DAY – 1: (9/6/2024)

1. Frequency of Limited Range Array Elements – (Array, Hashing)

Array of N size which contains integers from 1 to P. The Task is to find the frequency of the elements making in-place changes within the array. Note: The elements greater than N in the array can be ignored for counting and modifying the array. subset

```
public static void frequencyCount(int arr[], int N, int P)
{
    // code here
    int hash[] = new int[P+1];

    for(int i = 0;i<N;i++){
        if(arr[i]<=P){

         hash[arr[i]]++;

                }
    }
        for(int i = 0;i<N;i++){
        if(i<P){
        arr[i] = hash[i+1];
        }
        else{
            arr[i] = 0;
        }
```

```
//Function to count the frequency of all elements from 1 to N in
public static void frequencyCount(int arr[], int N, int P)
{
    // code here
    HashMap <Integer,Integer> map = new HashMap<>();

    for(int i =0;i<N;i++){
        if(map.containsKey(arr[i])){
            map.put(arr[i],map.get(arr[i])+1);
        }
        else{
            map.put(arr[i],1);
        }
    }

    for(int i = 0;i<N;i++){
        arr[i] = map.getOrDefault(i+1,0);
    }
}
```

3355

2. Convert an array into Zig-Zag fashion – (Arrays)

Given an array arr of distinct elements of size n, the task is to rearrange the elements of the array in a zig-zag fashion so that the converted array should be in the below form:

arr[0] < arr[1] > arr[2] < arr[3] > arr[4] < . . . . arr[n-2] < arr[n-1] > arr[n].

```
class Solution {
    public static void zigZag(int n, int[] arr) {
        for(int i = 0;i<n-1;i=i+1){

            if(i%2==0){
        if(arr[i]>arr[i+1]){
            int temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;
        }
            }
            else{
        if(arr[i] < arr[i+1]){
            int temp = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = temp;

        }
            }
```

3. Subarray Sums divisible by K : (HashMap, Prefix Sum)

Given an integer array nums and an integer k, return the number of non-empty subarrays that have
a sum divisible by k. A subarray is a contiguous part of an array
Input: nums = [4,5,0,-2,-3,1], k = 5
Output: 7 **[4, 5, 0, -2, -3, 1], [5], [5, 0], [5, 0, -2, -3], [0], [0, -2, -3], [-2, -3]**

The main idea behind this problem is finding the remainders of the sum and storing them in a hashmap
(O(1) retrieval of values). If the remainder is occurring again {-> (4 9 9 7..) remainders %k (4 4 4 2…) } there must be a value divisible by k or a sum divisible by k.

```
for(int x : nums){
    prefixsum += x;

    mod = prefixsum%k;
    if(mod<0) mod += k;

    if(map.containsKey(mod)){
        count += map.get(mod);
        map.put(mod,map.get(mod)+1);
    }
    else{
        map.put(mod,1);
    }

}

return count;
```

Another approach is Brute-force, which takes $O(n^2)$.

DAY – 2: (10/6/2024)

4. Height Checker – (Arrays, Counting Sort)

You are given an integer array heights representing the current order that the students are standing in.

Return the number of indices where heights[i] != expected[i].

Input: heights = [1,1,4,2,1,3]

Output: 3

Explanation: heights: [1,1,4,2,1,3]

expected: [1,1,1,2,3,4] Indices 2, 4, and 5 do not match.

```
class Solution {
    public int heightChecker(int[] heights) {
        int x[] = new int[heights.length];
        x = heights.clone();
        Arrays.sort(x);
        int count = 0;
        for(int i=0;i<heights.length;i++){
            if(x[i] != heights[i]) count++;
        }
        return count;
```

```
public int heightChecker(int[] heights) {
    int heightfreq [] = new int[101];
    for(int i = 0;i<heights.length;i++){
        heightfreq[heights[i]]++; }
    //Counting sort
    int exp[] = new int[heights.length];
    int k = 0;
    for(int i =0;i<101;i++){
        int c = heightfreq[i];

        while(c-- > 0){
            exp[k++] = i;
        }
    }
    k = 0;
    for(int i=0;i<heights.length;i++){
        if(exp[i]!=heights[i]) k++; }
    return k;
```

5. It means the nut can only be compared with the bolt and the bolt can only be compared with the nut to see which one is bigger/smaller. The elements should follow the following order: { !,#,$,%,&,*,?,@,^ }.

Input: n = 5, nuts[] = {@, %, $, #, ^}, bolts[] = {%, @, #, $ ^}

Output:

$ % @ ^

$ % @ ^

```
HashSet<Character> set = new HashSet<>();
String s = "!,#,$,%,&,*,?,@,^";
char c [] = s.toCharArray();
for(int i =0;i<nuts.length;i++){
    set.add(nuts[i]);
}
char x[] = new char[nuts.length];
int k=0;
for(char i : c){
    if(set.contains(i))
        x[k++] = i;
}
for(int i =0;i<x.length;i++){
nuts[i] = x[i];
  bolts[i] = x[i];
}
```

DAY – 3 : (11/6/2024)

6. Selection Sort – $O(N^2)$ 7. Bubble Sort – $O(N^2)$ 8. Insertion Sort – $O(N^2)$

```
static void selection_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int mini = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[mini]) {
                mini = j;
            }
        }
        //swap
        int temp = arr[mini];
        arr[mini] = arr[i];
        arr[i] = temp;
    }
```

```
//code here
for(int i = n-1;i>=0;i--){
    for(int j = 0;j<=i-1;j++){
        if(arr[j] > arr[j+1]){
            arr[j] = arr[j]^arr[j+1];
            arr[j+1] = arr[j]^arr[j+1];
            arr[j] = arr[j]^arr[j+1];
        }
    }
}
```

```
static void insertion_sort(int[] arr, int n) {
    for (int i = 0; i <= n - 1; i++) {
        int j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            int temp = arr[j - 1];
            arr[j - 1] = arr[j];
            arr[j] = temp;
            j--;
        }
```

9. Relative Sort Array – (HashMap, TreeSet, Arrays)

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1. Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in ascending order.

Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19],
arr2 = [2,1,4,3,9,6]

Output: [2,2,2,1,4,3,3,9,6,7,19]

O(nlogn) O(n)

```
HashMap<Integer,Integer> map = new HashMap<>();
for(int i =0;i<arr1.length;i++){
 if(map.containsKey(arr1[i])){
     map.put(arr1[i],map.get(arr1[i])+1);
 }
 else{
     map.put(arr1[i],1); } }
 int k = 0;
 for(int i:arr2){
     if(map.containsKey(i)){
         int x = map.get(i);
         while(x-- >0){
             arr1[k++] = i;
         }
         map.remove(i);}}
TreeSet<Integer> lis = new TreeSet<>();
for(Integer x : map.keySet()){
 lis.add(x);     }
for(Integer x: lis){
 int y = map.get(x);
 while(y-- > 0){
```

```
public int[] relativeSortArray(int[] arr1, int[] arr2) {
    int max = 0;
    for(int i : arr1){
        max = Math.max(max,i);
    }
   int ans[] = new int[max+1];
   for(int i =0;i<arr1.length;i++){
    ans[arr1[i]]++;
   }
   int k = 0;
   //Counting Sort
   for(int i =0;i<arr2.length;i++){
    while(ans[arr2[i]]-- > 0){
        arr1[k++] = arr2[i];} }
   for(int i =0;i<ans.length;i++){
    while(ans[i]-- > 0){
        arr1[k++] = i; } }
   return arr1;
```

10. Maximum Tip Calculator

To maximize total tips, they must distribute the orders such that: A can handle at most x orders, B can handle at most y orders.

Given that x + y ≥ n, all orders can be managed by either A or B. Return the maximum possible total tip after processing all the orders.

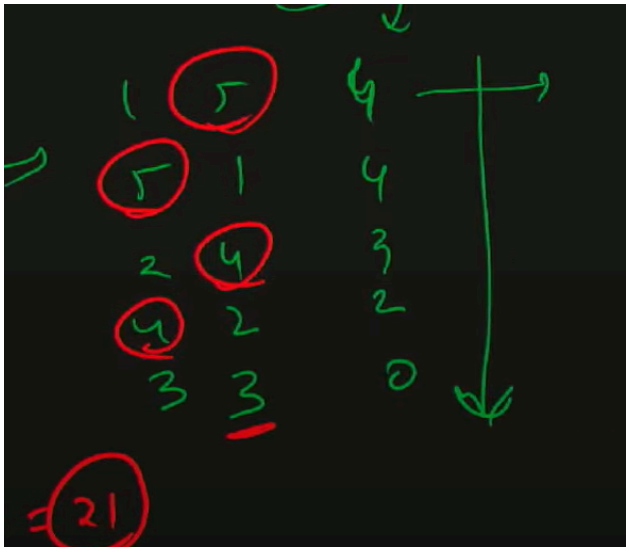
Input: n = 5, x = 3, y = 3, arr = {1, 2, 3, 4, 5}, brr = {5, 4, 3, 2, 1} Output: 21

Explanation: Person A will serve the 3rd, 4th, and 5th order while Person B will serve the rest so the total tip from

A = 3+4+5 & B = 5 + 4 i.e. 21.

Lambda function for sorting in descending order

```
int [][] a = new int[n][3];

for(int i =0;i<n;i++){
    a[i][0] = arr[i];
    a[i][1] = brr[i];
    a[i][2] = Math.abs(arr[i] - brr[i]);
}
Arrays.sort(a, (p,q)-> q[2] - p[2]);
long res = 0;
for(int i =0;i<n;i++){
    if(x==0){  //arr is empty
        res += a[i][1];
    }
    else if(y==0){  //brr is empty
        res += a[i][0];
    }
    else{
        if(a[i][0] >= a[i][1]){
         res+= a[i][0];
         x--;
        }
        else
        {
            res+= a[i][1];
            y--;
        }
```



Day – 4 (12/6/2024)

11. Sort Colors – (Arrays, Two Pointers)

Sort the numbers 0,1,2 without using built-in functions O(n)

```
int low = 0;
int high = nums.length -1;
int mid = 0;
while(mid <=high){

    if(nums[mid] == 0){
        swap(nums,mid,low);
        mid++;
        low++;
    }
    else if(nums[mid] == 2){
        swap(nums,mid,high);
        high--;
    }
    else{
        mid++;
    }
```

12. Count numbers containing 4

Input: n = 44

Output: 9

Explanation: 4, 14, 24, 34, 40, 41, 42, 43 & 44, there are total 9 numbers containing 4 as a digit.

O(nlogn)

```
int count = 0;

for(int i = 1;i<=n;i++){
    int x = i;

    while(x>0){
        int r = x%10;
        if(r==4){
         count++;
         break;

        }

        x = x/10;
    }
}
return count;
}
```

O(nlogn) -> DP

```
    while(a>0){
        if(a%10 == 4) return true;
        a = a/10;
    }
    return false;
}
public static int countNumberswith4(int n) {
    boolean []dp = new boolean[n+1];
    int count = 0;
    for(int i = 1;i<=n;i++){
        if(dp[i])
            count++;
        if(contains(i)){
            count++;
            dp[i] = true;
        }
    }
    return count;
```

DAY – 5 (13/6/2024)

13. Pandovan Sequence – O(n) (Arrays)

P(n) = P(n-2) + P(n-3)

```
//code here.
int arr[] = new int[n+1];
if(n==0 || n==1 || n==2) return 1;
arr[0] = 1;
arr[1] = 1;
arr[2] = 1;
int mod = 1000000007;
for(int i = 3; i<=n;i++){
    arr[i] = (arr[i-2] + arr[i-3])%mod;
}
return arr[n];
```

14. Minimum Number of Moves to Seat Everyone – (Arrays, Greedy, Counting Sort)

Input:

seats = [4,1,5,9]

students = [1,3,2,6]

Output: 7

- The first student is not moved.
- The second student is moved from position 3 to position 4 using 1 move.
- The third student is moved from position 2 to position 5 using 3 moves.
- The fourth student is moved from position 6 to position 9 using 3 moves.

In total, 0 + 1 + 3 + 3 = 7 moves were used.

O(nlogn)

```
Arrays.sort(seats);
Arrays.sort(students);
int count = 0;
for(int i = 0;i<seats.length;i++){
    count += Math.abs(seats[i] - students[i]);
}
return count;
```

O(n)

```
int minMovesToSeat(vector<int>& seats, vector<int>& students) {
    vector<int> seatLoc(*max_element(seats.begin(), seats.end())+1, 0);
    vector<int> studentsFreq(*max_element(students.begin(), students.end())+1, 0);
    for(auto loc : seats) seatLoc[loc]++;
    for(auto student : students) studentsFreq[student]++;

    int ans = 0, j=0;
    for(int i=0; i<seatLoc.size(); i++){
        if(seatLoc[i] == 0) continue;
        while(studentsFreq[j]==0) j++;
        ans += abs(i-j);
        studentsFreq[j]--;
        seatLoc[i]--;
        if(seatLoc[i] != 0) i--;
    }
```

DAY – 6 (14/6/2024)

15. Minimum Increment to Make Array Unique – (Sorting, Greedy, Counting sort)

You are given an integer array nums. In one move, you can pick an index i where 0 <= i < nums.length and

increment nums[i] by 1.Return the minimum number of moves to make every value in nums unique.

nums = [1,2,2]

Output: 1

Explanation: After 1 move, the array could be [1, 2, 3]

O(nlogn)

O(n)

```
Arrays.sort(nums);
int count = 0;
int prev = nums[0];
for(int i = 1;i<nums.length;i++){
    if(nums[i] <= prev){
        count += prev + 1 - nums[i];
        nums[i] = prev + 1;

    }
    prev = nums[i];
}
return count;
```

```
int max = -1;
for(int i : nums){
    if(i>max) max = i;
}

int arr [] = new int[max+1+nums.length];

for(int i =0;i<nums.length;i++){
    arr[nums[i]]++;
}

int count = 0;
for(int i = 0;i<arr.length;i++){
    if(arr[i]<=1) continue;

    int extra = arr[i] - 1;
    arr[i+1] += extra;
    count+= extra;
}
return count;
```

DAY – 7 (15/6/2024)

16. Mobile numeric keypad – (DP)

You can only press the current button or buttons that are directly up, left, right, or down from it (for ex if you press 5, then pressing 2, 4, 6 & 8 are allowed). Diagonal movements and pressing the bottom row corner buttons (* and #) are prohibited. Given a number n, find the number of possible unique sequences of length n that you can create by pressing buttons. You can start from any digit.

```
private static final char[][] keypad = {
    {'1','2','3'},
    {'4','5','6'},
    {'7','8','9'},
    {'*','0','#'}

};
private static final int[][] direc = {
    {-1,0},
    {1,0},
    {0,-1},
    {0,1}

};

public long getCount(int n) {
    // Your code goes here

    if(n<=0) return 0;
    if(n==1) return 10;

    long [][] dp = new long[n+1][10];

    for(int i =0;i<=9;i++){
        dp[1][i] = 1;
    }
```

```
    for(int length = 2; length <=n;length++){
        for(int dig = 0;dig<=9;dig++){
            int[] pos = findpos(dig);
            dp[length][dig] = 0;


        for(int[] d: direc){
            int nrow = pos[0] + d[0];
            int ncol = pos[1] + d[1];

            if(isValid(nrow,ncol)){
                int newdigit = keypad[nrow][ncol] - '0';
                dp[length][dig] += dp[length - 1][newdigit];
            }
        }
        dp[length][dig] += dp[length -1][dig];

}
    }

    long total = 0;
    for(int i =0;i<10;i++){
        total += dp[n][i];
    }

    return total;
}
```

```
private int[] findpos(int digit){
    char c = (char)(digit + '0'); //int to char
    for(int i = 0;i<keypad.length;i++){
        for(int j = 0;j<keypad[i].length;j++){
            if(keypad[i][j] == c){
                return new int []{i,j};
            }

        }
    }
    return null;


}

private boolean isValid(int r, int col){
    return r>=0 && r<keypad.length && col>=0 &&
    col < keypad[0].length && keypad[r][col] != '*' &&
    keypad[r][col] != '#';
}
```

18. IPO – (Greedy, PriorityQueue, Arrays)

You are given n projects where the ith project has a pure profit, profit [i] and a minimum capital of capital[i] is needed to start it.

Initially, you have w capital. When you finish a project, you will obtain its pure profit and the profit will be added to your total capital. Pick a list of at most k distinct

projects from given projects to maximize your final capital, and return the final maximized capital.

k = 2, w = 0, profits = [1,2,3], capital = [0,1,1]

Output: 4

Explanation: Since your initial capital is 0, you can only start the project indexed 0.

After finishing it you will obtain profit 1 and your capital becomes 1.

With capital 1, you can either start the project indexed 1 or the project indexed 2.

Since you can choose at most 2 projects, you need to finish the project indexed 2 to get the maximum capital.

Therefore, output the final maximized capital, which is 0 + 1 + 3 = 4.

O(nlogn)

```
int n = profits.length;
int [][] arr = new int[n][2];
for(int i =0;i<n;i++){
    arr[i][0] = capital[i];
    arr[i][1] = profits[i];
}
Arrays.sort(arr, (a,b) -> Integer.compare(a[0],b[0]));
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
int j = 0;
for(int i = 0;i<k;i++){
    while(j<n && arr[j][0]<=w){
        pq.add(arr[j][1]);
        j++;

    }

    if(pq.isEmpty()) break;
    w+= pq.poll();
}
```

DAY – 8 (16/6/2024)

19. Minimum Prime Pair with Target Sum – (Arrays, ArrayList) -
Sieve of Eratosthenes

O(nloglogn)

```java
//Sieve of Eratosthenes
public static boolean isprime(int n, boolean [] primes){
    primes[0] = false;
    primes[0] = false;
    for(int i = 2;i<=n;i++)
    primes[i] = true;
    for(int i = 2;i*i<=n;i++){
        if(primes[i] == true){
            for(int p = i*i;p<=n;p+=i){
                primes[p] = false;
            }
        }
    }
    return false;
}
```

```java
public static ArrayList<Integer> getPrimes(int n) {
    ArrayList<Integer> arr = new ArrayList<>();
    boolean primes [] = new boolean[n+1];
    isprime(n,primes);

    for(int i =2;i<n;i++){
        int x = i;
        int y = n-i;
        if(primes[i] && primes[n-i]){
            arr.add(x);
            arr.add(y);
            break;
        }
    }
    if(arr.isEmpty()){
        arr.add(-1);
        arr.add(-1);
        return arr;
    }

    return arr;
}
```

***Bitset DP to find all possible sums in array**

20. Patching Array – (Greedy, Arrays) [HARD]

Given a sorted integer array nums and an integer n, add/patch elements to the array such that any number in the range [1, n] inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

O(logn)

```java
class Solution {
    public int minPatches(int[] nums, int n) {
        int minpatch = 0;
        long maxnumber =0;
        int i=0,N=nums.length;

        while(maxnumber < n){
            if(i<N && maxnumber + 1 >= nums[i]){
                maxnumber+= nums[i];
                i++;
            }
            else{
                minpatch++;
                maxnumber += (maxnumber+1);
            }
        }
        return minpatch;
    }
}
```

Day - 9 (17/6/2024)

21. Sum of Square Numbers - [Arrays, 2 pointers]

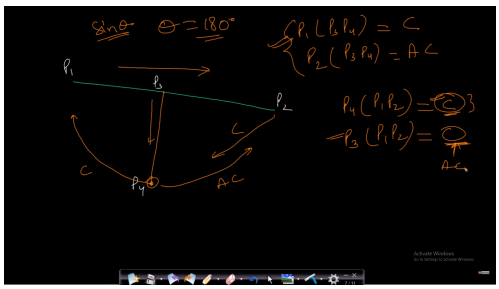
Given a non-negative integer c, decide whether there're two integers a and b such that a2 + b2 = c.

