh Technology Group | |
| Binary Trees | Binary Tree Maximum Path Sum | Samsung + Facebook | |
| Binary Trees | Serialize and Deserialize Binary Tree | Flipkart InMobi Linkedin MAQ Software Microsoft Paytm Quikr Yahoo | |
| Binary Trees | Binary Tree to DLL | Accolite Amazon Goldman Sachs Microsoft Morgan Stanley Salesforce Snapdea | |
| Binary Trees | Print all k-sum paths in a binary tree | Accolite Amazon Goldman Sachs | |
| | | | |
| | | | |
| Binary Search Trees | Lowest Common Ancestor of a Binary Search Tree | Accolite Amazon Flipkart MAQ Software Microsoft Samsung Synopsys | |
| Binary Search Trees | Binary Search Tree \| Set 1 (Search and Insertion) | Accolite Amazon Microsoft Paytm Samsung | |
| Binary Search Trees | Minimum element in BST | Microsoft | |
| Binary Search Trees | Predecessor and Successor | Google + Adobe + Goladman Sachs + Direct | |
| Binary Search Trees | Check whether BST contains Dead End | Walmart | |
| Binary Search Trees | Binary Tree to BST | HSBC | |
| Binary Search Trees | Kth largest element in BST | Accolite Amazon Samsung SAP Labs Microsoft | |
| Binary Search Trees | Validate Binary Search Tree | OYO Rooms Qualcomm Samsung Snapdeal VMWare Walmart Wooker Amazon | |
| Binary Search Trees | Kth Smallest Element in a BST | Accolite Amazon Google | |
| Binary Search Trees | Delete Node in a BST | Adobe Barclays | |
| Binary Search Trees | Flatten BST to sorted list | Microsoft | |
| Binary Search Trees | Preorder to Postorder | Amazon Linkedin Flipkart | |
| Binary Search Trees | Count BST nodes that lie in a given range | D-E-Shaw Google | |
| Binary Search Trees | Populate Inorder Successor for all Nodes | Sap labs | |
| Binary Search Trees | Convert Normal BST to Balanced BST | Paytm | |
| Binary Search Trees | Merge two BSTs | DE Shaw India | |
| Binary Search Trees | Given n appointments, find all conflicting appointments | Samsung | |
| Binary Search Trees | Replace every element | Samsung | |
| Binary Search Trees | Construct BST from given preorder traversal | Adobe Morgan Stanley Microsoft | |
| Binary Search Trees | Find median of BST in O(n) time and O(1) space | Amazon | |
| Binary Search Trees | Largest BST in a Binary Tree | Amazon D-E-Shaw Samsung Microsoft Flipkart | Important |
| | | | |
| | | | |
| Heaps & Hashing | Choose k array elements such that difference of maximum and minimum is minimized | | |
| Heaps & Hashing | Heap Sort | Adobe | |
| Heaps & Hashing | Top K Frequent Elements | Amazon Microsoft | |
| Heaps & Hashing | k largest elements in an array | Amazon Microsoft Walmart Adobe | |
| Heaps & Hashing | Next Greater Element | Amazon + Microsoft + Flipkart + Adobe | |
| Heaps & Hashing | K'th Smallest/Largest Element in Unsorted Array | ABCO Accolite Amazon Cisco Hike Microsoft Snapdeal VMWare Google Adobe | |
| Heaps & Hashing | Find the maximum repeating number in O(n) time and O(1) extra space | Accolite Amazon | |
| Heaps & Hashing | K-th smallest element after removing some integers from natural numbers | ABCO Accolite Amazon Cisco Hike Microsoft Snapdeal VMWare Google Adobe | |
| Heaps & Hashing | Find k closest elements to a given value | Amazon OYO Rooms | |
| Heaps & Hashing | K'th largest element in a stream | Amazon Cisco Hike OYO Rooms Walmart Microsoft Flipkart | |
| Heaps & Hashing | Connect Ropes | Amazoon + Oyo + Goldman Sachs | |
| Heaps & Hashing | Cuckoo Hashing | Amaxon | |
| Heaps & Hashing | Itinerary from a List of Tickets | Microsoft + Ola + eBay | |
| Heaps & Hashing | Largest Subarray with 0 Sum | Amazon MakeMyTrip Microsoft | |
| Heaps & Hashing | Count distinct elements in every window of size k | Accolite Amazon Microsoft | |
| Heaps & Hashing | Group Shifted Strings | Oracle | |
| Heaps & Hashing | Merge K Sorted lists | Microsoft + Ola + eBay | |
| Heaps & Hashing | Find Median from Data Stream | Adobe Amazon Apple Belzabar D-E-Shaw Facebook Flipkart Google Intuit Micro | |
| Heaps & Hashing | Sliding Window Maximum | Amazon Directi Flipkart Microsoft Google | |
| Heaps & Hashing | Find the smallest positive number | Accolite Amazon Samsung Snapdeal | |
| Heaps & Hashing | Find Surpasser Count of each element in array | Amazon Morgan Stanley Ola Cabs SAP Labs | |
| Heaps & Hashing | Tournament Tree and Binary Heap | Amazon Ola Cabs Samsung Synopsys Walmart Microsoft | |
| Heaps & Hashing | Check for palindrome | Amazon Cisco D-E-Shaw Facebook FactSet Morgan Stanley Paytm Zoho | |
| Heaps & Hashing | Length of the largest subarray with contiguous elements | Amazon Intuit Microsoft | |
| Heaps & Hashing | Palindrome Substring Queries | Amazon Morgan Stanley Ola Cabs SAP Labs | |
| Heaps & Hashing | Subarray distinct elements | Microsoft + Ola + eBay | |
| Heaps & Hashing | Find the recurring function | MAQ Software | |
| Heaps & Hashing | K maximum sum combinations from two arrays | Amazon | |
| | | | |
| | | | |
| Graphs | BFS | Samsung + Delhivery + SAP Labs | |
| Graphs | DFS | Samsung + Intuit + Goldman Sachs | |
| Graphs | Flood Fill Algorithm | Google + Adobe + Apple | |
| Graphs | Number of Triangles | IBM | |
| Graphs | Detect cycle in a graph | Lenksart | |
| Graphs | Detect cycle in an undirected graph | Samsung | |
| Graphs | Rat in a Maze Problem | Sharechat + Directi | |

| | | | |
|---|---|---|---|
| **Graphs** | Steps by Knight | Samsung | |
| **Graphs** | Clone graph | Google + MAQ Software + Apple + Facebook | |
| **Graphs** | Number of Operations to Make Network Connected | Samsung | |
| **Graphs** | Dijkstra's shortest path algorithm | Amazon | |
| **Graphs** | Topological Sort | Amazon + Google + Flipkart + Oyo + Fipkart + Samsung | |
| **Graphs** | Oliver and the Game | Sharechat + Directi | |
| **Graphs** | Minimum time taken by each job to be completed given by a Directed Acyclic Graph | Amazon | |
| **Graphs** | Find whether it is possible to finish all tasks or not from given dependencies | Directi + Sharechat | |
| **Graphs** | Find the number of islands | Razorpay | |
| **Graphs** | Prim's Algo | Visa | |
| **Graphs** | Negative Weighted Cycle | Amazon | |
| **Graphs** | Floyd Warshall | Google + Uber | |
| **Graphs** | Graph Coloring | Morgan Stanley | |
| **Graphs** | Snakes and Ladders | Goldman Sachs +Makemytrip | |
| **Graphs** | Kosaraju's Theorem | Paytm | |
| **Graphs** | Journey to moon | Lenksart + Payload | |
| **Graphs** | Vertex Cover | Intuit | |
| **Graphs** | M Coloring Problem | Uber | |
| **Graphs** | Cheapest Flights Within K Stops | Uber + Paypal | |
| **Graphs** | Find if there is a path of more than k length from a source | Cisco + Intuit | |
| **Graphs** | Bellman Ford | Sharechat + Directi | |
| **Graphs** | Bipartitie Graph | Microsoft Flipkart | |
| **Graphs** | Word-Ladder | Microsoft | |
| **Graphs** | Allen Dictionary | Samsung | |
| **Graphs** | Kruskals MST | Amazon Cisco Samsung | Important |
| **Graphs** | Total number spanning trees graph | Amazon Cisco Samsung Microsoft Flipkart | |
| **Graphs** | Travelling Salesman | Google + Microsoft + Opera | Important |
| **Graphs** | Find longest path directed acyclic graph | Google | |
| **Graphs** | Two Clique Problem | Microsoft | |
| **Graphs** | Minimise the cash flow | Intuit + Uber | |
| **Graphs** | Chinese postman | Intuit | |
| **Graphs** | Water Jug | Intuit + Uber | |
| **Graphs** | Water Jug 2 | MakeMyTrip MAQ Software | |
| | | | |
| **Tries** | Construct a trie from scratch | Accolite Amazon D-E-Shaw FactSet Microsoft | |
| **Tries** | Print unique rows in a given boolean matrix | Amazon Zoho | |
| **Tries** | Word Break Problem \| (Trie solution) | Amazon Google Hike IBM MAQ Software Microsoft Walmart Zoho | |
| **Tries** | Given a sequence of words, print all anagrams together | Amazon D-E-Shaw Goldman Sachs Morgan Stanley Snapdeal Microsoft | |
| **Tries** | Find shortest unique prefix for every word in a given list | Microsoft Google | |
| **Tries** | Implement a Phone Directory | Amazon + Microsoft + Snapdeal | |
| | | | |
| **DP** | Knapsack with Duplicate Items | Amazon | |
| **DP** | BBT counter | Microsoft | |
| **DP** | Reach a given score | Samsung | |
| **DP** | Maximum difference of zeros and ones in binary string | Ola | |
| **DP** | Climbing Stairs | Intuit | |
| **DP** | Permutation Coefficient | Amazon | |
| **DP** | Longest Repeating Subsequence | Google + Amazon | |
| **DP** | Pairs with specific difference | Ola | |
| **DP** | Longest subsequence-1 | Amazon | |
| **DP** | Coin Change | Microsoft+ Samsung + Barclays + Apple + Adobe | |
| **DP** | LIS | Amazon + Google + Facebook + Fidelity International | |
| **DP** | Longest Common Subsequence | Siemens + Amazon + Google | |
| **DP** | Word Break | Amazon + Google + Microsoft + Walmart + Apple + IBM | |
| **DP** | Combination Sum IV | Adobe Amazon Microsoft | |
| **DP** | House Robber | Apple + Uber | |
| **DP** | Houe Robber 2 | Arrays Dynamic Programming | |
| **DP** | Decode Ways | Adobe + Uber | |
| **DP** | Unique Paths | Google + Microsoft | |
| **DP** | Jumps Game | Facebook Amazon Microsoft Google | |
| **DP** | Knapsack Problem | Amazon Directi Flipkart GreyOrange Microsoft Mobicip Morgan Stanley Oracle | |
| **DP** | nCr | Google | |
| **DP** | Catalan Number | Amazon + Google | |
| **DP** | Edit Distance | Google + Goldman Sachs + Citrix | |
| **DP** | Subset Sum | Amazon + Google | |
| **DP** | Gold mine | Samsung | |
| **DP** | Assembly Line Scheduling | Goldman Sachs | |
| **DP** | Maximize The Cut Segments | Amazon OYO Rooms Microsoft | |
| **DP** | Maximum sum increasing subsequence | Amazon Morgan Stanley Microsoft | |
| **DP** | Count all subsequences having product less than K | Goldman Sachs | |
| **DP** | Maximum sum increasing subsequence | Amazon Morgan Stanley Microsoft | |
| **DP** | Egg dropping puzzle | Amazon D-E-Shaw Goldman Sachs Google Hike MakeMyTrip MAQ Software My | |
| **DP** | Max length chain | Amazon Microsoft | |
| **DP** | Largest Square in Matrix | Amazon Samsung | |
| **DP** | Maximum Path Sum | Amazon + Microsoft + Oyo + Directi | |
| **DP** | Minimum Number of Jumps | Adobe Amazon Housing.com Moonfrog Labs Walmart Microsoft Google Flipkar | |
| **DP** | Minimum removals from array to make max – min <= K | Amazon | |
| **DP** | Longest Common Substring | Webarch Club | |
| **DP** | Partition Equal Subset Sum | Amazon + Accolite + Traveloca + Adobe | |
| **DP** | Longest Palindromic Subsequnce | Amazon Google | |
| **DP** | Count Palindromic Subsequences | Myntra | |
| **DP** | Longest Palindromic Substring | Amazon + Microsoft + Samsung + Visa | |
| **DP** | Longest Alternating Sequence | Ola | |
| **DP** | Weighted Job Scheduling | Intuit | |
| **DP** | Coin Game | Salesforce | |
| **DP** | Coin Game Winner | Ola | |
| **DP** | Optimal Strategy for a game | Google + IBM | |
| **DP** | Word Wrap | Microsoft | |
| **DP** | Mobile numeric keypad | Amazon Microsoft | |
| **DP** | Maximum Length of Pair Chain | Amazon Microsoft | |
| **DP** | Matrix Chain Multiplication | Walmart + Flipkart | |
| **DP** | Maximum profit by buying and selling a share at most twice | Accolite Amazon Microsoft | |
| **DP** | Optimal BST | Google | |
| **DP** | Largest Submatrix with sum 0 | Amazon MakeMyTrip Microsoft | |
| **DP** | Largest area rectangular sub-matrix with equal number of 1's and 0's | Amazon Directi Intuit MakeMyTrip Microsoft Samsung Google Flipkart | |
| | | | |
| **Bit Manipulation** | Count set bits in an integer | Adobe Apple | |