Input: c = 5 Output: true

Explanation: 1 * 1 + 2 * 2 = 5

(Osqrt(c))

```
long i = 0;
long j = (long)Math.sqrt(c);
while(i<=j){
    long sum = i*i + j*j;
    if(sum > c) j--;
    else if(sum<c) i++;
    else {
        return true;
}
}
return false;
```



22. Check If two Line segments Intersect - [Arrays] Given the coordinates of the endpoints(p1,q1, and p2,q2) of the two line segments. Check if they intersect or not. If the Line segments intersect return true otherwise return false.

```
int direction(int p[],int q[], int r[]){

    long   val = (long)(q[1]-p[1])*(r[0]-q[0]) -
    (long)(r[1]-q[1])*(q[0]-p[0]);
    if(val == 0) return 0;
    return (val > 0) ? 1 : 2;

}

boolean func(int p[], int q[], int r[]){
    if(r[0] <= Math.max(p[0],q[0]) && r[0] >= Math.min(p[0],q[0]) &&
    r[1] <= Math.max(p[1],q[1]) && r[1] >= Math.min(p[1],q[1]))
    return true;

    return false;
```

Anticlockwise 2

Clockwise 1

Collinear 0

Direction = m - m1

```
}
String doIntersect(int p1[], int q1[], int p2[], int q2[]) {
    int o1 = direction(p1,q1,p2);
    int o2 = direction(p1,q1,q2);
    int o3 = direction(p2,q2,p1);
    int o4 = direction(p2,q2,q1);

    if(o1!=o2 && o3!=o4) return "true";

    if(o1 == 0 && func(p1,q1,p2)) return "true";

    if(o2 == 0 && func(p1,q1,q2)) return "true";

    if(o3 == 0 && func(p2,q2,p1)) return "true";

    if(o4 == 0 && func(p2,q2,q1)) return "true";

    return "false";
```

Day - 10 (18/6/2024)

23. Most Profit Assigning Work - [PriorityQueue, Greedy, Binary Search]

You have n jobs and m workers. You are given three arrays: difficulty, profit, and worker where: difficulty[i] and profit[i] are the difficulty and the profit of the ith job, and worker[j] is the ability of jth worker (i.e., the jth worker can only complete a job with difficulty at most worker[j]). Every worker can be assigned at most one job, but one job can be completed multiple times.

difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]

Output: 100 Explanation: Workers are assigned jobs of difficulty [4,4,6,6] and they get a profit of [20,20,30,30] separately.

O(NLOGN)

```
PriorityQueue<int []> pq = new PriorityQueue<>((a,b) -> b[0] - a[0]);
for(int i=0;i<difficulty.length;i++)
    pq.offer(new int[]{profit[i],difficulty[i]});
Arrays.sort(worker);
int n = worker.length;
int res = 0;

for(int i = n-1;i>=0 && !pq.isEmpty();){
    if(pq.peek()[1] <= worker[i]){
        res+= pq.peek()[0];
        i--;
    }
    else{
        pq.poll();
    }
    }
    return res;
```

Another approach is using binary search. Sort difficulty and profit via difficulty. Assign increasing profit ->

0 - 20, 1 - 10; Automatically worker if he has 1, then he'll take 20 profit. So update 1 - 20. Use Binary search, find the max difficulty exactly less than worker[i]. Append profit.

24. Number of Rectangles in a Circle

Given a circular sheet of radius, r. Find the total number of rectangles with integral length and width that can be cut from the sheet that can fit on the circle, one at a time.

Input: r=2 Output: 8

(1x1)(1x2)(1x3)(2x1)(2x2)(2x3)(3x1)(3x2)

```
    int count = 0;

    for(int i = 1;i<2*r;i++){
        int y = 1;

        while(4*r*r >= i*i + y*y){
            count++;
            y++;
        }
    }
    return count;
}
```

Day - 11 (19/6/2024)

25. Minimum Number of Days to Make m Bouquets - [Arrays, Binary Search]

You are given an integer array bloomDay, an integer m and an integer k. You want to make m bouquets. To make a bouquet, you need to use k adjacent flowers from the garden. The garden consists of n flowers, the ith flower will bloom in the bloomDay[i] and then can be used in exactly one bouquet.

bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3 Output: 12

After day 7: [x, x, x, x, _, x, x] After day 12: [x, x, x, x, x, x, x]

It is obvious that we can make two bouquets in different ways.

```
public boolean ispossible(int [] bloomDay, int day, int m,int k){
    int count = 0;
    int noday = 0;
    for(int i = 0;i<bloomDay.length;i++){
        if(bloomDay[i] <= day){
            count++;
        }
        else{
            noday += (count/k);
            count = 0;
        }
    }
    noday += (count/k);

    if(noday >= m) return true;


return false;
```

O(nlogn) O(logn)

```java
int min = Integer.MIN_VALUE;
int max = 0;
for(int i: bloomDay){
    if(min > i ){
        min = i;
    }
    if(i > max){
        max = i;
    }
}
if(bloomDay.length < m*k) return -1;
for(int i = min;i<=max;i++){
    if(ispossible(bloomDay,i,m,k)){
        return i;
    }
}
return -1;
```

```java
if(bloomDay.length < (long) m*k) return -1;

int low = 1;
int high = (int) 1e9;
while(low<high){
    int mid = low + (high-low)/2;
    if(ispossible(bloomDay,mid,m,k)){

        high = mid ;
    }
    else{
        low = mid + 1;
    }
}
return low;
```

26. Find maximum volume of a cuboid

```java
double sqrt = Math.sqrt(perimeter*perimeter - 24*area);
double d1 = (perimeter - sqrt)/12.0;
double d2 = (perimeter/4.0) - 2*d1;
double vol = Math.pow(d1,2)*d2;
return (vol*100.0)/100.0;
```

Day - 12 (20/6/2024)

27. Magnetic Force Between Two Balls - [Binary Search]

Rick stated that magnetic force between two different balls at positions x and y is |x - y|. Given the integer array position and the integer m. Return the required force. Input: position = [1,2,3,4,7], m = 3 Output: 3

Explanation: Distributing the 3 balls into baskets 1, 4 and 7 will make the magnetic force between ball pairs [3, 3, 6]. The minimum magnetic force is 3. We cannot achieve a larger minimum magnetic force than 3.

*High = (pos[n-1] - pos[0])/m-1 -> the maximum possible minimum distance when placing **m** balls in **n** baskets.*

O(nlognn)

```java
Arrays.sort(position);
int low = 1;
int high = (position[position.length -1] - position[0])/(m-1);
int ans = 1;
while(low<=high){
    int mid = low + (high - low)/2;
    if(isPossible(position,mid,m)){
        ans = mid;
        low = mid + 1;
    }
    else{
        high = mid - 1;
    }
}
return ans;
```

```java
public boolean isPossible(int []arr, int mid, int m){
    int count = 1;
    int l = arr[0];
    for(int i =0;i<arr.length;i++){
        if(arr[i] - l >= mid){
            l = arr[i];
            count++;
        }
        if(count >= m) return true;
    }
    return false;
}
```

HASHMAP IMPLEMENTATION:

```
class MyHashMap
{
    class Node
    {
        int key;
        int value;
        Node next;
        public Node(int k,int v)
        {
            key = k;
            value = v;
        }
    }

    Node map[];
    int m = 10001;
```

```
public MyHashMap()
{
    map = new Node[10001];
}

public void put(int key, int value)
{
   int k = key % m;

    if(map[k] == null)
    {
        map[k] = new Node(key , value);
    }
    else
    {
       Node p = map[k];
```

```
while(p!=null)
{
    if(p.key == key)
    {
        p.value = value;
        break;
    }
    p = p.next;
}

if(p == null)
{
    Node node = new Node(key,value);
   node.next = map[k];
   map[k] = node;
}
```

```
public int get(int key)
{
    int k = key % m;

    if(map[k] == null)
        return -1;

    Node p = map[k];

    while(p!= null)
    {
        if(p.key == key)
            return p.value;

        p = p.next;
    }

    return -1;
```

```
public void remove(int key)
{
    int k = key % m;

    Node p = map[k];

    while(p!= null)
    {
        if(p.key == key)
        {
            p.value = -1;
            break;
        }

        p = p.next;
    }
```

Day - 13 (21/6/2024)

28. Compare two fractions - [Strings]

Input: str = "5/6, 11/45" Output: 5/6

Explanation: 5/6=0.8333 and 11/45=0.2444, So 5/6 is greater fraction.

```
String compareFrac(String str) {
    String [] temp = str.split("[/,]");
    int a = Integer.parseInt(temp[0]);
    int b = Integer.parseInt(temp[1]);
    int c = Integer.parseInt(temp[2].trim());
    int d = Integer.parseInt(temp[3]);
    int ad = a*d;
    int cb = c*b;
    String ans = "";

    if(ad >cb) ans = a +"/"+b;
    else if(ad<cb) ans = c +"/"+d;

    else ans = "equal";

    return ans;
```

29. Grumpy Bookstore Owner - [Arrays, Sliding Window]

Return the maximum number of customers that can be satisfied throughout the day.

Input: customers = [1,0,1,2,1,1,7,5], grumpy = [0,1,0,1,0,1,0,1], minutes = 3

Output: 16

Explanation: The bookstore owner keeps themselves not grumpy for the last 3 minutes.
The maximum number of customers that can be satisfied = 1 + 1 + 1 + 1 + 7 + 5 = 16.

O(n)

```
int n = grumpy.length;

int sum0 = 0;
int sum1 = 0;
int max = 0;
for(int i = 0;i<n;i++){
    if(grumpy[i] == 0) sum0+=customers[i];
    if(i<minutes){
        sum1 += (grumpy[i] == 1) ?customers[i]:0;
    }
    else{
    sum1 += ((grumpy[i] == 1) ?customers[i] : 0) -
    ((grumpy[i-minutes] == 1) ?customers[i-minutes] : 0);
}
max = Math.max(max,sum1);
}
return max+sum0;
```

Day - 14 (22/6/2024)

30. Count Number of Nice Subarrays - [Arays, Sliding Window, Prefix sum]

Subarray is nice if it contains k odd numbers

nums = [1,1,2,1,1], k = 3 Output: 2

The only sub-arrays with 3 odd numbers are [1,1,2,1] and [1,2,1,1].

O(n)

```
public int numberOfSubarrays(int[] nums, int goal) {
    return getAtmost(nums,goal) - getAtmost(nums,goal-1);
}
private int getAtmost(int [] nums, int k){
    int s = 0;
    int count = 0;
    for(int i =0,j=0;i<nums.length;i++){
        s += nums[i]%2;
        while(s>k){
            s -= nums[j++]%2;
        }
        count += (i-j)+1;
    }
    return count;
}
```

O(n) - prefix sum (y - x = k -> y - k = x, add the freq of x in the ans)

```
  public int numberOfSubarrays(int [] nums, int
      int s = 0;
      int ans = 0;
      int [] arr = new int[nums.length+1];
      arr[0] = 1;
      for(int i =0;i<nums.length;i++){
          s += nums[i]%2;
          if(s>=k){
              ans += arr[s - k];
          }

          arr[s]++;



eturn ans;
  }
```

31. Extract Number from String

Given a sentence containing several words and numbers. Find the largest number among them which does not contain 9. If no such number exists, return -1.

Note: Numbers and words are separated by spaces only.

sentence="This is alpha 5057 and 97" Output: 5057

Explanation: 5057 is the only number that does not have a 9.

O(m+n+klogk)

```
long ExtractNumber(String sentence) {
    String [] s = sentence.split(" ");
  ArrayList<Long> arr = new ArrayList<>();
  for(int i = 0;i<s.length;i++){
      if(Character.isDigit(s[i].charAt(0)) && isValid(s[i]))
      arr.add(Long.parseLong(s[i]));
  }
  Collections.sort(arr);
  if(arr.size() != 0) return arr.get(arr.size() - 1);

  return -1;
}
boolean isValid(String i){
    for(int j = 0;j<i.length();j++){
        if(!(Character.isDigit(i.charAt(j))) || (i.charAt(j) == '9') ){
            return false;
        }
    }
    return true;
```

Day - 15 (23/6/2024)

32. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Given an array of integers nums and an integer limit, return the size of the longest non-empty subarray such that the absolute difference between any two elements of this subarray is less than or equal to limit.

Input: nums = [8,2,4,7], limit = 4 Output: 2

Explanation: All subarrays are:

[8] with maximum absolute diff |8-8| = 0 <= 4.

[8,2] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.

[2] with maximum absolute diff |2-2| = 0 <= 4.

[2,4] with maximum absolute diff |2-4| = 2 <= 4.

[2,4,7] with maximum absolute diff |2-7| = 5 > 4.

[4] with maximum absolute diff |4-4| = 0 <= 4.

[4,7] with maximum absolute diff |4-7| = 3 <= 4.

[7] with maximum absolute diff |7-7| = 0 <= 4.

Therefore, the size of the longest subarray is 2.

O(n)

```
LinkedList<Integer> inc = new LinkedList<>();
LinkedList<Integer> dec = new LinkedList<>();
int left = 0;
int ans = 0;
for(int right = 0;right<nums.length;right++){
    int x = nums[right];
    while(inc.size() > 0 && nums[right] < inc.getLast()){
        inc.removeLast();
    }
    inc.add(x);
    while(dec.size() > 0 && nums[right] > dec.getLast()){
        dec.removeLast();
    }
    dec.add(x);
    while(dec.getFirst() - inc.getFirst() > limit){
        if(dec.getFirst() == nums[left]){
            dec.removeFirst();
        }
        if(inc.getFirst() == nums[left]) inc.removeFirst();
    left++;
    }
    ans = Math.max(ans,(right - left)+1);
}

return ans;
```

-> find the absolute difference of the window (max - min)

33. Print Bracket Number

Input: str = "((()("

Output: 1 2 3 4 4 5

O(n)

```
ArrayList<Integer> bracketNumbers(String str) {
    int x = 1;
    Stack<Integer> s = new Stack<>();
    ArrayList<Integer> ans = new ArrayList<>();

    for(int i = 0;i<str.length();i++){
        if(str.charAt(i) == '('){
            ans.add(x);
            s.push(x);
            x++;
        }

        else if(str.charAt(i) == ')'){
            ans.add(s.pop());
        }
    }
    return ans;
```

34. Second Largest

Without using sort function

O(n)

```
  int second = Integer.MIN_VALUE;
  int first = Integer.MIN_VALUE;

  for(int i:arr){
      if(i> first){
          second = first;
          first = i;
      }
      else if(i>second && i!=first){
          second = i;
      }
  }

  return (second == Integer.MIN_VALUE) ? -1: second;
```

Day - 16 - (24/6/2024)

35. Summed Matrix - [Arrays]

A matrix is constructed of size n*n and given an integer ‘q’. The value at every cell of the matrix is given as, M(i,j) = i+j, where ‘M(i,j)' is the value of a cell, ‘i’ is the row number, and ‘j’ is the column number. Return the number of cells having value ‘q’.

Input: n = 4, q = 7

Output: 2

Explanation: Matrix becomes

2 3 4 5

3 4 5 6

4 5 6 7

5 6 7 8 The count of 7 is 2.

```
// code here
if(q>2*n) return 0;
else if(q>n) return 2*n - q + 1;

return q -1;
```

36. Minimum Operations to Make Binary Array Elements Equal to One I - [sliding window]

Choose any 3 consecutive elements from the array and flip all of them. Flipping an element means changing its value from 0 to 1, and from 1 to 0. Return the minimum number of operations required to make all elements in nums equal to 1. If it is impossible, return -1.

O(n)

```
int ans = 0;
for(int i = 0;i<nums.length -2;i++){

        if(nums[i] == 1) continue;

         for(int j = i;j<=i+2;j++){
            nums[j] = nums[j]^1;
    }
    ans++;
}
if(nums[nums.length -1] == 0 || nums[nums.length -2] == 0) return -1;

return ans;
```

37. Minimum Operations to Make Binary Array Elements Equal to One II - [sliding window]

You can do the following operation on the array any number of times (possibly zero): Choose any index i from the array and flip all the elements from index i to the end of the array. Flipping an element means changing its value from 0 to 1, and from 1 to 0.

Return the minimum number of operations required to make all elements in nums equal to 1.

O(n)

```
int times = 0;

for(int i = 0;i<nums.length;i++){

    if((nums[i] == 1 && times%2 == 1) || (nums[i] == 0 && times%2 == 0)){
        times++;
    }
}
return times;
```

Input: nums = [0,1,1,0,1] Output: 4

Choose the index i = 1. The resulting array will be nums = [0,0,0,1,0].

Choose the index i = 0. The resulting array will be nums = [1,1,1,0,1].

Choose the index i = 4. The resulting array will be nums = [1,1,1,0,0].

Choose the index i = 3. The resulting array will be nums = [1,1,1,1,1].

38. 995. Minimum Number of K Consecutive Bit Flips - [sliding window] [hard]

You are given a binary array nums and an integer k.

A k-bit flip is choosing a subarray of length k from nums and simultaneously changing every 0 in the subarray to 1, and every 1 in the subarray to 0.

Return the minimum number of k-bit flips required so that there is no 0 in the array. If it is not possible, return -1.

O(n)

```
int n = nums.length,ans=0,times=0;
boolean []check = new boolean[nums.length];
for(int i =0;i<n;i++){
    if((nums[i] == 1 && times%2==1) ||
    (nums[i]==0 && times%2==0)){
        times++;
        ans++;
        if(i+k-1 >= n) return -1;
        else{
            check[i+k-1] = true;
        }
    }
    if(check[i]){
        times--;
    }
}
return ans;
```

Less space complexity

```
int n = nums.length,ans=0,times=0;

for(int i =0;i<n;i++){
    if(i>=k){
        if(nums[i-k] > 1){
            nums[i-k] -= 2;
            times--;
        }
    }

    if((nums[i] == 1 && times%2==1) || (nums[i]==0 && times%2==0)){
        if(i+k>n) return -1;
        times++;
        ans++;
        nums[i] += 2;
    }
}
return ans;
```

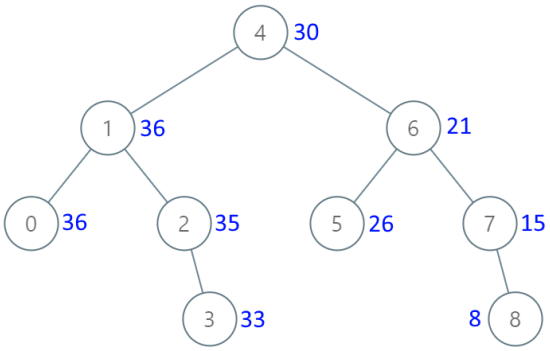
Day - 17 - (25/6/2024)

39. Binary Search Tree to Greater Sum Tree - [Recursion, Trees]

Given the root of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

O(n)

```java
class Solution {
    int sum = 0;
    public TreeNode bstToGst(TreeNode root) {
        if(root != null){
            bstToGst(root.right);
            sum += root.val;
            root.val = sum;
            bstToGst(root.left);
        }
        return root;
```



40. Left Rotate Matrix K times - [Arrays]

You are given an integer k and matrix mat. Rotate the elements of the given matrix to the left k times and return the resulting matrix.

Input: k=1, mat=[[1,2,3],[4,5,6],[7,8,9]]

1 2 3 2 3 1

4 5 6 => 5 6 4

7 8 9 8 9 7

```java
int [][] ans = new int[mat.length][mat[0].length];
k = k%mat[0].length;
for(int i =0;i<mat.length;i++){
    int x = 0;
    for(int j = k;j<mat[0].length;j++){
        ans[i][x] = mat[i][j];
        x++;
    }
    for(int j = 0;j<k;j++){
        ans[i][x] = mat[i][j];
        x++;
    }
}

return ans;
```

Day - 18 - (26/6/2024)

41. Coverage of all Zeros in a Binary Matrix

Given a binary matrix contains 0s and 1s only, we need to find the sum of coverage of all zeros of the matrix where coverage for a particular 0 is defined as a total number of ones around a zero in left, right, up and bottom directions.

O(n*m)

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

            if(j-1>=0 && matrix[i][j-1] == 1 ) sum++;
            if(j+1<m && matrix[i][j+1] == 1) sum++;
            if(i-1 >= 0 && matrix[i-1][j] == 1) sum++;
            if(i+1<n && matrix[i+1][j] == 1) sum++;

        }
    }
}
return sum;
```

42. Balance a Binary Search Tree - [Arrays, Recursion, Graph]

Given the root of a binary search tree, return a balanced binary search tree with the same node values. If there is more than one answer, return any of them.

A binary search tree is balanced if the depth of the two subtrees of every node never differs by more than 1.

Input: root = [1,null,2,null,3,null,4,null,null] Output: [2,1,3,null,null,null,4]

This is not the only correct answer, [3,1,4,null,2] is also correct.

O(nlogn)

```
public TreeNode isbalanced(List<Integer> arr, int start, int end){
    if(start > end) return null;


        int mid = start + (end - start)/2;
        TreeNode node = new TreeNode(arr.get(mid));
        node.left = isbalanced(arr,start,mid-1);
        node.right = isbalanced(arr,mid+1,end);
        return node;
```

```
}
public TreeNode balanceBST(TreeNode root) {
    List<Integer> arr = new ArrayList<>();
    addtree(root,arr);
    return isbalanced(arr,0,arr.size()-1);
}
```

```
public void addtree(TreeNode node,List<Integer> arr){
    if(node == null){
        return;
    }
    addtree(node.left,arr);
    arr.add(node.val);
    addtree(node.right,arr);
}
```

Day - 19 - (27/6/2024)

43. Toeplitz matrix - [Arrays]

A Toeplitz (or diagonal-constant) matrix is a matrix in which each descending diagonal from left to right is constant, i.e., all elements in a diagonal are the same. Given a rectangular matrix mat, your task is to complete the function isToeplitz which returns true if the matrix is Toeplitz otherwise, it returns false.

mat = [[6, 7, 8], Output: true

[4, 6, 7],

[1, 4, 6]]

O(n*m)

```
int n = mat.length;
int m = mat[0].length;
for(int i = 0;i<n-1;i++){
    for(int j = 0;j<m-1;j++){

            if(mat[i+1][j+1]!=mat[i][j]) return false;
    }
}
return true;
```

44. 1791. Find Center of Star Graph - [Arrays, Graphs]

There is an undirected star graph consisting of n nodes labeled from 1 to n. A star graph is a graph where there is one center node and exactly n - 1 edges that connect the center

node with every other node. You are given a 2D integer array edges where each edges[i] = [ui, vi] indicates that there is an edge between the nodes ui and vi. Return the center of the given star graph.

O(1)

```
int a = edges[0][0];
int b = edges[0][1];

return (a == edges[1][0] || a == edges[1][1]) ? a:b;
}
```

Day - 20 - (28/6/2024)

45. Maximum Total Importance of Roads - [Arrays]

You are given an integer n denoting the number of cities in a country. The cities are numbered from 0 to n - 1. You are also given a 2D integer array roads where roads[i] = [ai, bi] denotes that there exists a bidirectional road connecting cities ai and bi. You need to assign each city with an integer value from 1 to n, where each value can only be used once. The importance of a road is then defined as the sum of the values of the two cities it connects.

O(nlogn)

```
long [] arr = new long[n];
for(int [] i : roads){
   arr[i[0]]++;
   arr[i[1]]++;
}

long res = 0,c=1;

Arrays.sort(arr);
for(long i : arr){
   res += i*(c++);
}

return res;
```

O(n)

```
int [] arr = new int[n];
for(int [] i : roads){
   arr[i[0]]++;
   arr[i[1]]++;
}

int cnt [] = new  int[n];
for(int b : arr){
   cnt[b]++;
}

long sum = 0;
long val = 1;
for(long  i =0;i<n;i++){
   for(int j = 0;j<cnt[(int)i];j++){
       sum += i * val++;
   }
}
return sum;
```

46. The Palindrome Pattern - [Arrays] [HARD]

Input:

arr[][] = [[1, 0, 0], [0, 1, 0], [1, 1, 0]] Output: 1 R

Explanation: In the first test case, 0-1-0 is a palindrome

```
public boolean Palinrow(int row, int n, int [][]arr){
    int l = 0;
    int r = n-1;
    while(l<r){
        if(arr[row][l] == arr[row][r]){
            l++;
            r--;
        }
        else  return false;
    }
    return true;
}

public boolean Palincol(int col, int n, int [][]arr){
    int l = 0;
    int r = n-1;
    while(l<r){
        if(arr[l][col] == arr[r][col]){
            l++;
            r--;
        }
        else return false;
    }
    return true;
```

```
public String pattern(int[][] arr) {
    for(int i = 0;i<arr.length;i++){
        if(Palinrow(i,arr.length,arr)) return i + " R";
    }
     for(int i = 0;i<arr.length;i++){
        if(Palincol(i,arr.length,arr)) return i + " C";
    }
    return "-1";
}
```

Day - 21 - (29/6/2024)

47. Identical Linked Lists - [Linked List]

Given the two singly Linked Lists respectively. The task is to check whether two linked lists are identical or not.

```
if(head1 == null || head2 == null){
    if(head1 != head2) return false;

else return true;
}
if(head1.data == head2.data)
return areIdentical(head1.next,head2.next);
else return false;
```

48. 2192. All Ancestors of a Node in a Directed Acyclic Graph - [ArrayList, graphs]

You are given a positive integer n representing the number of nodes of a Directed Acyclic Graph (DAG). The nodes are numbered from 0 to n - 1 (inclusive).

You are also given a 2D integer array edges, where edges[i] = [fromi, toi] denotes that there is a unidirectional edge from fromi to toi in the graph.

Return a list answer, where answer[i] is the list of ancestors of the ith node, sorted in ascending order.

A node u is an ancestor of another node v if u can reach v via a set of edges.

Example 1:

Input: n = 8, edgeList = [[0,3],[0,4],[1,3],[2,4],[2,7],[3,5],[3,6],[3,7],[4,6]]

Output: [[],[],[],[0,1],[0,2],[0,1,3],[0,1,2,3,4],[0,1,2,3]]

O(n)

```java
public List<List<Integer>> getAncestors(int n, int[][] edges) {
    List<List<Integer>> res = new ArrayList<>();
     List<List<Integer>> dc = new ArrayList<>();

   for(int i = 0;i<n;i++){
    res.add(new ArrayList<>());
    dc.add(new ArrayList<>());
   }
   for(int [] e : edges){
    dc.get(e[0]).add(e[1]);
   }

   for(int i = 0;i<n;i++){
    dfs(i,i,res,dc);
   }

   return res;
```

```java
   private void dfs (int x, int cur, List<List<Integer>> ans,
ist<List<Integer>> dc ){

for(int ch : dc.get(cur)) {
    if(ans.get(ch).size() == 0 ||
    ans.get(ch).get(ans.get(ch).size() -1 ) != x){
        ans.get(ch).add(x);
        dfs(x,ch,ans,dc);
    }
```

Day - 22 - (30/6/2024)

49. Delete node in Doubly Linked List - [Linked List]

Given a doubly Linked list and a position. The task is to delete a node from a given position (position starts from 1) in a doubly linked list and return the head of the doubly Linked list.

```
Node temp = head;


if(x==1){
    return head.next;
}
if(x == 2){
    head.next = head.next;

}
if(x==3){
    temp = head.next;
    temp.next = temp.next.next;
    return head;
}

while( x-- > 2  ){
    temp= temp.next;
}
temp.next = temp.next.next;
return head;
```

50. 1579. Remove Max Number of Edges to Keep Graph Fully Traversable

Alice and Bob have an undirected graph of n nodes and three types of edges:

Type 1: Can be traversed by Alice only.

Type 2: Can be traversed by Bob only.

Type 3: Can be traversed by both Alice and Bob.

Given an array edges where edges[i] = [typei, ui, vi] represents a bidirectional edge of type typei between nodes ui and vi, find the maximum number of edges you can remove so that after removing the edges, the graph can still be fully traversed by both Alice and Bob. The graph is fully traversed by Alice and Bob if starting from any node, they can reach all other nodes.

Return the maximum number of edges you can remove, or return -1 if Alice and Bob cannot fully traverse the graph.

```
UnionFind a = new UnionFind(n);
UnionFind b = new UnionFind(n);

int edgesreq = 0;
for(int []e:edges){
    if(e[0] == 3){
        edgesreq+= (a.perf(e[1],e[2]) | b.perf(e[1],e[2]));
    }
}
 for(int []e:edges){
    if(e[0] == 2){
        edgesreq+=  b.perf(e[1],e[2]);
    }
    else if(e[0] == 1){
        edgesreq+= a.perf(e[1],e[2]);
    }
}

if(a.isconnected() && b.isconnected())
return edges.length - edgesreq;
```

```
    return -1;


}

class UnionFind{
    int [] rep;
    int [] comp;
    int conn;

    UnionFind(int n){
        rep = new int[n+1];
        comp = new int[n+1];
        conn = n;

        for(int i = 0;i<=n;i++){
            rep[i] = i;
            comp[i] = i;
        }
```

```
int findRep(int x){
    if(rep[x] == x) return x;
    return rep[x] = findRep(rep[x]);
}

int perf(int x, int y){
    x = findRep(x);
    y = findRep(y);

    if(x == y){
        return 0;
    }
```

```
if(comp[x]>comp[y]){
    comp[x]+= comp[y];
    rep[y] = x;
}
else{
    comp[y]+= comp[x];
    rep[x] = y;
}
conn--;
return 1;
```

```
boolean isconnected(){
    return conn == 1;
}
```

Striver Sheet

1. Merge Sort: O(n)

```java
private ArrayList<Integer> temp = new ArrayList<>();
void merge(int arr[], int l, int m, int r)
{
    ArrayList<Integer> temp = new ArrayList<>();
    int low = l;
    int high = m+1;
    while(low <= m && high <= r){
        if(arr[low] <= arr[high]){
            temp.add(arr[low]);
            low++;
        }
        else{
            temp.add(arr[high]);
            high++;
        }
    }
    while(low <= m){
        temp.add(arr[low++]);
    }
    while(high <= r){
        temp.add(arr[high++]);
    }
```

```java
    for(int i = l;i<=r;i++){
        arr[i] = temp.get(i-l);
    }
    // Your code here

void mergeSort(int arr[], int l, int r)
{
    if(l >= r) return;

    int mid = l + (r-l)/2;

    mergeSort(arr,l,mid);
    mergeSort(arr,mid+1,r);
    merge(arr,l,mid,r);
}
```

2. Bubble Sort - O(n)

```java
    int swap = 0;

    if(n == 1) return;

    for(int j = 0;j<=n-2;j++){
        if(arr[j] > arr[j+1]){
            arr[j] = arr[j]^arr[j+1];
            arr[j+1] = arr[j]^arr[j+1];
            arr[j] = arr[j]^arr[j+1];
            swap = 1;
        }
    }

    if(swap == 0) return;

    bubbleSort(arr,n-1);
}
```

3. Insertion Sort (recursive) - O(n):

```
static void insert(int arr[],int i,int n)
{
    if(i == n) return;
    int min = i;
    while(min > 0 && arr[min - 1] > arr[min]){
        int temp = arr[min];
        arr[min] = arr[min - 1];
        arr[min - 1] = temp;
        min--;
    }

    insert(arr,i+1,n);
     // Your code here
}
//Function to sort the array using insertion sor
public void insertionSort(int arr[], int n)
{
    insert(arr,0,n);
    //code here
```

4. QuickSort - O(n)

```
static void quickSort(int arr[], int low, int high)
{
    if(low >= high) return;

        int pi = partition(arr,low,high);
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);
    }


static int partition(int arr[], int low, int high)
{

    int pivot = arr[high];
    int i = low-1;


    for(int j = low;j<high;j++){
        if(arr[j] < pivot){
            i++;
            swap(arr,i,j);
        }
    }
    swap(arr, i+1,high);
    return i+1;
```

Arrays

5. Second largest element - [easy]

```
if(arr.size() < 2) return -1;
int f = Integer.MIN_VALUE;
int s = Integer.MIN_VALUE;
for(int i : arr){
    if(i > f){
        s = f;
        f = i;
    }
    else if(i > s && f != i){
        s = i;
    }
}
return (s == f)? -1:s;
```

6. Check if Array Is Sorted and Rotated - [Easy]

```
int ans = 0;
for(int i = 0;i<nums.length-1;i++){
    if(nums[i] > nums[i+1]) ans++;
}
if(nums[0] < nums[nums.length-1]) ans++;
return ans <= 1;
```

7. Remove duplicates from sorted array - [Easy]

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

}
return k+1;
```

8. Right Rotate array - [Medium]

Same thing for left rotate array

```
public void reverse( int [] arr, int i, int j){
    while(i<j){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
        j--;
    }
}
public void rotate(int[] nums, int k) {
    k = k % nums.length;
    if(k<0) k = k+nums.length;

    reverse(nums,0,nums.length - k - 1);
    reverse(nums, nums.length - k,nums.length - 1);
    reverse(nums,0,nums.length -1);
```

9. Move Zeroes

```
int i = 0;
int j = 0;

while(j < nums.length){
    if(nums[j]!=0){
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
        i++;
    }
    j++;
}
```

10.Union of Two Sorted Arrays

Given two sorted arrays of size n and m respectively, find their union. The Union of two arrays can be defined as the common and distinct elements in the two arrays. Return the elements in sorted order.

Method - 1 Add all numbers to treeset and convert it to array list

Or

```
// add your code here
ArrayList<Integer> lis = new ArrayList<>();
int i = 0,j=0;
while(i<n || j<m){
    if(i < n && (j>=m || arr1[i] <= arr2[j])){
        if(lis.isEmpty() || lis.get(lis.size() - 1 ) != arr1[i]){
            lis.add(arr1[i]);
        }
        i++;
    }
    else{
        if(lis.isEmpty() || lis.get(lis.size() - 1 ) != arr2[j]){
            lis.add(arr2[j]);
    }
    j++;
}
```

11. Single Number

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.

```
int r = nums[0];
for(int i = 1;i<nums.length;i++){
    r = r^nums[i];
}
return r;
```

12. Longest Sub-Array with Sum K - [Medium]

Given an array arr containing n integers and an integer k. Your task is to find the length of the longest Sub-Array with the sum of the elements equal to the given value k.

```
    HashMap<Integer,Integer> map = new HashMap<>();


    int sum = 0;
    int max = 0;
    for(int i = 0;i<N;i++){
   sum += A[i];
   if(sum == K){
       max = i+1;
   }
   if(map.containsKey(sum - K)){
       max = Math.max(max,i - map.get(sum-K));
   }

   if(!map.containsKey(sum)){
       map.put(sum,i);
   }
}
return max;
```

13. Sort Colors - [Medium]

```
public void sortColors(int[] nums) {
    int low = 0;
    int mid = 0;
    int high = nums.length - 1;
    while(mid <= high){
        if(nums[mid] == 0){
            swap(nums,mid,low);
            mid++;
            low++;
        }
        else if(nums[mid] == 2){
            swap(nums,mid,high);
            high--;
        }
        else{mid++;
```

14. Majority Elements - Moore Voting Algorithm - [easy]

Algorithm goes like, the majority element count is always in the lead compared to the other elements. If the i == candidate element, count is incremented or else decremented. If the count == 0, candidate is changed to the current number(i);

Majority element (>= n/2) retains its count and is saved in the candidate variable.

O(n) space with no additional space initiated.

```
int can = 0;
int count = 0;
for(int i : nums){
    if(count == 0){
        can = i;
    }

    if(i == can){
        count++;
    }
    else{
        count--;
    }
}
return can;
```

Another approach is using a HashMap O(n).

15. Max sum in sub Array - [Easy]

```
// Your code goes here
long sum = 0;
long max = Integer.MIN_VALUE;
for(int i = 0;i<N-1;i++){
    sum = arr[i] + arr[i+1];

    if(sum > max) max = sum;

}
return max;
```

Find the max sum of 2 consecutive numbers in an array

Long - (-2^63) - (2^63 - 1)

Int - (-2^31) - (2^31 - 1)

16. Best time to buy and sell stock - [Easy]

```
int diff = 0;
int i = 0;
int j = 1;
while(j<prices.length){
    if(prices[i] <=  prices[j]){
        diff = Math.max(diff,prices[j] - prices[i]);
        j++;
    }
    else{
        i++;
    }
}
return diff;
```

17. Rearrange Array Elements by Sign - [Medium]

Alternate positive numbers and negatives in an array;

```
int [] ans = new int[nums.length];
int pos = 0;
int neg = 1;

for(int i = 0;i<nums.length;i++){
    if(nums[i] >= 0 ){
        ans[pos] = nums[i];
        pos += 2;
    }
    else{
        ans[neg] = nums[i];
        neg += 2;
    }
```

18. Array Leaders - [Easy]

Given an array arr of n positive integers, your task is to find all the leaders in the array. An element of the array is considered a leader if it is greater than all the elements on its right side or if it is equal to the maximum element on its right side. The rightmost element is always a leader.

Input: n = 6, arr[] = {16,17,4,3,5,2}

Output: 17 5 2

```
 ArrayList<Integer> lead = new ArrayList<>();
 int leader = arr[n-1];
lead.add(leader);

for(int i = n-2;i>=0;i--){
    if(arr[i] >= leader){
        leader = arr[i];
        lead.add(leader);
    }
}

Collections.reverse(lead);
return lead;
```

Stack based approach

```
Stack<Integer> s = new Stack<>();
s.push(arr[n-1]);


for(int i = n-2;i>=0;i--){

    if(arr[i] >= s.peek()){
        s.push(arr[i]);
    }
}

Collections.reverse(s);


return new ArrayList(s);
```

19.Longest Consecutive Sequence - [Medium]

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

Example 1:

Input: nums = [100,4,200,1,3,2] Output: 4

Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

```
Arrays.sort(nums);
int count = 1;
int max = 1;
if(nums.length == 0) return 0;


for(int i = 1;i<nums.length;i++){
    if(nums[i] == nums[i-1]) continue;

    if(nums[i] == nums[i-1]+1){
        count++;

        max = Math.max(max,count);
    }

    else{
        count = 1;
    }
```

20. Set Matrix Zeros

Set all rows and columns which contains 0 as 0;

```
public void setZeroes(int[][] matrix) {
    for(int i =0;i<matrix.length;i++){
     for(int j = 0;j<matrix[0].length;j++){
         if(matrix[i][j] == 0){
             fill(matrix,i,j);
         }
     }

    }
     for(int i =0;i<matrix.length;i++){
     for(int j = 0;j<matrix[0].length;j++){
         if(matrix[i][j] == -111){
             matrix[i][j] = 0;
         }
```

```
public void fill(int [][] arr, int i, int j){
    for(int x = 0;x<arr[0].length;x++){
        if(arr[i][x] == 0) continue;
        arr[i][x] = -111;

    }
    for(int x = 0;x<arr.length;x++){
        if(arr[x][j] == 0) continue;
        arr[x][j] = -111;
    }
```

Another approach

```
int n = matrix.length;
int m = matrix[0].length;
int [] row = new int[n];
int [] col = new int [m];
for(int i = 0;i<n;i++){
    for(int j = 0;j<m;j++){
        if(matrix[i][j] == 0){
            row[i] = 1;
            col[j] = 1; }}}
for(int i = 0;i<n;i++){
    for(int j = 0;j<m;j++){
        if(row[i] == 1 || col[j] == 1)
        matrix[i][j] = 0;
    }
```

21. Rotate Image - [Medium]

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

```
int n = matrix.length;

//transpose the matrix
for(int i =0;i<n;i++){
    for(int j = i;j<n;j++){
        int temp =  matrix[i][j];
        matrix[i][j] = matrix[j][i];
         matrix[j][i] = temp;
    }
}
//reverse each row
for(int k = 0;k<n;k++){
    int i = 0;
    int j = n-1;
    while(i<j){
        int temp = matrix[k][i];
        matrix[k][i] = matrix[k][j];
        matrix[k][j] = temp;
        i++;
        j--;
```

You have to rotate the image in-place, which means you have to modify the

input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

22. Segregate 0's and 1's:

```
int pos = arr.length - 1;
for(int i = arr.length-1;i>=0;i--){
    if(arr[i] == 1){
        int temp = arr[i];
        arr[i] = arr[pos];
        arr[pos] = temp;
        pos--;
    }
}
```