| | | | |
|---|---|---|---|
| **Bit Manipulation** | Find the two non-repeating elements in an array of repeating elements | Accolite Amazon FactSet Google MakeMyTrip Microsoft Qualcomm Samsung | |
| **Bit Manipulation** | Program to find whether a no is power of two | Adobe | |
| **Bit Manipulation** | Find position of the only set bit | Microsoft | |
| **Bit Manipulation** | Count number of bits to be flipped to convert A to B | Maq Software | |
| **Bit Manipulation** | Count total set bits in all numbers from 1 to n | Microsoft | |
| **Bit Manipulation** | Copy set bits in a range | Facebook | |
| **Bit Manipulation** | Calculate square of a number without using *, / and pow() | Amazon | |
| **Bit Manipulation** | Divide two integers without using multiplication, division and mod operator | Microsoft | |
| **Bit Manipulation** | Power Set | Google + Adobe + Paytm | |
| | | | |
| | | | |
| **Segment Trees** | Range Sum Query - Immutable | | |
| **Segment Trees** | Range Minimum Query | Google Interview Qs | |
| **Segment Trees** | Range Sum Query - Mutable | Alibaba | |
| **Segment Trees** | Create Sorted Array through Instructions | Samsung + Accolite | |
| **Segment Trees** | Count of Range Sum | Walmart | |
| **Segment Trees** | Count of Smaller Numbers After Self | Codenation Google | |