23. Subarray sum equals to K

```
HashMap<Integer,Integer> map = new HashMap<>();
int sum =0;
int count = 0;
map.put(sum,1);
for(int i : nums){
 sum += i;
 if(map.containsKey(sum-k)){
     count += map.get(sum-k);
 }
 map.put(sum, map.getOrDefault(sum,0)+1);
}
return count;
```

24. 229. Majority Element II

Given an integer array of size n, find all elements that appear more than. ⌊ n/3⌋ times.

```
ArrayList<Integer> arr = new ArrayList<>();
int maj1 = 0;
int maj2 = 0;
int c1 = 0;
int c2 = 0;

for(int i : nums){
    if(maj1 == i){
        c1++;
    }
    else if(maj2 == i) c2++;
    else if(c1 == 0){
        maj1 = i;
        c1++;
    }
```

```
if(c1 > nums.length/3) arr.add(maj1);

if(c2 > nums.length/3) arr.add(maj2);

return arr;
```

```
    else if(c2 == 0) {
        maj2 = i;
        c2++;
    }
    else{
        c1--;
        c2--;
    }
}

c1 = 0;
c2 = 0;

for(int i :nums){
    if(i == maj1) c1++;
    else if(i == maj2) c2++;
}
```

25. 3Sum [Medium]

```
ArrayList<List<Integer>> temp = new ArrayList<>();
Arrays.sort(nums);
int c = 0;
for(int i = 0;i<nums.length;i++){
    if(i>0 && nums[i] == nums[i-1]){
        continue;
    }
    int j = i+1;
    int k = nums.length -1;

    while(j<k){
        int sum = nums[i] + nums[j] + nums[k];
        if(sum < 0){
            j++;
        }
        else if(sum > 0){
            k--;
        }
        else{
```

```
}
else{
    temp.add(Arrays.asList(nums[i],nums[j],nums[k]));
    j++;
    k--;
    while(j<k && nums[j]==nums[j-1]) j++;
    while(j<k && nums[k]== nums[k+1]) k--;
}
```

26. Pascal's triangle - [Medium]

```java
public List<Integer> NcR (int row){
    int ans = 1;
    ArrayList<Integer> arr = new ArrayList<>();
    arr.add(1);
    for(int col = 1; col<row;col++){
        ans = ans * (row - col);
        ans = ans/col;
        arr.add(ans);
    }
   return arr;
}
```

```java
public List<List<Integer>> generate(int numRows) {
    ArrayList<List<Integer>> arr = new ArrayList<>();
    for(int i = 1;i<=numRows;i++){
        arr.add(NcR(i));
    }
    return arr;
```

27. 4Sum - [Medium]

```java
if(nums == null || nums.length < 4){
    return new ArrayList<>();
}
Arrays.sort(nums);
HashSet<List<Integer>> arr = new HashSet<>();

for(int i = 0;i<nums.length-3;i++){
    for(int j = i+1;j<nums.length-2;j++){
        int l = j+1;
        int r = nums.length - 1;
```

```
while(l<r){
    long sum = (long) nums[i] + (long) nums[j]+ (long) nums[l] +

    if(sum < target){
        l++;
    }
    else if(sum > target){
        r--;
    }
    else{
        arr.add(Arrays.asList(nums[i],nums[j],nums[l],nums[r]));
        l++;
        r--;
    }
```

Return new ArrayList<>(arr);

28. Jump Game - [Medium]

Return true if you can reach the last index, or false otherwise.

Input: nums = [2,3,1,1,4] Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

```
int target = nums.length - 1;
for(int i = nums.length -1;i>=0;i--){
    if(nums[i] + i >= target){
        target = i;
    }
}
return target == 0;
```

29. Longest Subarray with 0 sum - [Medium]

```
HashMap<Integer,Integer> map = new HashMap<>();
int max = 0;

int sum = 0;
for(int i = 0;i<n;i++){
    sum += arr[i];

    if(sum == 0){
        max = Math.max(max,i+1);
    }
    if(!map.containsKey(sum)){
       map.put(sum,i);
    }
    else{
        max = Math.max(max, i - map.get(sum));
    }


}
return max;
```

30. Merge Sorted Array

```
int i = m-1;
int j = n-1;
int k = m+n - 1;

while(j>=0){
 if(i>=0 && nums1[i] > nums2[j]){
     nums1[k--] = nums1[i--];
 }
 else{
     nums1[k--] = nums2[j--];
 }
```

31. Subarray with given XOR - [Medium]

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

B = 6

Ans = 4

The subarrays having XOR of their elements as 6 are:

[4, 2], [4, 2, 2, 6, 4], [2, 2, 6], [6]

X = xor^k (if x is present, inc count or else add xor in map)

```
HashMap<Integer,Integer> map = new HashMap<>();
map.put(0,1);
int xor = 0;
int count = 0;
int x = 0;
for(int i : A){
    xor = xor^i;
    x = xor^B;
    if(map.containsKey(x)){
        count = count + map.get(x);
    }
    map.put(xor, map.getOrDefault(xor,0)+1);
}
return count;
```

32.453. Minimum Moves to Equal Array Elements - [Medium]

Given an integer array nums of size n, return the minimum number of moves required to make all array elements equal.

In one move, you can increment n - 1 elements of the array by 1.

Input: nums = [1,2,3]

Output: 3

Explanation: Only three moves are needed (remember each move increments two elements):

[1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]

Count the difference from the minimum to the rest and add them.

```
int min = nums[0];
int count = 0;
for(int i : nums){
    if(i < min) min = i;
}
for(int i : nums){
    count += i - min;
}
return count;
```

33. Minimum Moves to Equal Array Elements II - [Medium]

Given an integer array nums of size n, return the minimum number of moves required to make all array elements equal.

In one move, you can increment or decrement an element of the array by 1.

Input: nums = [1,2,3]

Output: 2

Explanation: Only two moves are needed (remember each move increments or decrements one element):

[1,2,3] => [2,2,3] => [2,2,2]

```
Arrays.sort(nums);
int count = 0;
for(int i =0;i<nums.length/2;i++){
    count += nums[nums.length - i - 1] - nums[i];
}
return count;
```

34. Merge Intervals - [Medium]

```
if(intervals.length <=1){
    return intervals;
}
Arrays.sort(intervals, (a,b) -> a[0] - b[0]);

int [] newint = intervals[0];
ArrayList<int []> res = new ArrayList<>();
res.add(newint);
for(int [] i: intervals){
    if(i[0] <= newint[1]){
        newint[1] = Math.max(newint[1], i[1]);
    }

    else{
        newint = i;
        res.add(newint);
    }
}
return res.toArray(new int[res.size()][]);
```

If it was a 2 d array instead of an arraylist, we have to specifically change the value in res.

```
res[0] = newint;
int resIndex = 0;

for (int[] i : intervals) {
    if (i[0] <= newint[1]) {
        newint[1] = Math.max(newint[1], i[1]);
        res[resIndex][1] = newint[1]; // Update
    } else {
        newint = i;
        resIndex++;
        res[resIndex] = newint; // Add new inter
    }
}
```

35. Missing and Repeating number - [Medium]

```
HashSet<Integer> set = new HashSet<>();
long  dup = 0;
long sum = 0;
for(int i : arr){
    if(set.contains(i)){
        dup = (long) i;
        break;
    }
    set.add(i);
}
for(int i : arr){
    sum += i;
}
sum -= dup;
long x = (long) n;
 sum = (x * (x+1)/2) - sum;

return new int[] {(int) dup, (int) sum };
// code here
```

b

```java
long S = (long) n*(n+1)/2;
long P = (long) n *(n+1)*(2*n+1)/6;


long sum = 0;
long sums = 0;

for(int i : arr){
    sum += i;
    sums += (long) i*i;
}

long val1 = S - sum;
long val2 = P - sums;


val2 = val2/val1;


long A = (val1 + val2)/2;
long B = val2 - A;

return new int[]{(int) B, (int) A};
```

36. Count Inversions - [Medium]

```
static long  merge(long arr[], int l, int mid, int r)
    {

        ArrayList<Long > temp = new ArrayList<>();
        int start=l;
        int midStart= mid+1;
        long count=0;

         while(start<=mid && midStart<=r){
             if(arr[start]<=arr[midStart]){
                temp.add(arr[start]);
                start++;

             }else{
                temp.add(arr[midStart]);
                 count += (mid-start+1);
                 midStart++;
             }

         }

         while(start<=mid){
           temp.add(arr[start]);
             start++;
```

```
    while(start<=mid){
      temp.add(arr[start]);
        start++;

    }
    while(midStart<=r){
       temp.add(arr[midStart]);
        midStart++;

    }
    for(int i=l;i<=r;i++){
        arr[i]=temp.get(i-l);
    }
    return count;
```

```
static long mergeSort(long arr[],int l, int r)
    {
        //code here
        long count=0;
        if(l>=r)   return count;
      int mid=(l+r)/2;
        count += mergeSort(arr,l,mid);
         count += mergeSort(arr,mid+1,r);
         count += merge(arr,l,mid,r);
          return count;

    }
static long inversionCount(long arr[], long n) {
        // Your Code Here
        return mergeSort( arr,0,(int)n-1);
```

37. Reverse Pairs - [HARD]

Given an integer array nums, return the number of reverse pairs in the array.

A reverse pair is a pair (i, j) where:

0 <= i < j < nums.length and

nums[i] > 2 * nums[j].

Input: nums = [1,3,2,3,1]

Output: 2

Explanation: The reverse pairs are:

(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1

(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

O(nlogn) - Merge Sort

```
private int mergesort(int [] nums, int left, int right){
    int count = 0;
    if(left >= right) return count;
    int mid = left + (right - left)/2;
    count += mergesort(nums, left, mid);
    count += mergesort(nums, mid+1, right);
    count += countPairs(nums,left,mid,right);
    merge(nums,left,mid,right);
    return count;
}
```

```java
private void merge(int [] arr, int left, int mid, int right){
    ArrayList<Integer> temp = new ArrayList<>();
    int l = left;
    int r = mid+1;
    while(l <= mid && r <= right){
        if(arr[l] <= arr[r]){
            temp.add(arr[l++]);
        }
        else{
            temp.add(arr[r++]);
        }
    }
    while( l<=mid){
        temp.add(arr[l++]);
    }
    while( r<=right){
        temp.add(arr[r++]);
    }
    for(int i = left;i<=right;i++){
        arr[i] = temp.get(i-left);
```

```java
private int countPairs(int [] arr, int left, int mid, int right){
    int count = 0;
    int r = mid + 1;
    for(int i = left;i<=mid;i++){
        while( r <= right && arr[i] > 2* (long) arr[r]){
            r++;
        }
        count += (r - (mid+1));
    }

    return count;
}
```

Return mergesort(nums,0,nums.length-1);

38. Max Product Subarray - [Medium]

```java
double pre = 1;
double suf = 1;
double max = -11;

for(int i =0;i<nums.length;i++){
    if(pre == 0) pre = 1;
    if(suf == 0) suf = 1;

    pre = pre * nums[i];
    suf = suf * nums[nums.length -i -1];

    max = Math.max(max,Math.max(suf,pre));
```

39. Daily Temperatures - [Medium]

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0 instead.

Input: temperatures = [73,74,75,71,69,72,76,73] Output: [1,1,4,2,1,1,0,0]

```
int [] result = new int[temperatures.length];
Stack<Integer> s = new Stack<>();

for(int i = 0;i < temperatures.length;i++){
    while(!s.isEmpty() && temperatures[i] > temperatures[s.peek()]){
        int x = s.pop();
        result[x] = i - x;
    }
    s.push(i);
}
return result;
```

40. Lexicographical Numbers - [Medium]

```
public void dfs(int curr, int n, List<Integer> arr){
if(curr > n) return;

arr.add(curr);
for(int i = 0;i<10;i++){
if(curr * 10 + i > n) return;
dfs(curr * 10 + i, n, arr);
}
}
public List<Integer> lexicalOrder(int n) {
    List<Integer> arr = new ArrayList<>();
    for(int i = 1;i<10;i++){
        dfs(i,n,arr);
    }
    return arr;
```

41. Special Binary String - [hard]

The number of 0's is equal to the number of 1's.

Every prefix of the binary string has at least as many 1's as 0's.

You are given a special binary string s.

A move consists of choosing two consecutive, non-empty, special substrings of s, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.

Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.

Input: s = "11011000" Output: "11100100"

Explanation: The strings "10" [occuring at s[1]] and "1100" [at s[3]] are swapped.

```
if("".equals(s)) return "";

int count = 0;
List<String> arr = new ArrayList<>();
for(int i = 0,j = 0;i<s.length();i++){
    count += (s.charAt(i) == '1') ? 1: -1;

    if(count == 0){
        String t = "1" + makeLargestSpecial(s.substring(j+1,i)) + "0";
        arr.add(t);
        j = i + 1;
    }
}
arr.sort(Comparator.reverseOrder());
return String.join("",arr);
```

42. 1590. Make Sum Divisible by P - Medium

Given an array of positive integers nums, remove the smallest subarray (possibly empty) such that the sum of the remaining elements is divisible by p. It is not allowed to remove the whole array.

Return the length of the smallest subarray that you need to remove, or -1 if it's impossible.

A subarray is defined as a contiguous block of elements in the array.

Input: nums = [3,1,4,2], p = 6 Output: 1

Explanation: The sum of the elements in nums is 10, which is not divisible by 6. We can remove the subarray [4], and the sum of the remaining elements is 6, which is divisible by 6.

```
HashMap<Integer, Integer> map = new HashMap<>();
int sum = 0;
map.put(0,-1);
for(int i : nums){
    sum = (sum + i)%p;
}
if(sum == 0) return 0;
int rem = sum%p;
map.put(0,-1);
sum = 0;
int minn = Integer.MAX_VALUE;
for(int i = 0;i<nums.length;i++){
    sum = (sum + nums[i]) %p;
    sum = ( sum + p) % p;
    int cur_rem = (sum - rem + p)%p;
```

```
    if(map.containsKey(cur_rem)){
        int min_len = i - map.get(cur_rem);
        if(min_len == 1) return 1;
        if(min_len < nums.length)
        minn = Math.min(minn, min_len );
    }
    map.put(sum,i);
}
return (minn == Integer.MAX_VALUE) ? -1: minn;
```

43. Divide Players Into Teams of Equal Skill - Medium

You are given a positive integer array skill of even length n where skill[i] denotes the skill of the ith player. Divide the players into n / 2 teams of size 2 such that the total skill of each team is equal. The chemistry of a team is equal to the product of the skills of the players on that team.

Return the sum of the chemistry of all the teams, or return -1 if there is no way to divide the players into teams such that the total skill of each team is equal.

Input: skill = [3,2,5,1,3,4] Output: 22

Explanation:

Divide the players into the following teams: (1, 5), (2, 4), (3, 3), where each team has a total skill of 6. The sum of the chemistry of all the teams is: 1 * 5 + 2 * 4 + 3 * 3 = 5 + 8 + 9 = 22.

```
Arrays.sort(skill);
int i = 0;
int j = skill.length - 1;
int diff = skill[j] + skill[i];
long count = 0;
while(i < j){
    if(diff == skill[j] + skill[i]){
        count += (skill[j] * skill[i]);
        i++;
        j--;
    }
    else return -1;
}
return count;
```

BINARY SEARCH

1. Binary Search - [Easy]

```
int l = 0;
int h = nums.length - 1;
if(nums.length == 0) return -1;
else if (nums.length == 1 && nums[0] == target) return 0;
while(l<=h){
    int mid = l + (h-l)/2;
    if( nums[mid] == target){
        return mid;
    }
    else if(nums[mid] < target ){
        l = mid + 1;
    }
    else{
        h = mid - 1;
    }
}
return -1;
```

2. Floor in a Sorted Array - [EASY]

Given a sorted array arr[] of size n without duplicates, and given a value x. Floor of x is defined as the largest element k in arr[] such that k is smaller than or equal to x. Find the index of k(0-based indexing).

```
int max = -1;
int l = 0;
int  h = n -1;
while(l<=h){
    int mid = l + (h-l)/2;
    if(arr[mid] == x) return mid;

    if(arr[mid] < x  ){
        max = mid;
        l = mid +1;
    }
    else{
        h = mid - 1;
    }
}
return max;
```

3. Floor and Ceil of X [Easy]

```
Arrays.sort(arr);
// code here
int l = 0;
int h = arr.length - 1;

int max = -1;
int nmax = -1;

while(l<=h){
    int mid = l +(h-l)/2;

    if(arr[mid] == x){
        return new int[] {arr[mid], arr[mid]};
    }
    else if( arr[mid] < x){
        max = arr[mid]; //floor
        l = mid + 1;
    }
    else{
        nmax = arr[mid]; //ceil
        h = mid - 1;
    }
}

return new int [] {max,nmax};
```

4. Partition point - [Easy]

Given an unsorted array of integers. Find an element such that all the elements to its left are smaller and to its right are greater. Print -1 if no such element exists. Note that there can be more than one such element. In that case print the first such number occurring in the array.

Input: N = 7, arr[] = {4, 3, 2, 5, 8, 6, 7} Output: 5

Explanation: To the left of element 5 every element is smaller to it and to the right of element 5 every element is greater to it.

```
long resmax [] = new long[arr.length];
long resmin [] = new long[arr.length];
long max = Integer.MIN_VALUE;
long min = Integer.MAX_VALUE;
for(int i = 0;i<N;i++){
    max = Math.max(max,arr[i]);
    resmax[i] = max;
}
for(int i = N - 1; i>=0;i--){
    min = Math.min(min,arr[i]);
    resmin[i] = min;
}
```

The last if condition states that if the first element or an element greater than its prev max && if the last element or an element lesser than its next element (from the end), then return that element. If its the first element, it

checks with the index of resmin. If its the last element, it checks with the previous index of resmax. Interesting Approach.

5. Find the Closest Element to Zero - [Easy] (Array)

```
public boolean contains(int [] arr, int x){
    for(int i : arr){
        if(i == x) return true;
    }
    return false;
}
public int findClosestNumber(int[] nums) {
    int x = nums[0];
    for(int i : nums){
        if(Math.abs(i) < Math.abs(x)){
            x = i;
        }
    }
    if(x < 0 && contains(nums,Math.abs(x))){
        return Math.abs(x);
    }
    else{
        return x;
```

6.Find First and Last Position of Element in Sorted Array - [Medium]

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

```
public int[] searchRange(int[] nums, int target) {

    if(nums == null ) return new int[] {-1,-1};
    int x = -1;
    int y = -1;
    x = binaryL(nums,target);
    y = binaryH(nums,target);
    return new int[]{x,y};
```

```
while(l<=h){
    int mid = l + (h - l)/2;
    if(arr[mid] == x) {
        ans = mid;
        h = mid -1;
    }
```

```
while(l<=h){
    int mid = l + (h - l)/2;
    if(arr[mid] == x) {
        ans = mid;
        l = mid + 1;
    }
```

7. Search in a Rotated Sorted Array - [Medium]

```
if(nums[low] <= nums[mid]){
    if(nums[low] <= target && target < nums[mid]){
        high = mid - 1;
    }
    else{
        low = mid + 1;
    }
}
else{
    if(nums[mid] < target && target <= nums[high]){
        low = mid + 1;
    }
    else{
        high = mid - 1;
    }
}

turn -1;
```

8. Search in Rotated Sorted Array II - [Medium]

There is an integer array nums sorted in non-decreasing order (not necessarily with distinct values).

Given the array nums after the rotation and an integer target, return true if target is in nums, or false if it is not in nums.

You must decrease the overall operation steps as much as possible.

Input: nums = [2,5,6,0,0,1,2], target = 0 Output: true

```java
int low = 0;
int high = nums.length - 1;
while(low <= high){
    int mid = low + (high - low)/2;
    if(nums[mid] == target) return true;
    else if(nums[mid] == nums[low] && nums[mid] == nums[high]){
        low++;
        high--;
    }
    else if(nums[low] <= nums[mid]){
        if(nums[low] <= target && target < nums[mid]){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }
    else{
        if(nums[mid] < target && target <= nums[high]) low = mid +1;
        else high = mid - 1;
```

9. Minimum Swaps to Group All 1's Together II - [Arrays]

A swap is defined as taking two distinct positions in an array and swapping the values in them. A circular array is defined as an array where we consider the first element and the last element to be adjacent. Given a binary circular array nums, return the minimum number of swaps required to group all 1's present in the array together at any location.

Input: nums = [1,1,0,0,1]

Output: 0

Explanation: All the 1's are already grou ped together due to the circular property of the array.

Thus, the minimum number of swaps required is 0.

```java
int ones = 0;
int n = nums.length;
for(int i : nums) ones += i;
if(ones == 0) return 0;
int a [] = new int[2*n];
for(int i =0;i<2*n;i++){
    a[i] = nums[i%n];
}
int cur = 0;
for(int i = 0;i<ones;i++){
    cur +=nums[i];
}
int max = cur;
for(int i = ones; i<2*n;i++){
    cur += a[i] - a[i-ones];
    max = Math.max(max,cur);
}
return ones - max;
```

10. Find Minimum in Rotated Sorted Array - [Medium]

```java
int low = 0;
int high = nums.length - 1;
int mid = 0;

while(low<=high){
    mid = low + (high - low)/2;

    if(nums[mid] < nums[high]){
        high = mid;
    }
    else{
        low = mid + 1;
    }
}
return nums[mid];
```

11. Find Kth rotation - [Easy]

```
int l = 0;
int h = arr.size() - 1;
int mid = 0;
while(l<=h){
    mid = l + (h-l)/2;

    if(arr.get(mid) < arr.get(h)){
        h = mid;
    }
    else{
        l = mid + 1;
    }
}

return mid;
```

12. Single Element in a Sorted Array - [Medium]

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return the single element that appears only once.

Your solution must run in O(log n) time and O(1) space.

Input: nums = [1,1,2,3,3,4,4,8,8]

Output: 2

Logic - if the mid element's duplicate(next) is in even position or the duplicate(prev) is in odd position , then the single element is on the left side of mid (index). If the mid element's duplicate(next) is in odd position or duplicate(prev) is in even position, the single element in on the right side of the mid (index).

```
  int low = 0;
  int high = arr.length - 1;
  while(low <= high){
      int mid = low + (high - low)/2;

( (mid < arr.length - 1 && arr[mid] == arr[mid+1] && ((mid+1 )%2 == 0) )
      ||
  (mid > 0 && (arr[mid] == arr[mid - 1] && ((mid - 1 )%2 == 1))))){
          high = mid - 1;
      }
se if( (mid > 0 && arr[mid] == arr[mid - 1] && (mid-1)%2 == 0
      ||
      (mid < arr.length - 1 && arr[mid] == arr[mid+1] && (mid+1)%2 == 1))){
          low = mid + 1;
      }
      else{
          return arr[mid];
      }
```

13. Find the peak element - [Medium]

Element greater than its immediate neighbors

```
  int l = 0;
  int h = arr.length - 1;

  while(l<=h){
      int mid = l + (h-l)/2;

      if( mid > 0 && arr[mid - 1] > arr[mid] ){
          h = mid -1;
  }
      else if( mid < arr.length - 1 && arr[mid + 1] > arr[mid])
      {
          l = mid + 1;
      }
      else{
          return mid;
      }
  }
  return -1;
```

14. Square root of a number [Easy] (logn)

Given an integer n, find the square root of n. If n is not a perfect square, then return floor(√n).

Note: Floor value of an integer n is the greatest number less than or equal to n.

```
    long l = 0;
    long h = n;
    long ans = -1;
    while(l<=h){
        long mid = l + (h-l)/2;

        if(mid*mid <= n){
            ans = mid;
            l = mid + 1;
        }
        else{
            h = mid - 1;
        }
    }
    return ans;
```

v

15. Find nth root of M - [Easy]

```
public int func(int n, int m, int x){
    long a = 1;
    for(int i = 0;i<n;i++){
        a = a*x;
        if(a > m) return 2;
    }

    return (a < m) ? 0: 1;
}
public int NthRoot(int n, int m)
{
    // code here
    int l = 1;
    int h = m;

    while(l<=h){
        int mid = l +(h-l)/2;
        int x = func(n,m,mid);
        if(x == 1) return mid;
        else if(x == 2){
            h = mid - 1;
        }
        else{
            l = mid + 1;
        }
```

16. . Koko Eating Bananas - [Medium]

Koko loves to eat bananas. There are n piles of bananas, the ith pile has piles[i] bananas. The guards have gone and will come back in h hours.

Koko can decide her bananas-per-hour eating speed of k. Each hour, she chooses some pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, she eats all of them instead and will not eat any more bananas during this hour.

Return the minimum integer k such that she can eat all the bananas within h hours.

```
int max = Integer.MIN_VALUE;
for(int i : piles){
    max = Math.max(max,i);
}
int l = 1;
int hi = max;
max = Integer.MAX_VALUE;
while(l<=hi){
    int mid = l +(hi-l)/2;
    int x = eathour(piles,h,mid);
    if(x <= h){
        hi = mid-1;
    }
    else{
        l = mid + 1;
    }
}
return l;
```

Do binary search from 1 to max element of the array. If the sum is greater than equals to h, reduce h. Or else increase l and return l.

```
public int eathour(int [] arr, int h, int x){
    int sum = 0;
    for(int i =0;i<arr.length;i++){
        sum += Math.ceil((double)(arr[i]) / (double)(x));
    }
    return sum;
```

17. Min number of Days to make m bouquets - [Medium] - (logn)

```
public int minDays(int[] bloomDay, int m, int k) {
    if (bloomDay.length < (long) m*k){
        return -1;
    }
    int l = 0;
    int h = (int) 1e9;

    while(l<=h){
        int mid = l + (h-l)/2;

        if(ispossible(bloomDay,m,k,mid)){
            h = mid-1;
        }
        else{
            l = mid + 1;
        }
    }
    return l;
```

```
int count = 0;
int ncount = 0;
for(int i = 0;i<arr.length;i++){
    if(arr[i] <= x){
        count++;
    }
    else{
        ncount += (count/k);
        count = 0;
    }
}
ncount += count/k;
return (ncount >= m);
```

18. Find the Smallest Divisor Given a Threshold -[Medium]

Given an array of integers nums and an integer threshold, we will choose a positive integer divisor, divide all the array by it, and sum the division's result. Find the smallest divisor such that the result mentioned above is less than or equal to threshold.

Each result of the division is rounded to the nearest integer greater than or equal to that element. (For example: 7/3 = 3 and 10/2 = 5).

Input: nums = [1,2,5,9], threshold = 6 Output: 5

Explanation: We can get a sum to 17 (1+2+5+9) if the divisor is 1.

If the divisor is 4 we can get a sum of 7 (1+1+2+3) and if the divisor is 5 the sum will be 5 (1+1+1+2).

```
public int smallestDivisor(int[] nums, int threshold) {
    int l = 1;
    int h = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for(int i : nums){
        if(i>h) h = i;
    }
    while(l<=h){
        int mid = l + (h-l)/2;
        int x = func(mid,nums);
        if(x > threshold){
            l = mid + 1;
        }
        else{
            h= mid-1;
        }
    }
   return l;
```

```
public int func(int mid, int [] arr){
    int sum = 0;
    for(int i : arr){
        sum += Math.ceil((double) i / (double) mid);
    }
    return sum;
```

19. Capacity To Ship Packages Within D Days -[Medium]

A conveyor belt has packages that must be shipped from one port to another within days days.

The ith package on the conveyor belt has a weight of weights[i]. Each day, we load the ship with packages on the conveyor belt (in the order given by weights). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within days days.

Input: weights = [1,2,3,4,5,6,7,8,9,10], days = 5

Output: 15

```java
int max = 0;
for(int i : weights){
    if(i>max) max = i;
}
int l = max;
int h = 0;
for(int i : weights) h += i;
while(l<=h){
    int mid = l + (h-l)/2;
    if(func(weights,mid,days)){
        h = mid - 1;
    }
    else{
        l = mid + 1;
```

```java
public boolean func(int arr
    int days = 1;
    int cur = 0;
    for(int i : arr){
        cur += i;
        if(cur > mid){
            days++;
            cur = i;
        }
    }
    return days <= x;
```

20. Kth Missing Positive Number - [Medium]

Given an array arr of positive integers sorted in a strictly increasing order, and an integer k.

Return the kth positive integer that is missing from this array.

Input: arr = [2,3,4,7,11], k = 5 Output: 9

Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9.

```java
int l = 0;
int h = arr.length - 1;
while(l<=h){
    int mid = l + (h-l)/2;
    int missing = arr[mid] - (mid + 1);
    if(missing < k){
        l = mid + 1;
    }
    else{
        h = mid - 1;
    }
}
//return arr[high] + more
// return high + 1 + k;
// low = high + 1;
return l + k;
```

21. Aggressive Cows - [Medium]

You are given an array 'arr' consisting of 'n' integers which denote the position of a stall.

You are also given an integer 'k' which denotes the number of aggressive cows.

You are given the task of assigning stalls to 'k' cows such that the minimum distance between any two of them is the maximum possible.

Print the maximum possible minimum distance.

Sample Input 1 :

6 4

0 3 4 7 10 9 output: 3

The maximum possible minimum distance between any two cows will be 3 when 4 cows are placed at positions {0, 3, 7, 10}. Here distance between cows are 3, 4 and 3 respectively.

(min) max : The minimum distance should be maximum from the set of minimum distances.

```
public static int aggressiveCows(int []stalls, int
    int l = 1;
    Arrays.sort(stalls);
    int h = stalls[stalls.length - 1] - stalls[0];
    while(l<=h){
        int mid = l + (h-l)/2;
        if(calc(stalls,mid,k)){
            l = mid + 1;
        }
        else{
            h = mid - 1;
        }
    }

    return h;
```

```
lic static boolean calc(int [] ar
 int count = 1;
 int last = arr[0];
 for(int i : arr){
     if(i - last >= mid){
         count++;
         last = i;
     }
     if(count >= k) return true;
     }
     return false;
```

Why return h? When low goes equal to h and calc returns false, then h = (mid - 1), which is the (min) max distance.

22. Allocate Books - [Medium]

Given an array ‘arr’ of integer numbers, ‘arr[i]’ represents the number of pages in the ‘i-th’ book. There are ‘m’ number of students, and the task is to allocate all the books to the students.

Allocate books in such a way that:

1. Each student gets at least one book.

2. Each book should be allocated to only one student.

3. Book allocation should be in a contiguous manner.

You have to allocate the book to ‘m’ students such that the maximum number of pages assigned to a student is minimum. If the allocation of books is not possible, return -1.

Input: ‘n’ = 4 ‘m’ = 2

‘arr’ = [12, 34, 67, 90] Output: 113

12 | 34, 67, 90 - the sum of all the pages of books allocated to student 1 is ‘12’, and student two is ‘34+ 67+ 90 = 191’, so the maximum is ‘max(12, 191)= 191’.

12, 34 | 67, 90 - the sum of all the pages of books allocated to student 1 is ‘12+ 34 = 46’, and student two is ‘67+ 90 = 157’, so the maximum is ‘max(46, 157)= 157’.

12, 34, 67 | 90 - the sum of all the pages of books allocated to student 1 is ‘12+ 34 +67 = 113’, and student two is ‘90’, so the maximum is ‘max(113, 90)= 113’.

(max) min

```java
if(m > n) return -1;
int l = Integer.MIN_VALUE;
int h = 0;
for(Integer i : arr) {
    if(i > l) l = i;
    h+= i;
}
while(l<=h){
    int mid = l +(h-l)/2;

    int students = ispossible(arr,mid);

    if(students > m){
        l = mid + 1;
    }
    else{
        h = mid - 1;
    }
}
return l;
```

```java
public static int ispossible(Ar
    int n = arr.size();
    int count = 1;
    long sum = 0;

    for(Integer i : arr){
        if(sum + i <= mid){
            sum += i;
        }
        else{
            count++;
            sum = i;
        }
    }
    return count;
```

23. Split Array Largest Sum & Painter's Partition Problem - [Hard Same as Allocate Books]

Given an integer array nums and an integer k, split nums into k non-empty subarrays such that the largest sum of any subarray is minimized. Return the minimized largest sum of the split. A subarray is a contiguous part of the array.

Input: nums = [7,2,5,10,8], k = 2 Output: 18

```java
public int splitArray(int[] nums, int k)
    int l = Integer.MIN_VALUE;
    int h = 0;
    for(int i : nums){
        if(i>l) l = i;
        h += i;
    }
    while(l<=h){
        int mid = l + (h-l)/2;
        int count = isposs(nums,mid);
        if(count > k){
            l = mid + 1;
        }
        else{
            h = mid - 1;
        }
    }
    return l;
```

```java
public int isposs(int [] arr, int mid){
    int count = 1;
    int sum = 0;
    for(int i : arr){
        if(sum + i <= mid){
            sum += i;
        }
        else{
            sum = i;
            count++;
        }
    }
    return count;
```

24. Median of Two Sorted Arrays (O(log(m+n)) -[Hard]

```
int n1 = nums1.length;
int n2 = nums2.length;
int i = 0;
int j = 0;
int n = (n1 + n2)/2;
int x2 = n;
int x1 = x2 - 1;
int a1 = -1;
int a2 = -1;
int count = 0;
while(i < n1 && j<n2){
    if(nums1[i] < nums2[j]){
        if(count == x1) a1 = nums1[i];
        if(count == x2) a2 = nums1[i];
        count++;
        i++;
    }
```

```
    else{
        if(count == x1) a1 = nums2[j];
        if(count == x2) a2 = nums2[j];
        count++;
        j++;
    }
}
while(i<n1){
    if(count == x1) a1 = nums1[i];
        if(count == x2) a2 = nums1[i];
        count++;
        i++;

}

while(j<n2){
    if(count == x1) a1 = nums2[j];
        if(count == x2) a2 = nums2[j];
        count++;
        j++;
```

```
if((n1+n2)%2 == 1){
    return (double) a2;
}

return (double) ((double) (a1 +a2) / 2.0);
```

25. Row with max 1s - [Easy]

You are given a 2D array consisting of only 1's and 0's, where each row is sorted in non-decreasing order. You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return -1.

Input: arr[][] = [[0, 1, 1, 1],

[0, 0, 1, 1],

[1, 1, 1, 1],

[0, 0, 0, 0]]

Output: 2

Explanation: Row 2 contains 4 1's.

```java
public int lowerbound(int [][]arr, int m, int x){
    int l = 0;
    int h = arr[0].length-1;
    int ans = h+1;

    while(l<=h){
        int mid = l + (h-l)/2;
        if(arr[m][mid] >= x){
            ans = mid;
            h = mid -1;
        }
        else{
            l = mid + 1;
        }
    }
    return ans;
```

```java
public int rowWithMax1s(int arr[][]) {
    int n = arr.length;
    int m = arr[0].length;
    int index = -1;
    int count_ones = 0;

    for(int i = 0; i<n;i++){
        int count = m - lowerbound(arr,i,1);
        if(count > count_ones){
            count_ones = count;
            index = i;
        }
    }
    return index;
```

26. Search a 2D Matrix - [Medium]

Better

```java
public boolean BinarySearch(int [][]arr, int x, int m){
    int l = 0;
    int h = arr[0].length - 1;
    while(l<=h){
        int mid = l + (h-l)/2;
        if(arr[m][mid] == x) return true;
        else if(arr[m][mid] > x) h = mid - 1;
        else  l = mid + 1;
    }
    return false;
}
public boolean searchMatrix(int[][] matrix, int target) {
    int n = matrix.length;
    int m = matrix[0].length;
    int ans = 0;
    for(int i = 0;i<n;i++){
        if(matrix[i][0] <= target && target <= matrix[i][m-1]){
            return BinarySearch(matrix,target,i);
            }
```

Optimal

```java
int n = matrix.length;
int m = matrix[0].length;
int l = 0;
int h = n*m - 1;
while(l<=h){
    int mid = l + (h-l)/2;
    int row = mid/m;
    int col = mid%m;

    if(matrix[row][col] > target){
        h = mid - 1;
    }
    else if(matrix[row][col] == target) return true;
    else{
        l = mid + 1;
```

27. 703. Kth Largest Element in a Stream - [Easy] PQ

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element.

KthLargest(int k, int[] nums) Initializes the object with the integer k and the stream of integers nums.

int add(int val) Appends the integer val to the stream and returns the element representing the kth largest element in the stream.

Input

["KthLargest", "add", "add", "add", "add", "add"]

[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]

Output

[null, 4, 5, 5, 8, 8]

Here k is 3, for every 3 elements and coming ones, you need to return the kth value (sorted). Using a minHeap to store elements in ascending order and whenever a new element that is greater than peek() (lowest element) is added and that element is added and the peek is deleted to maintain the count of 3. INTERESTING!

O(log K)

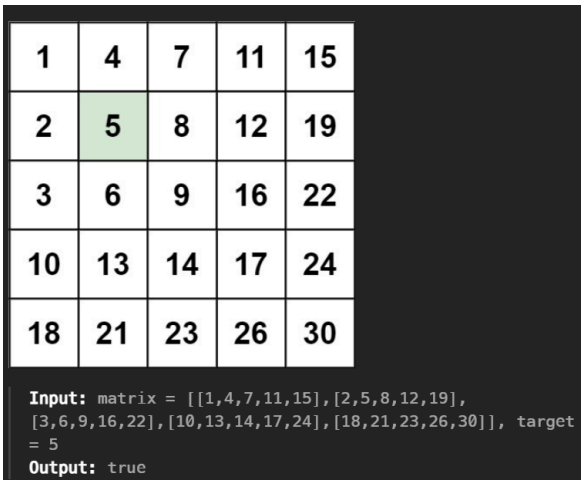
```java
PriorityQueue<Integer> minHeap;
int k;

public KthLargest(int k, int[] nums) {
    this.k = k;
    minHeap = new PriorityQueue<>();
    for(int i : nums){
        add(i);
    } }
public int add(int val) {
    if(minHeap.size() < k || val > minHeap.peek()){
        minHeap.add(val);
        if(minHeap.size() > k){
            minHeap.remove();
        }
    }
    return minHeap.peek();

}
```

28. Search in a Rotated Matrix - II - [Medium]

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.



1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 5
Output: true

L = 0 and h = col -1, if the target < current element, col = col - 1 as the elements in that column are anyways greater, if the target > current element, row = row + 1 as the elements in that row are smaller than target.

```java
public boolean searchMatrix(int[][] matrix, int target) {
    int n = matrix.length;
    int m = matrix[0].length;
    int l = 0;
    int h = m-1;

    while(l < n && h >= 0){
        if(matrix[l][h] == target) return true;

    else if(matrix[l][h] > target){
        h--;
    }
    else l++;
}
return false;
```

29. Peak in a 2d matrix - II - [Medium] - O(log(m))

```java
public int[] findPeakGrid(int[][] mat) {
    int n = mat.length;
    int m = mat[0].length;
    int l = 0;
    int h = m-1;
    while(l<=h){
        int mid = l + (h-l)/2;
        int row = findmax(mat,n,m,mid);
        int left = (mid - 1 >= 0) ? mat[row][mid-1] : -1;
        int right = (mid + 1 < m) ? mat[row][mid+1] : -1;
        if(mat[row][mid] > left && mat[row][mid] > right){
            return new int[] {row,mid};
        }
        else if(mat[row][mid] < left){
            h = mid - 1;
        }
        else l = mid + 1;
    }
    return new int[] {-1,-1};
```

```java
public int findmax(int [][] arr, int n, int m,
    int max = -1;
    int index = 0;
    for(int i = 0;i<n;i++){
        if(arr[i][mid] > max){
            max = arr[i][mid];
            index = i;
        }
    }
    return index;
```

30. Median in a row-wise sorted Matrix - [Hard]

Given a row wise sorted matrix of size R*C where R and C are always odd, find the median of the matrix.

R = 3, C = 3

M = [[1, 3, 5],

[2, 6, 9],

[3, 6, 9]]

Output: 5

Explanation: Sorting matrix elements gives

us {1,2,3,3,5,6,6,9,9}. Hence, 5 is median.