**Web Development**
https://www.youtube.com/watch?v=ngc9gnGgUdA
https://www.youtube.com/watch?v=k3Vfj-e1Ma4
https://www.youtube.com/watch?v=VsUzmlZfYNg
https://www.youtube.com/watch?v=ZxKM3DCV2kE
https://www.youtube.com/watch?v=EKQCQL1G1JU
https://www.youtube.com/watch?v=RDV3Z1KCBvo

**Android App Development**
https://www.youtube.com/watch?v=fis26HvvDII
https://www.youtube.com/watch?v=7dAt-JMSCVQ
https://www.youtube.com/watch?v=bCXD5aADPfE
https://www.youtube.com/watch?v=BBWyXo-3JGQ
https://www.youtube.com/watch?v=j-LOab_PzzU
https://www.youtube.com/watch?v=BCSlZIUj18Y
https://www.youtube.com/watch?v=w4USF9e7b4U

**ios App Development**
https://www.youtube.com/watch?v=09TeUXjzpKs
https://www.youtube.com/watch?v=KCgYDCKqato
https://www.youtube.com/watch?v=x4DydJKVvQk
https://www.youtube.com/watch?v=3pIXMwvJLZs
https://www.youtube.com/watch?v=F2ojC6TNwws

**Blockchain Development**
https://www.youtube.com/watch?v=CgXQC4dbGUE
https://www.youtube.com/watch?v=Wn_Kb3MR_cU

**ML Projects**
https://www.youtube.com/watch?v=AWvsXxDtEkU
https://www.youtube.com/watch?v=qmqCYC-MBQo
https://www.youtube.com/watch?v=UQAKwWUYnuI
https://www.youtube.com/watch?v=KUFmCwCVXWs

**DSA Based Projects**

https://github.com/ayonious/File-Compression/tree/master/src/main/java/prog
https://github.com/clementmihailescu/Sorting-Visualizer/tree/master/client/app/algorithms
https://github.com/utsavipatil/Notepad
https://github.com/the-squad/sudoku-desktop-game