Findcount - gives the number of elements less than or equal to mid.

```
int median(int matrix[][], int R, int C) {
    // code here
    int l = findMin(matrix,R,0);
    int h = findMax(matrix,R,C-1);
    int req = (R*C)/2;

    while(l<=h){
        int mid = l + (h-l)/2;
        int count = FindCount(matrix,R,C,mid);


        if(count <= req ){
            l = mid + 1;
        }
        else  h = mid - 1;

    }
    return l;
```

```
int FindCount(int [][] arr, int r, int c, int x){
    int count = 0;
    for(int i = 0;i<r;i++){
        count += UpperBound(arr[i],c,x);
    }
    return count;
}
int UpperBound(int [] arr ,int c, int x){
    int l = 0;
    int h = c-1;
    int ans = c;
    while(l<=h){
        int mid = l + (h-l)/2;

        if(arr[mid] <= x ){
            l = mid + 1;
        }
        else{
            ans = mid;
         h = mid - 1;
        }
    }
    return ans;
```

STRINGS

1. Remove Outermost Parentheses - [Easy]

```
public String removeOuterParentheses(String s) {
    StringBuilder str = new StringBuilder();
    if(s.length() <= 2) return "";
    int count = 1;
    char c[] = s.toCharArray();

    for(int i = 1;i<c.length;i++){
        if(c[i] == '('){
            count++;
            if(count>1) str.append(c[i]);
        }
        else{
            if(count > 1) str.append(c[i]);
            count--;
        }

    }
    return str.toString();
```

2. Reverse the words in a String - [Medium]

```java
public String reverseWords(String s) {
    String []c = s.trim().split(" ");
    StringBuilder sb = new StringBuilder();

    for(int i = c.length-1;i>=0;i--){
     if(c[i].length() > 0){
         sb.append(c[i] + " ");
     }
    }
    return sb.toString().trim();
}
```

3. Largest Odd Number in String

```java
char []c = num.toCharArray();
int x = c[c.length-1] - '0'; //char to int
if(x%2 == 1) return num;

int max = Integer.MIN_VALUE;
for(int i = c.length - 1;i>=0;i--){
    int y  = c[i] - '0';
    if(y%2 == 1){
        return num.substring(0,i+1);
    }
}
return "";
```

4. Longest Common Prefix

Brute Force

```java
Arrays.sort(strs);
String s = "";
String f = strs[0];
String l = strs[strs.length-1];
for(int i = 0;i<Math.min(f.length(),l.length());i++){
    if(f.charAt(i) != l.charAt(i)){
        return s;
    }
    s += f.charAt(i);
}

return s;
```

Trie !

5. Isomorphic Strings

```java
public boolean isIsomorphic(String s, String t) {
    HashMap<Character,Character> map = new HashMap<>();
    HashSet<Character> set = new HashSet<>();

    for(int i = 0;i<s.length();i++){
        char c = s.charAt(i);
        char d = t.charAt(i);
        if(map.containsKey(c)){
            if(map.get(c) != d) return false;
        }
        else if(set.contains(d)) return false;
        map.put(c,d);
        set.add(d);
    }

    return true;
```

6. Maximum Distance in Arrays - [Medium Arrays]

You are given m arrays, where each array is sorted in ascending order.

You can pick up two integers from two different arrays (each array picks one) and calculate the distance. We define the distance between two integers a and b to be their absolute difference |a - b|.

Return the maximum distance.

Input: arrays = [[1,2,3],[4,5],[1,2,3]]

Output: 4

Explanation: One way to reach the maximum distance 4 is to pick 1 in the first or third array and pick 5 in the second array.

```java
int min = arrays.get(0).get(0);
int max = arrays.get(0).get(arrays.get(0).size() - 1);
int res = 0;

for(int i = 1; i<arrays.size();i++){
    res = Math.max(res,(max - arrays.get(i).get(0)));
    res = Math.max(res,(arrays.get(i).get(arrays.get(i).size() - 1) - min));

    max = Math.max(max,arrays.get(i).get(arrays.get(i).size() - 1));
    min = Math.min(min, arrays.get(i).get(0));
}
return res;
```

7. Rotate String -[Easy]

```java
public boolean rotateString(String s, String goal) {
    if(s.length() != goal.length()) return false;

   return (s+s).contains(goal);
```

8. Best Sightseeing Pair - [Medium]

You are given an integer array values where values[i] represents the value of the ith sightseeing spot. Two sightseeing spots i and j have a distance j - i between them.

The score of a pair (i < j) of sightseeing spots is values[i] + values[j] + i - j: the sum of the values of the sightseeing spots, minus the distance between them.

Return the maximum score of a pair of sightseeing spots.

Input: values = [8,1,5,2,6] Output: 11

Explanation: i = 0, j = 2, values[i] + values[j] + i - j = 8 + 5 + 0 - 2 = 11

```
//Approach - Max Prefix sum
int [] arr1 = new int[values.length];
// arr1 stores max sum from 0 - n;
int n = values.length;
int max = Integer.MIN_VALUE;
for(int i = 0;i<n;i++){
    max = Math.max(max,values[i]+i);
    arr1[i] = max;
}
max = Integer.MIN_VALUE;
for(int i = 1; i<n;i++){
//max calculates the maximum of i-1 in arr1 as i<j
    max = Math.max(max, (arr1[i-1] + (values[i] - i)));
}
return max;
```

9. . Maximum Number of Points with Cost - [Medium]

You are given an m x n integer matrix points (0-indexed). Starting with 0 points, you want to maximize the number of points you can get from the matrix.

To gain points, you must pick one cell in each row. Picking the cell at coordinates (r, c) will add points[r][c] to your score.

However, you will lose points if you pick a cell too far from the cell that you picked in the previous row. For every two adjacent rows r and r + 1 (where 0 <= r < m - 1), picking cells at coordinates (r, c1) and (r + 1, c2) will subtract abs(c1 - c2) from your score.

Return the maximum number of points you can achieve.

Approach - two arrays / loops (one for left and one for right), calculate the value of curr element, Max of curr val or prev val - 1 (distance is increased). And update the dp array.

```
int n = points.length;
int m = points[0].length;
long dp[] = new long[m];
for(int i = 0;i<n;i++){
    for(int j = 0;j<m;j++){
        dp[j] += points[i][j];
    }
    for(int j = 1; j<m;j++){
        dp[j] = Math.max(dp[j], dp[j-1]-1);
    }
    for(int j = m-2;j>=0;j--){
        dp[j] = Math.max(dp[j],dp[j+1]-1);
    }
}
long res = 0;
for(long i : dp){
    if(i > res) res = i;
}
return res;
```

10. Valid Anagram

```
if(s.length() != t.length()) return false;

char [] s1 = s.toCharArray();
char [] t1 = t.toCharArray();
Arrays.sort(s1);
Arrays.sort(t1);
String x = new String(s1);
return x.equals(new String(t1));
    }
```

11. Sort Characters By Frequency - [Medium]

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.

Input: s = "tree"

Output: "eert"

```
HashMap<Character,Integer> map = new HashMap<>();
for(int i = 0; i<s.length();i++){
    map.put(s.charAt(i),map.getOrDefault(s.charAt(i),1)+1);
}
StringBuilder s1 = new StringBuilder();
while(!map.isEmpty()){
    char maxc = 'a';
    int max = 0;
    for(Map.Entry<Character,Integer> m : map.entrySet()){
        if(m.getValue() > max){
            max = m.getValue();
            maxc = m.getKey();
        }
    }
    for(int i = 1;i<max;i++) s1.append(maxc);
    map.remove(maxc);
}
return s1.toString();
```

Or

```
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<Pair> pq = new PriorityQueue<>((a,b) -> b.val - a.val);
for(char c : map.keySet())
pq.offer(new Pair(c,map.get(c)));
s = "";
while(!pq.isEmpty()){
    Pair p = pq.poll();

    for(int i = 0;i<p.val;i++) s += p.ch;

    pq.remove(p);
}
return s;
```

```
class Pair{
    int val;
    char ch;

    public Pair(char ch, int val){
        this.val = val;
        this.ch = ch;
    }
```

Or

```
char strs[] = s.toCharArray();
int freq[] = new int[128];
for(int i = 0;i<strs.length;i++){
    freq[strs[i]]++;
}
for(int i = 0;i<strs.length;){
    char cmax = ',';
    for(int j =0;j<128;j++){
        if(freq[j] > freq[cmax]) cmax = (char)j;
    }
    while(freq[cmax]>0){
        strs[i++] = cmax;
        freq[cmax]--;
    }
}
return new String(strs);
```

12. Maximum Nesting Depth of the Parentheses - [Easy]

Given a valid parentheses string s, return the nesting depth of s. The nesting depth is the maximum number of nested parentheses.

Input: s = "(1+(2*3)+((8)/4))+1"

Output: 3

```
public int maxDepth(String s) {
    int count = 0;
    int max = 0;
    for(char c : s.toCharArray()){
        if(c == '('){
            count++;
            max = Math.max(max,count);
        }
        else if(c == ')'){
            count--;
        }
    }
    return max;
```

13. String to Integer (ATOI) - {MEDIUM}

```
long res = 0;
int sign = 1;
int n = s.length();
int i = 0;
s = s.trim();
if(i<n && (s.charAt(i) == '-' || s.charAt(i) == '+')){
    sign = (s.charAt(i) == '-') ? -1 : 1;
    i++;
}
while(i<n && Character.isDigit(s.charAt(i))){
    res = res * 10 + (s.charAt(i) - '0');

    if(res * sign < Integer.MIN_VALUE){
        return Integer.MIN_VALUE;
    }
    else if(res * sign > Integer.MAX_VALUE)
    return Integer.MAX_VALUE;
    i++;
}
return (int) res*sign;
```

14. Count number of substrings - [Medium]

Given a string of lowercase alphabets, count all possible substrings (not necessarily distinct) that have exactly k distinct characters.

Input: S = "aba", K = 2

Output:3

Explanation:The substrings are: "ab", "ba" and "aba".

Take a map array[26], iterate over the array, increment the count of characters, increment the count of distinct (1st occurrence) character in a separate variable. If the distinct count variable > K, decrement the count of the jth character and if the the count of jth character == 0, decrement the distinct count variable (it should be <= K). count the value (i-j + 1) for K and K-1.

```java
long countsub(String S, int k){
    int i = 0;
    int j = 0;
    int d_count = 0;
    long count = 0;
    int []freq = new int[26];

    for(i = 0;i<S.length();i++){
        freq[S.charAt(i) - 'a']++;

        if(freq[S.charAt(i) - 'a'] == 1) d_count++;

        while(d_count > k){
            freq[S.charAt(j) - 'a']--;
            if(freq[S.charAt(j) - 'a'] == 0) d_count--;
            j++;
        }
        count += (i-j)+1;

    }
    return count;
}
long substrCount (String S, int K) {
    return countsub(S,K) - countsub(S,K-1);
```

15. Longest Palindromic Substring

Input: s = "babad"

Output: "bab"

```java
public String longestPalindrome(String s) {
    String res = "";
    for(int i = 0;i<s.length();i++){
        for(int j = i;j<s.length();j++){
            String x = s.substring(i,j+1);
            if(checkpa(x) && x.length() > res.length()){
                res = x;
            }
        }
    }
    return res;
```

```java
public Boolean checkpa(String s){
    int i = 0;
    int j = s.length() - 1;
    while(i<=j){
        if(s.charAt(i) == s.charAt(j)){
            i++;
            j--;
        }
        else{
            return false;
        }
    }
    return true;
```

Dynamic -

```java
public String longestPalindrome(String s) {
    int s1 = 0;
    int e = 0;
    int maxlen = 1;
    boolean [][] dp = new boolean[s.length()][s.length()];
    for(int i = 0;i<s.length();i++){
        dp[i][i] = true;
        for(int j = 0;j<s.length();j++){
            if(s.charAt(i) == s.charAt(j) && (i - j <= 2 || dp[j+1][i-1])){
                dp[j][i] = true;
                if(i - j + 1 > maxlen){
                    maxlen = i-j+1;
                    s1 = j;
                    e = i;
                }
            }
        }
    }
    return s.substring(s1,e+1);
```

16. Number Complement - [Easy]

The complement of an integer is the integer you get when you flip all the 0's to 1's and all the 1's to 0's in its binary representation. For example, The integer 5 is "101" in binary and its complement is "010" which is the integer 2. Given an integer num, return its complement.

Input: num = 5 Output: 2

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

```java
public boolean checknum(int idx, int num){
    return ((1 << idx) & num) != 0;
}
public int findComplement(int num) {
    int highest = 30;
    for(;highest>=0;highest--){
        if(checknum(highest,num)) break;
    }
    highest++;

    int mask = (1 << highest) - 1;
    return num ^ mask;
```

or

```java
public int findComplement(int num) {
    if(num == 0) return 1;
    int len = num.toBinaryString().length();
    int max = (1 << len) - 1;
    return num ^ max;
}
```

17. . Sum of Beauty of All Substrings - [Medium]

The beauty of a string is the difference in frequencies between the most frequent and least frequent characters.

Given a string s, return the sum of beauty of all of its substrings.

Input: s = "aabcb"

Output: 5

Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bcb"], each with beauty equal to 1.

Input: s = "aabcbaa"

Output: 17

Brute Force O(n^3)

```
int count = 0;
for(int i = 0;i<s.length();i++){
    int [] map = new int[26];
    int max = 0;
    int min = Integer.MAX_VALUE;
    for(int j = i;j<s.length();j++){
        map[s.charAt(j) - 'a']++;
        max = Math.max(max,map[s.charAt(j) - 'a']);
        min = Integer.MAX_VALUE;
    for(int x : map){
        if(x>0) min = Math.min(min,x);
    }
    count += max - min;
}
}
return count;
```

18. Total count

You are given an array arr[] of positive integers and a threshold value k. For each element in the array, divide it into the minimum number of small integers such that each divided integer is less than or equal to k. Compute the total number of these integer across all elements of the array.

Input: k = 3, arr[] = [5, 8, 10, 13] Output: 14

Explanation: Each number can be expressed as sum of different numbers less than or equal to k as 5 (3 + 2), 8 (3 + 3 + 2), 10 (3 + 3 + 3 + 1), 13 (3 + 3 + 3 + 3 + 1). So, the sum of count of each element is (2+3+4+5)=14.

```java
class Solution {
    int totalCount(int k, int[] arr) {
        // code here
        int count = 0;
        for(int i : arr){
            if(i%k == 0) count += i/k;

        else{
            count += i/k + 1;
        }
        }
        return count;
    }
}
```

19. 567. Permutation in String - [Medium]

Given two strings s1 and s2, return true if s2 contains a Permutation of s1, or false otherwise. In other words, return true if one of s1's permutations is the substring of s2.

Input: s1 = "ab", s2 = "eidbaooo" Output: true

Explanation: s2 contains one permutation of s1 ("ba").

```java
public boolean checkInclusion(String s1, String s2) {
    if(s1.length() > s2.length()) return false;
    char ch1 [] = new char[26];
    for(char c : s1.toCharArray()){
        ch1[c - 'a']++;
    }
    int idx;
    char ch2 [] = new char[26];
    for(idx = 0; idx < s1.length();idx++){
        ch2[s2.charAt(idx) - 'a']++;
    }
    while(idx < s2.length()){
        if(match(ch1,ch2)) return true;
        ch2[s2.charAt(idx) - 'a']++;
        ch2[s2.charAt(idx - s1.length()) - 'a']--;
        idx++;
    }
    return match(ch1,ch2);
```

```java
public boolean match(char [] ch1, char [] ch2){
    for(int i = 0;i<ch1.length; i++){
        if(ch1[i] != ch2[i]) return false;
    }
    return true;
}
```

20. Sentence Similarity III - [Medium]

You are given two strings sentence1 and sentence2, each representing a sentence composed of words. A sentence is a list of words that are separated by a single space

with no leading or trailing spaces. Each word consists of only uppercase and lowercase English characters.

Two sentences s1 and s2 are considered similar if it is possible to insert an arbitrary sentence (possibly empty) inside one of these sentences such that the two sentences become equal. Note that the inserted sentence must be separated from existing words by spaces.

s1 = "Hello Jane" and s2 = "Hello my name is Jane" can be made equal by inserting "my name is" between "Hello" and "Jane" in s1.

```
String s1 [] = sentence1.split(" ");
String s2 [] = sentence2.split(" ");
int i = 0;
int j = 0;
while(i < s1.length && j < s2.length){
    if(!s1[i].equals(s2[j])) break;
    i++;
    j++;
}
if( i == s1.length || j == s2.length) return true;
int l1 = s1.length - 1;
int l2 = s2.length - 1;
while(l1 >= i && l2 >= j){
    if(!s1[l1].equals(s2[l2])){
        return false;
    }
    l1--;
    l2--;
}
return l1 < i || l2 < j;
```

Linked List

1. Array to Linked List - [Easy]

```
static Node constructLL(int arr[]) {
    // code here
    Node head = new Node(0);
    Node temp = head;
    for(int i : arr){
        temp.next = new Node(i);
        temp = temp.next;
    }
    return head.next;
}
```

2. Insert val at the end of LinkedList - [Easy]

```
Node insertAtEnd(Node head, int x) {
    // code here
    if(head == null) return new Node(x);
    Node temp = head;
    while(temp.next!=null){
        temp = temp.next;
    }
    temp.next = new Node(x);
    return head;
```

3. Length of LL

```
public int getCount(Node head) {
    // code here
    int count = 0;
    while(head!=null){
        count++;
        head = head.next;
    }
    return count;
}
```

4. Delete node without giving head

Node.val = node.next.val

Node.next = node.next.next;

5. Construct doubly Linked List

```
Node head = new Node(0);
Node temp = head;
Node tempp = head;
for(int i = 0;i<arr.length;i++){
    temp.next = new Node(arr[i]);
    tempp = temp;
    if(i == 0){
        tempp = null;
    }
    temp = temp.next;
    temp.prev = tempp;
}
return head.next;
```

```
    if(ind >= arr.length) return null;
    Node temp = new Node(arr[ind]);
    temp.next = recursive(arr,ind+1);
    if(temp.next != null){
        temp.next.prev = temp;
    }
    return temp;

ode constructDLL(int arr[]) {
    return recursive(arr,0);
```

6. Insert a node in DLL

```
Node temp = head_ref;
Node tempp = head_ref;
int i = 0;
while(i<pos){
    temp = temp.next;
    i++;
}
tempp = temp.next;
temp.next = new Node(data);
temp.next.prev = temp;
temp = temp.next;
temp.next = tempp;
```

7. Delete a node in DLL

```
Node temp = head;
Node tempp = head;
int count = 0;
int i = 1;
if(x == 1){
    return head.next;
}
while(tempp!= null){  //count length
    count++;
    tempp = tempp.next;
}
while(i < x-1){
    temp = temp.next;
    i++;
}
if(count == x){
    temp.next = null;
    return head;
}
temp.next = temp.next.next;
temp.next.prev = temp;
return head;
```

8. Reverse a DLL

```
public DLLNode reverseDLL(DLLNode head) {
    DLLNode curr = head;
    DLLNode n = head;
   head.prev = null;
   while(curr.next != null){
        n = n.next;
        curr.next = curr.prev;
        curr = n;
   }
   curr.next = curr.prev;
   return curr;
```

9. Middle of LL

```
public ListNode middleNode(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    //tortoise & hare approach
    while(fast!= null && fast.next != null){
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
```

10. Reverse a LL

```
ListNode curr = head;
ListNode n = head;
ListNode prev = null;
while(curr != null){
    n = n.next;
    curr.next = prev;
    prev = curr;
    curr = n;
}
return prev;
```

11. Reverse a LL (recursion)

```
public ListNode reverseList(ListNode head) {
    if(head == null || head.next == null){
     return head;
    }
    ListNode newHead = reverseList(head.next);
//     ListNode nexthead = head.next;
//     nexthead.next = head;
    head.next.next = head;
    head.next = null;
    return newHead;
```

12. N - ary post Order tree traversal

```
List<Integer> ans;
public void postTrav(List<Integer> ans , Node root){
if(root == null) return;
for(Node x : root.children){
    postTrav(ans,x);
}
ans.add(root.val);

public List<Integer> postorder(Node root) {
    ans = new ArrayList<>();
    postTrav(ans,root);
    return ans;
}
```

13. LinkedList Cycle

```
public boolean hasCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    if(head == null) return false;
    while(fast != null && fast.next != null ){

        fast = fast.next.next;
        slow = slow.next;
        if(fast == slow) return true;
    }
    return false;
```

14. Beginning of the first node of the loop

```java
public ListNode detectCycle(ListNode head) {
    ListNode fast = head;
    ListNode start = head;
    ListNode slow = head;
    while(fast!=null && fast.next!= null){
        fast = fast.next.next;
        slow = slow.next;
       if(slow == fast){
        while(slow != start){
            slow = slow.next;
            start = start.next;
        }
        return start;
       }
        }
    return null;
```

15. Length of the Loop

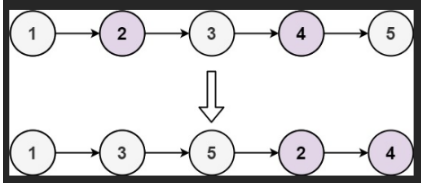
```java
  int count = 0;
  boolean isstart = false;
  Node slow = head;
  Node fast = head;
  Node start = head;
  while(fast!=null && fast.next!=null){
      fast = fast.next.next;
      slow = slow.next;

      if(slow == fast){
          while(start != slow){
              start = start.next;
              slow = slow.next;
          }
          isstart = true;
          break;
      }
  }
  if(isstart){
      start = start.next;
      count = 1;
      while(start != slow){
          start = start.next;
          count++;
      }
  }
  return count;
```

16. Palindrome Linked List

```
 if(head == null || head.next == null) return true;
ListNode fast = head;
ListNode slow = head;
ListNode temp = head;
while(fast!= null && fast.next!= null){
 fast = fast.next.next;
 slow = slow.next;
}
ListNode reversed = recrev(slow);
ListNode rev = reversed;
while(rev!= null){
 if(temp.val != rev.val){
     recrev(reversed);
     return false;
 }
 rev = rev.next;
 temp = temp.next;
}
recrev(reversed);
return true;
```

17. Odd Even LL

Example 1:



Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]

```
ublic ListNode oddEvenList(ListNode head) {
   if(head == null) return null;
   if(head.next == null) return head;
  ListNode odd = head;
  ListNode even = head.next;
  ListNode eh = even;
  while(even!=null && even.next!=null){
   odd.next = even.next;
   odd = odd.next;
   even.next = odd.next;
   even = even.next;
  }
  odd.next = eh;
  return head;
```

18. Count Sub Islands - [Medium]

You are given two m x n binary matrices grid1 and grid2 containing only 0's (representing water) and 1's (representing land). An island is a group of 1's connected 4-directionally (horizontal or vertical). Any cells outside of the grid are considered water cells.

An island in grid2 is considered a sub-island if there is an island in grid1 that contains all the cells that make up this island in grid2. Return the number of islands in grid2 that are considered sub-islands.

```
public int countSubIslands(int[][] grid1, int[][] grid2) {
    boolean vis[][] = new boolean[grid1.length][grid1[0].length];
    int ans = 0;
    for(int i = 0;i<grid1.length;i++){
        for(int j = 0;j<grid1[0].length;j++){
            if(grid2[i][j] == 1 && !vis[i][j]){
                subTree = true;
                dfs(grid1,grid2,i,j,vis);
                if(subTree) ans++;
            }
        }
    }
    return ans;
```

```
boolean subTree = false;
    public void dfs(int [][] grid1, int [][] grid2,
    int x, int y, boolean[][]vis){
    if(grid1[x][y] == 0) subTree = false;
    vis[x][y] = true;
    int xA[] = {-1,0,0,1};
    int yA[] = {0,-1,1,0};
    for(int i = 0;i<xA.length;i++){
        int newx = x + xA[i];
        int newy = y + yA[i];
        if(newx < 0 || newx >=grid1.length || newy <0 ||
        newy >= grid1[0].length || grid2[newx][newy] == 0)
        continue;
        if(grid2[newx][newy] == 1 && !vis[newx][newy])
        dfs(grid1,grid2,newx,newy,vis);
```

19. Remove Nth Node From End of List - [Medium]

Given the head of a linked list, remove the nth node from the end of the list and return its head.

```
public ListNode removeNthFromEnd(ListN
    ListNode slow = new ListNode(0);
    ListNode s = head;
    ListNode f = head;
    slow.next = head;
    for(int i = 1;i<=n;i++){
        f = f.next;
    }
    if(f == null) {
        head = head.next;
        return head;
    }
    while(f.next!=null){
        s = s.next;
        f = f.next;
    }
    s.next = s.next.next;
    return slow.next;
```

20. Delete Middle Node - [medium]

```
ListNode fast = head;
ListNode s = new ListNode(0);
ListNode ss = s;
s.next = head;
while(fast!=null && fast.next!=null){
    fast = fast.next.next;
    ss = ss.next;
}
ss.next = ss.next.next;
return s.next;
```

21. Sort LL - Merge - Sort

```
public ListNode sortList(ListNode head) {
    if(head == null || head.next == null) return head;

    ListNode mid = MiddleLL(head);
    ListNode midnext = mid.next;
    mid.next = null;

    ListNode left = sortList(head);
    ListNode right = sortList(midnext);

    return merge(left,right);
```

```
ublic ListNode MiddleLL(ListNode head){
    ListNode slow = head;
    ListNode fast = head;
    while(fast.next!=null && fast.next.next!=null){
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
```

```
ListNode dummy = new ListNode(-1);
ListNode xx = dummy;
while(temp1!=null && temp2!=null){
    if(temp1.val < temp2.val){
        xx.next = temp1;
        xx = xx.next;
        temp1 = temp1.next;
    }
    else{
        xx.next = temp2;
        xx = xx.next;
        temp2 = temp2.next;
    }
}
    if(temp1!=null){
        xx.next = temp1;
    }
    if(temp2!=null)
        xx.next = temp2;

return dummy.next;
```

22. Sort a linked list of 0s, 1s and 2s - [Easy]

```
int [] count = new int[3];
Node temp = head;
while(temp!=null){
    count[temp.data]++;
    temp = temp.next;
}
int i = 0;
temp = head;
while(temp!=null){
    if(count[i] == 0){
        i++;
    }
      else{
        temp.data = i;
        count[i]--;
        temp = temp.next;
      }
}
return head;
```

23. Intersection of Two Linked Lists

```
ListNode t1 = headA;
ListNode t2 = headB;
int ac = 0;
int bc = 0;
while(t1!=null){
    ac++;
    t1 = t1.next;
}
while(t2!=null){
    bc++;
    t2 = t2.next;
}
t1 = headA;
t2=headB;
while(ac > bc){
    ac--;
    t1 = t1.next;
}
```

```
while(ac < bc){
    bc--;
    t2 = t2.next;
}
while(t1 != t2){
    t1= t1.next;
    t2 = t2.next;
}
return t1;
```

24. Add 1 to a Linked List Number

```
 Node c = head;
 Node prev = null;
 Node n = head;

 while(c!=null){
     n = n.next;
     c.next = prev;
     prev = c;
     c = n;
 }
 head = prev;
Node temp = head;
if(temp.data + 1 < 9){
    temp.data += 1;
}
else{
    while(temp.data + 1 > 9 && temp.next!= null){
        temp.data = 0;
        temp = temp.next;
    }

    if(temp.next != null){
        temp.data += 1;
    }
    else{
        temp.next = new Node(1);
        temp.data = 0;
```

```
    }
}
 c = head;
 prev = null;
 n = head;

 while(c!=null){
     n = n.next;
     c.next = prev
     prev = c;
     c = n;
 }
 head = prev;
 return head;
```

25. Most Stones Removed with Same Row or Column - [Medium]

On a 2D plane, we place n stones at some integer coordinate points. Each coordinate point may have at most one stone. A stone can be removed if it shares either the same row or the same column as another stone that has not been removed. Given an array stones of length n where stones[i] = [xi, yi] represents the location of the ith stone, return the largest possible number of stones that can be removed.

Input: stones = [[0,0],[0,1],[1,0],[1,2],[2,1],[2,2]] Output: 5

```
public int removeStones(int[][] stones) {
    List<List<Integer>> arr = newAList(stones);
    boolean vis[] = new boolean[stones.length];
    int number = 0;
    for(int i = 0;i<vis.length;i++){
        if(!vis[i]){
            dfs(arr,stones,vis,i);
            number++;
        }
    }
    return stones.length - number;
```

```
public List<List<Integer>> newAList(int[][] stones){
    List<List<Integer>> arr = new ArrayList<>();
    for(int i = 0;i<stones.length;i++){
        arr.add(new ArrayList<>());
    }
    for(int i = 0;i<stones.length;i++){
        for(int j = i+1;j<stones.length;j++){
    if(stones[i][0] == stones[j][0] || stones[i][1] == stones[j][1]){
        arr.get(i).add(j);
        arr.get(j).add(i);
    }
        }
    }
    return arr;
}
```

```java
public void dfs(List<List<Integer>> arr, int [][] stones,
boolean [] vis, int stone){
    vis[stone] = true;
    for(int n : arr.get(stone)){
        if(!vis[n]){
            dfs(arr,stones,vis,n);
        }
```

26. Add two numbers - [Medium]

```java
ListNode head = new ListNode(0), p = head;
int carry = 0;

while(l1!=null || l2!=null || carry==1){

    int sum = carry;

    if(l1!=null) {
        sum+=l1.val;
        l1 = l1.next;
    }


    if(l2!=null){
        sum+=l2.val;
        l2 = l2.next;
    }
```

```java
    p.next = new ListNode(sum%10);
    p = p.next;
    carry = sum/10;
}


return head.next;
```

27. Remove duplicates from a sorted doubly linked list

```java
if(head == null) return null;

Node temp = head;
while(temp.next !=null){
    if(temp.data == temp.next.data){
        Node nex = temp.next.next;
        temp.next = nex;
        if(nex !=null){
            nex.prev = temp;
        }
    }
    else{
        temp = temp.next;
    }
}
return head;
```

28. Reverse Nodes in k-Group - [HARD]

```java
public ListNode reverseKGroup(ListNode head, int k) {
    ListNode temp = head;
    ListNode prev = null;
    while(temp!=null){
        ListNode kthnode = findK(temp,k);

        if(kthnode == null){
            if(prev != null) prev.next = temp;
            break;
        }
         ListNode nextn = kthnode.next;
        kthnode.next = null;
        recrev(temp);
        if(temp == head){
            head = kthnode;
        }
        else{
            prev.next = kthnode;
        }
        prev = temp;
        temp = nextn;
    }
    return head;
```

```java
public ListNode recrev(ListNode head){
    ListNode temp = head;
    ListNode prevL = null;
    while(temp!=null){
        ListNode first = temp.next;
        temp.next = prevL;
        prevL = temp;
        temp = first;
    }
    return prevL;
}
public ListNode findK(ListNode temp, int k){
    k -= 1;
    while(temp !=null && k > 0){
        k--;
        temp = temp.next;
    }
    return temp;
```

29. Delete Nodes From Linked List Present in Array - [Medium]

```java
HashSet<Integer> set = new HashSet<>();
for(int i : nums){
    set.add(i);
}
ListNode temp = new ListNode(0);
temp.next = head;
ListNode x = temp;
while(temp.next!=null){
    if(set.contains(temp.next.val)){
        temp.next = temp.next.next;
    }
    else temp = temp.next;
}
return x.next;
```

30. Rotate List - Medium

```java
if(head == null ) return null;
ListNode temp = head;
int count = 1;
while(temp.next!=null){
    count++;
    temp = temp.next;
}
k = k % count;
while(k > 0){
    ListNode x = findn(head,temp);
    temp.next = head;
    head = temp;
    temp = x;
    x.next = null;
    k--;
}
return head;
```

```java
public ListNode findn(ListNode head, L
    ListNode x = head;
    while(x.next!=temp){
        x = x.next;
    }
    return x;
}
```

31. Palindrome Linked List - [Medium]

Given a singly linked list of integers. The task is to check if the given linked list is palindrome or not.

Input: LinkedList: 1->2->1->1->2->1 Output: true

```
boolean isPalindrome(Node head) {
    // Your code here
    Node fast = head;
    Node slow = head;
    int count = 0;

    while(fast!=null && fast.next!=null){
        fast = fast.next.next;
        slow = slow.next;
    }

    slow = rev(slow,fast);
    fast = head;
    while(slow!=null){
        if(slow.data != fast.data) return false;

        slow = slow.next;
        fast = fast.next;
    }
    return true;
}
```

32. K Sized Subarray Maximum - [Medium Sliding Window]

Given an array arr[] and an integer k. Find the maximum for each and every contiguous subarray of size k.

Input: k = 3, arr[] = [1, 2, 3, 1, 4, 5, 2, 3, 6] Output: [3, 3, 4, 5, 5, 5, 6]

```
public ArrayList<Integer> max_of_subarrays(int k, int arr[]) {
    // Your code here
    ArrayList<Integer> nums = new ArrayList<>();
    Deque<Integer> dq = new ArrayDeque<>();
    for(int i = 0;i<arr.length;i++){

      if(dq.size() != 0 && dq.getFirst() == i-k) dq.removeFirst();

      while(!dq.isEmpty() && arr[dq.getLast()] <= arr[i]) dq.removeLast();
      dq.add(i);

      if(i >= k-1 ) nums.add(arr[dq.getFirst()]);
    }
    return nums;
}
```

33.Check If Array Pairs Are Divisible by k - [Medium]

```
    int count[] = new int[k];
    for(int i : arr){
        int cnt = i%k;
        if(cnt < 0) cnt += k;
        count[cnt]++;
    }

    for(int i =1;i<k;i++){
        if(count[i] != count[k-i]) return false;
    }
    return count[0]%2==0;
```

34. Find the number of islands - [Medium]

Given a grid of size n*m (n is the number of rows and m is the number of columns in the grid) consisting of '0's (Water) and '1's(Land). Find the number of islands.

```
public void ispossible(int row, int col, char [][] grid){
if(row >= grid.length || col >= grid[0].length || row<0 || col < 0
|| grid[row][col] != '1') return;
        grid[row][col] = '0';

    int[] rNbr = {1, -1, 0, 0, 1, -1, 1, -1};
    int[] cNbr = {0, 0, 1, -1, 1, -1, -1, 1};

    for (int i = 0; i < 8; ++i) {
        int newR = row + rNbr[i];
        int newC = col +  cNbr[i];
        ispossible(newR, newC,grid);
    }
    }
public int numIslands(char[][] grid) {
    int count = 0;
    for(int i = 0;i<grid.length;i++){
        for(int j = 0;j<grid[0].length;j++){
            if(grid[i][j] == '1'){
            ispossible(i,j,grid);
            count++;
        }
    }
    }
    return count;
```

35. Minimum String Length After Removing Substrings

You can apply some operations to this string where, in one operation, you can remove any occurrence of one of the substrings "AB" or "CD" from s.

Return the minimum possible length of the resulting string that you can obtain.

Note that the string concatenates after removing the substring and could produce new "AB" or "CD" substrings.

Input: s = "ABFCACDB"

Output: 2

Explanation: We can do the following operations:

- Remove the substring "ABFCACDB", so s = "FCACDB".

- Remove the substring "FCACDB", so s = "FCAB".

- Remove the substring "FCAB", so s = "FC".

So the resulting length of the string is 2.

It can be shown that it is the minimum length that we can obtain.

```
    Stack<Character> stack = new Stack<>();

    for(char c : s.toCharArray()){

            if(!stack.isEmpty() && (c == 'D' && stack.peek() == 'C' ||
            c == 'B' && stack.peek() == 'A'))
            stack.pop();
        else stack.push(c);
}
return stack.size();
```

36. Linked List Matrix - [Easy]

Given a Matrix mat of n*n size. Your task is constructing a 2D linked list representation of the given matrix.

Input: mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

```
Node head = new Node(arr[0][0]);
Node temp = head;
for(int i = 1;i<arr[0].length;i++){
    temp.right = new Node(arr[0][i]);
    temp = temp.right;
}
temp = head;
Node temprev = head;
for(int i = 1;i<arr.length;i++){
    Node tempnex = new Node(arr[i][0]);
    Node tempit = tempnex;
    Node curr = tempnex;
    for(int j = 1;j<arr[0].length;j++){
        tempit.right = new Node(arr[i][j]);
        tempit = tempit.right;
    }

    while(temprev!=null){
        temprev.down = curr;
        temprev = temprev.right;
        curr = curr.right;
    }
    temprev = tempnex;
}
return head;
```

37. 921. Minimum Add to Make Parentheses Valid

A parentheses string is valid if and only if:

It is the empty string,

It can be written as AB (A concatenated with B), where A and B are valid strings, or

It can be written as (A), where A is a valid string.

You are given a parentheses string s. In one move, you can insert a parenthesis at any position of the string.

For example, if s = "()))", you can insert an opening parenthesis to be "(()))" or a closing parenthesis to be "())))".

Return the minimum number of moves required to make s valid.

Input: s = "())"

Output: 1

```
    int open = 0;
    int imbalance = 0;
    for(char c : s.toCharArray()){
        if(c == '(') open++;

        else {
        if(open > 0) open--;

        else imbalance++;
        }
}
return (imbalance + open);
```

```
Stack<Character> stack = new Stack<>();

for(char c : s.toCharArray()){
    if(!stack.isEmpty() && (c == ')' && stack.peek() == '(')){
        stack.pop();
    }
    else stack.push(c);
}

return stack.size();
```

38. Reorganize the array:

Given an array of elements arr[] with indices ranging from 0 to arr.size() - 1, your task is to write a program that rearranges the elements of the array such that arr[i] = i. If an element i is not present in the array, -1 should be placed at the corresponding index.

Input: arr[] = [-1, -1, 6, 1, 9, 3, 2, -1, 4, -1] Output: [-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]

```
 ArrayList<Integer> res = new ArrayList<>();
for(int i = 0;i<arr.size();i++){
    while(arr.get(i)!=i && arr.get(i) != -1){
        int correct = arr.get(i);
        while(arr.get(correct) == correct || correct >= arr.size() ) break;

        int temp = arr.get(correct);
        arr.set(correct, arr.get(i));
        arr.set(i,temp);
    }
}

for(int i = 0;i<arr.size();i++){
    if(arr.get(i)!=i) arr.set(i,-1);
}
return arr;
```

39. The number of the smallest unoccupied chair - Medium

There is a party where n friends numbered from 0 to n - 1 are attending. There is an infinite number of chairs in this party that are numbered from 0 to infinity. When a friend arrives at the party, they sit on the unoccupied chair with the smallest number.

For example, if chairs 0, 1, and 5 are occupied when a friend comes, they will sit on chair number 2.

When a friend leaves the party, their chair becomes unoccupied at the moment they leave. If another friend arrives at that same moment, they can sit in that chair.

You are given a 0-indexed 2D integer array times where times[i] = [arrivali, leavingi], indicating the arrival and leaving times of the ith friend respectively, and an integer targetFriend. All arrival times are distinct.

Return the chair number that the friend numbered targetFriend will sit on.

Example 1:

Input: times = [[1,4],[2,3],[4,6]], targetFriend = 1 Output: 1

- Friend 0 arrives at time 1 and sits on chair 0.

- Friend 1 arrives at time 2 and sits on chair 1.

- Friend 1 leaves at time 3 and chair 1 becomes empty.

- Friend 0 leaves at time 4 and chair 0 becomes empty.

- Friend 2 arrives at time 4 and sits on chair 0.

Since friend 1 sat on chair 1, we return 1.

```
    List<int []> arr = new ArrayList<>();
    for(int i = 0;i<times.length;i++){
        arr.add(new int[]{times[i][0],i});
    }
    arr.sort((a,b) -> Integer.compare(a[0],b[0]));
    PriorityQueue<Integer> available = new PriorityQueue<>();
    for(int i = 0;i<times.length;i++){
        available.add(i);
    }
PriorityQueue<int []> leaving = new PriorityQueue<>((a, b) -> Integer.compare(a[0], b[0]));
    for(int [] a : arr){
        int at = a[0];
        int ai = a[1];
        while(!leaving.isEmpty() && leaving.peek()[0] <= at){
            available.add(leaving.poll()[1]);
        }
        int chair = available.poll();
        if(ai == targetFriend) return chair;
        leaving.add(new int[]{times[ai][1],chair});
```

40. Two Smallests in Every Subarray - [Easy]

Given an array of integers arr, the task is to find and return the maximum sum of the smallest and second smallest element among all possible subarrays of size greater than one. If it is not possible, then return -1.

Input: arr = [4, 3, 1, 5, 6] Output: 11

```
// code here
if(arr.length <= 1) return -1;
int sum = 0;
int max = -1;
int k = 0;
for(int i = 0;i<arr.length;i++){
    if(i < 2){
        sum = sum + arr[i];
        max = Math.max(max,sum);
    }
    else{
        sum = sum + arr[i];
        sum = sum - arr[k];
        k++;
        max = Math.max(max,sum);
    }
}
return max;
```

41. Merge K sorted Linked Lists - [Medium]

```
public ListNode mergeKLists(ListNode[] lists) {
    PriorityQueue<ListNode> pq = new PriorityQueue<>((a,b) -> a.val - b.val);
    for(ListNode node : lists){
        if(node!=null){
            pq.add(node); //stores the first value of each linkedlist
        }
    }
    ListNode head = new ListNode(0);
    ListNode curr = head;

    while(!pq.isEmpty()){
        ListNode node = pq.poll();
        curr.next = node;
        curr = curr.next;

        if(node.next!=null){
            pq.add(node.next);  //removed node next element is stored
        }
    }
    return head.next;
```

42. Minimum in sliding window - dequeue

```java
public static int[] findMinimums(int[] nums, int k) {
    if (nums == null || nums.length < k || k <= 0) {
        return new int[0];
    }

    int[] result = new int[nums.length - k + 1];
    Deque<Integer> deque = new ArrayDeque<>();

    for (int i = 0; i < nums.length; i++) {
        // Remove elements that are out of the current window
        while (!deque.isEmpty() && deque.peekFirst() < i - k + 1) {
            deque.pollFirst();
        }

        // Remove elements that are larger than the current element
        while (!deque.isEmpty() && nums[deque.peekLast()] >= nums[i]) {
            deque.pollLast();
        }

        deque.addLast(i);

        // Add the minimum to the result array
        if (i >= k - 1) {
            result[i - k + 1] = nums[deque.peekFirst()];
```

43. Subarray range with given sum - [Medium] - {O(n)}

Given an unsorted array of integers arr[], and a target tar, determine the number of subarrays whose elements sum up to the target value.

Input: arr[] = [10, 2, -2, -20, 10] , tar = -10

Output: 3

Explanation: Subarrays with sum -10 are: [10, 2, -2, -20], [2, -2, -20, 10] and [-20, 10].

```java
// add your code here
HashMap<Integer,Integer> map = new HashMap<>();
int sum = 0;
map.put(0,1);
int ans = 0;
for(int i :arr){
    sum+= i;
    if(map.containsKey(sum - tar)) ans += map.get(sum - tar);

    map.put(sum,map.getOrDefault(sum,0)+1);
}
return ans;
```

44. 38. Separate Black and White Balls

There are n balls on a table, each ball has a color black or white.

You are given a 0-indexed binary string s of length n, where 1 and 0 represent black and white balls, respectively.

In each step, you can choose two adjacent balls and swap them.

Return the minimum number of steps to group all the black balls to the right and all the white balls to the left.

Input: s = "101"

Output: 1

Explanation: We can group all the black balls to the right in the following way:

- Swap s[0] and s[1], s = "011".

Initially, 1s are not grouped together, requiring at least 1 step to group them to the right.

(from right - based on 1)

```
long count = 0;
long ans = 0;
int j = s.length()-1;
while(j>=0){
    if(s.charAt(j) == '0') count++;

    else if(s.charAt(j) == '1'){
        ans += count;
    }
    j--;
}
return ans;
```

or

(from left based on 0)

```
long total = 0;
int swap = 0;
for(int i = 0;i<s.length();i++){
    if(s.charAt(i) == '0'){
        total += (i - swap);
        swap++;
    }
}
return total;
```

45. Flattening a Linked List

Given a Linked List, where every node represents a sub-linked-list and contains two pointers:

(i) a next pointer to the next node,

(ii) a bottom pointer to a linked list where this node is head.

Each of the sub-linked lists is in sorted order.

Flatten the Link List so all the nodes appear in a single level while maintaining the sorted order.

Note: The flattened list will be printed using the bottom pointer instead of the next pointer. Look at the printList() function in the driver code for more clarity.

Recursively find the last and last second roots, merge them (point next to null) and return the merged

```
Node flatten(Node root) {
   if(root == null || root.next == null) return root;

   Node headnex = flatten(root.next);
   root = merge(root,headnex);
   return root;
}
```

```
Node merge(Node temp1, Node temp2){
    Node dummy = new Node(-1);
    Node curr = dummy;

    while(temp1 != null && temp2!=null){
        if(temp1.data < temp2.data){
            curr.bottom = temp1;
            curr = temp1;
            temp1 = temp1.bottom;
        }
        else{
            curr.bottom = temp2;
            curr = temp2;
            temp2 = temp2.bottom;
        }
        curr.next = null;
    }

    if(temp1!=null) curr.bottom = temp1;
    else curr.bottom = temp2;

    if(dummy.bottom != null) dummy.bottom.next = null;
    return dummy.bottom;
}
```

46. Maximum Swap -[Medium]

You are given an integer num. You can swap two digits at most once to get the maximum valued number.Return the maximum valued number you can get.

Input: num = 2736 Output: 7236

Explanation: Swap the number 2 and the number 7.

Approach - Convert to String array and initiate another array of 10 digits to store the last occurrence of each digit present in the array. In the loop, first loop is for nums array iteration and second loop is from 9 - 0 where the loop condition is to check if d is greater than the initial digit of the first loop and the second condition is if the element's value (index)in last array is greater than i, then swap.

```
    char [] nums = Integer.toString(num).toCharArray();
    int n = nums.length;

    int last[] = new int[10];

    for(int i = 0;i<n;i++){
        last[nums[i] - '0'] = i;
    }
    for(int i = 0;i<n;i++){
        for(int d = 9;d > nums[i] - '0';d--){
            if(last[d] > i){
                char temp = nums[i];
                nums[i] = nums[last[d]];
                nums[last[d]] = temp;
                return Integer.parseInt(new String(nums));
            }
        }
    }
```

47. Split Linked List Alternatingly - [Easy]

Given a singly linked list's head. Your task is to complete the function alternatingSplitList() that splits the given linked list into two smaller lists. The sublists should be made from alternating elements from the original list.

The sublist should be in the order with respect to the original list.

Your have to return an array containing the both sub-linked lists.

Input: LinkedList = 0->1->0->1->0->1 Output: 0->0->0 , 1->1->1

```
Node head1 = head;
Node h1 = head;
if(head == null) return new Node[]{null,null};

if(head.next == null)
{
    return new Node[] {head1,null};
}
Node head2 = head.next;
Node h2 = head2;
while(head1 != null && head2 != null && head2.next!=null){
    head1.next = head2.next;
    head1 = head1.next;
    head2.next = head1.next;
    head2 = head2.next;
}
head1.next = null;
return new Node [] {h1,h2};
```

48. Find distinct subsets - [Medium] (recursive)

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

```java
public void backtrack(int [] nums, int s, List<Integer> path, List<List<Integer>> res){
    res.add(new ArrayList<>(path));

    for(int i = s;i<nums.length;i++){
        path.add(nums[i]);
        backtrack(nums,i+1,path,res);
        path.remove(path.size() - 1);
    }
}
public List<List<Integer>> subsets(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    backtrack(nums,0, new ArrayList<>(), res);
    return res;

}
```

49. 2044. Count Number of Maximum Bitwise-OR Subsets - [Medium]

Given an integer array nums, find the maximum possible bitwise OR of a subset of nums and return the number of different non-empty subsets with the maximum bitwise OR.

An array a is a subset of an array b if a can be obtained from b by deleting some (possibly zero) elements of b. Two subsets are considered different if the indices of the elements chosen are different.

The bitwise OR of an array a is equal to a[0] OR a[1] OR ... OR a[a.length - 1] (0-indexed).

Input: nums = [3,1]

Output: 2

Explanation: The maximum possible bitwise OR of a subset is 3. There are 2 subsets with a bitwise OR of 3: [3], [3,1]

```java
public void backtrack(int [] nums, int index, int currOR, int maxOR, int [] count){
    if(currOR == maxOR) count[0]++;

    for(int i = index,i<nums.length;i++){
        backtrack(nums,i+1,currOR | nums[i], maxOR, count);
    }
}
public int countMaxOrSubsets(int[] nums) {
    int x = 0;
    for(int i : nums){
        x = x | nums;
    }
    int [] count = new int[1];
    backtrack(nums,0,0,x,count);
    return count[0];
}
```

50. Nearest multiple of 10

A string str is given to represent a positive number. The task is to round str to the nearest multiple of 10. If you have two multiples equally apart from str, choose the smallest element among them.

Input: str = 29

Output: 30

Explanation: Close multiples are 20 and 30, and 30 is the nearest to 29.

```
    int n = str.length();
    StringBuilder sb = new StringBuilder(str);

    if(sb.charAt(n-1) > '5' && sb.charAt(n-1) <= '9'){
        sb.setCharAt(n-1,'0');
        int i = 2;

        while(n - i >= 0 && sb.charAt(n-i) == '9'){
            sb.setCharAt(n-i,'0');
            i++;
        }
        sb.setCharAt(n - i, (char) (sb.charAt(n - i) + 1));
    }
    else{
        sb.setCharAt(n-1,'0');
    }
    return sb.toString();
}
```

51. 1545. Find Kth Bit in Nth Binary String - [Medium]

S1 = "0"

Si = Si - 1 + "1" + reverse(invert(Si - 1)) for i > 1

Where + denotes the concatenation operation, reverse(x) returns the reversed string x, and invert(x) inverts all the bits in x (0 changes to 1 and 1 changes to 0).

S1 = "0"

S2 = "011"

S3 = "0111001"

S4 = "011100110110001"

Return the kth bit in Sn. It is guaranteed that k is valid for the given n.

Input: n = 3, k = 1

Output: "0"

Explanation: S3 is "0111001".

The 1st bit is "0".

The observation is the length of the Sn would be (2^n-1) so we calculate the length first and find the middle element. If k == mid, return 1 as before 1 is the previous string and after 1 is the reversed inversion of the previous string (pattern). If k < mid, find the same else find the char of the previous inversed string with length length - k + 1.

```
public char findKthBit(int n, int k) {
    if(n==1) return '0';
    int length = (1 << n) - 1;
    int mid = (length)/2 + 1;
    if(k == mid) return '1';
    if(k < mid) return (findKthBit(n-1,k));
    else return findKthBit(n-1,length - k + 1) == '0' ? '1' : '0';
}
```

52. Sort a k sorted doubly linked list

Given a doubly linked list, each node is at most k-indices away from its target position. The problem is to sort the given doubly linked list. The distance can be assumed in either of the directions (left and right).

Input: Doubly Linked List : 3 <-> 2 <-> 1 <-> 5 <-> 6 <-> 4 , k = 2

Output: 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6

```
PriorityQueue<DLLNode> pq = new PriorityQueue<>((a,b) -> a.data - b.data);
DLLNode temp = null;
DLLNode curr=null;
while(i<=k && head != null){
    pq.add(head);
    head = head.next;
    i++;
}
while(!pq.isEmpty()){
    if(temp == null){
        temp = pq.poll();
        temp.prev = null;
        curr = temp;
    }
    else{
        curr.next = pq.poll();
        curr.next.prev = curr;
        curr = curr.next;
    }
    if(head != null){
        pq.add(head);
        head = head.next;
    }
}

curr.next = null;
return temp;
```

53. Split a String Into the Max Number of Unique Substrings - [Backtracking]

Given a string s, return the maximum number of unique substrings that the given string can be split into.

You can split string s into any list of non-empty substrings, where the concatenation of the substrings forms the original string. However, you must split the substrings such that all of them are unique.

A substring is a contiguous sequence of characters within a string.

Input: s = "ababccc"

Output: 5

Explanation: One way to split maximally is ['a', 'b', 'ab', 'c', 'cc']. Splitting like ['a', 'b', 'a', 'b', 'c', 'cc'] is not valid as you have 'a' and 'b' multiple times.

```java
public int Backtrack(int index, String s, HashSet<String> set ){
    if(index == s.length()) return 0;
    int count = 0;
    for(int i = index+1;i<=s.length();i++){
        String sub = s.substring(index,i);
        if(!set.contains(sub)){
            set.add(sub);
            count = Math.max(count, 1 + Backtrack(i,s,set));
            set.remove(sub);
        }
    }
    return count;
}
public int maxUniqueSplit(String s) {
    HashSet<String> set = new HashSet<>();
    return  Backtrack(0,s,set);
```

54. Split the Array

Given an array arr[] of integers, the task is to count the number of ways to split given array elements into two disjoint groups such that the XOR of elements of each group is equal.

Note: The answer could be very large so print it by doing modulo with 109 + 7.

Input : arr[] = [1, 2, 3]

Output : 3

Explanation: {(1),(2, 3)}, {(2),(1, 3)}, {(3),(1, 2)} are three ways with equal XOR value of two groups.

```java
// Complete the function
int mid = 1000000007;
int ans = 0;
for(int i : arr){
    ans = ans ^ i;
}
    if(ans!=0) return 0;
return (int) ((Math.pow(2,arr.length - 1)) - 1)%1000000007;
```

55. Sub-arrays with equal number of occurrences - [Medium]

Given an array arr[] and two integers say, x and y, find the number of sub-arrays in which the number of occurrences of x is equal to the number of occurrences of y.

Input: arr[] = [1, 2, 1] , x= 1 , y = 2 Output: 2

```
int count = 0;
int ans = 0;
Map<Integer,Integer> map = new HashMap<>();
map.put(0,1);
for(int i = 0;i<arr.length;i++){
    if(arr[i] == x){
        count++;
    }
    else if(arr[i] == y) count--;

    if(map.containsKey(count)){
        ans += map.get(count);
    }
    map.put(count, map.getOrDefault(count,0)+1);
}
return ans;
```

56. 2583. Kth Largest Sum in a Binary Tree - [Medium, Level order]

You are given the root of a binary tree and a positive integer k.

The level sum in the tree is the sum of the values of the nodes that are on the same level.

Return the kth largest level sum in the tree (not necessarily distinct). If there are fewer than k levels in the tree, return -1.

Note that two nodes are on the same level if they have the same distance from the root.

Input: root = [5,8,9,2,1,3,7,4,6], k = 2 Output: 13

```java
public long kthLargestLevelSum(TreeNode root, int k) {
    PriorityQueue<Long> pq = new PriorityQueue<>((a,b) -> Long.compare(b,a));
    Queue<TreeNode> q = new LinkedList<>();
    if(root == null) return -1;
    q.offer(root);
    while(!q.isEmpty()){
        int level = q.size();
        long count = 0;
        for(int i = 0;i<level;i++){
            if(q.peek().left != null) q.offer(q.peek().left);
            if(q.peek().right != null) q.offer(q.peek().right);
            count = count+ q.poll().val;
        }
        pq.add(count);
    }
    while(k-- > 1){
        pq.poll();
    }
    return (!pq.isEmpty()) ? pq.poll() : -1;
```

57. Find the Sum of Last N nodes of the Linked List

Given a single linked list, calculate the sum of the last n nodes.

```java
    // write code here
    int len = 0;
Node temp = head;
while(temp!=null){
    len++;
    temp = temp.next;
}

temp = head;
int sum = 0;

if(n>=len){
    while(temp!=null){
    sum = sum + temp.data;
    temp = temp.next;
    }
    return sum;
}
```

```java
else{
    len = len - n;
    while(temp!=null && len > 0){
        temp = temp.next;
        len--;
    }

    while(temp!=null){
        sum = sum + temp.data;
        temp = temp.next;
    }
}
return sum;
}
```

58. 1233. Remove Sub-Folders from the Filesystem - [Medium]

Given a list of folders folder, return the folders after removing all sub-folders in those folders. You may return the answer in any order.

If a folder[i] is located within another folder[j], it is called a sub-folder of it. A sub-folder of folder[j] must start with folder[j], followed by a "/". For example, "/a/b" is a sub-folder of "/a", but "/b" is not a sub-folder of "/a/b/c".

The format of a path is one or more concatenated strings of the form: '/' followed by one or more lowercase English letters.

For example, "/leetcode" and "/leetcode/problems" are valid paths while an empty string and "/" are not.

Input: folder = ["/a","/a/b","/c/d","/c/d/e","/c/f"] Output: ["/a","/c/d","/c/f"]

Explanation: Folders "/a/b" is a subfolder of "/a" and "/c/d/e" is inside of folder "/c/d" in our filesystem.

```java
public List<String> removeSubfolders(String[] folder) {
    Arrays.sort(folder);
   ArrayList<String> arr = new ArrayList<>();
   arr.add(folder[0]);
   for(int i = 1;i<folder.length;i++){
    String last = arr.get(arr.size() - 1) + "/";
    if(!folder[i].startsWith(last)) arr.add(folder[i]);
   }
    return arr;
```

59. Max depth of a tree:

```java
public int maxDepth(TreeNode root) {
    if(root == null) return 0;
    return calcdepth(root);
}

public int calcdepth(TreeNode root){
    if(root == null) return 0;

    int left = Math.max(1,1+calcdepth(root.left));
    int right = Math.max(1,1+calcdepth(root.right));

    return Math.max(left,right);
}
```

60 458. Height of Binary Tree After Subtree Removal Queries -[HARDDDDDDDDD]

You are given the root of a binary tree with n nodes. Each node is assigned a unique value from 1 to n. You are also given an array queries of size m.

You have to perform m independent queries on the tree where in the ith query you do the following:

Remove the subtree rooted at the node with the value queries[i] from the tree. It is guaranteed that queries[i] will not be equal to the value of the root.

Return an array answer of size m where answer[i] is the height of the tree after performing the ith query.

The queries are independent, so the tree returns to its initial state after each query.

The height of a tree is the number of edges in the longest simple path from the root to some node in the tree.

Input: root = [1,3,4,2,null,6,5,null,null,null,null,null,7], queries = [4]

Output: [2]

Explanation: The diagram above shows the tree after removing the subtree rooted at node with value 4.

The height of the tree is 2 (The path 1 -> 3 -> 2).

```
private Map<Integer, Integer> h = new HashMap<>();
private Map<Integer, Integer> d = new HashMap<>();
private Map<Integer, List<Integer>> dh = new HashMap<>();
public int clc(TreeNode root, int depth){
    if(root == null) return -1;

    d.put(root.val,depth);

    int l = clc(root.left,depth+1);
    int r = clc(root.right,depth+1);

    int height = 1 + Math.max(l,r);
    h.put(root.val,height);

    return height;
}
```

```
public int[] treeQueries(TreeNode root, int[] queries) {
  clc(root,0);

  for(Map.Entry<Integer,Integer> map: h.entrySet()){
    int nval = map.getKey();
    int height = map.getValue();
    int depth = d.get(nval);
    dh.computeIfAbsent(depth,k -> new ArrayList<>()).add(height);
  }
  for(List<Integer> hl : dh.values()){
    Collections.sort(hl, Collections.reverseOrder());
  }
  int [] res = new int[queries.length];

  for(int i = 0;i<queries.length;i++){
    int nval = queries[i];
    int depth = d.get(nval);
    int height = h.get(nval);
    List<Integer> hl = dh.get(depth);
```

```
    int max;
    if(hl.size() == 1){
        max = depth - 1;
    }
    else if(hl.get(0) == height) max = hl.get(1) + depth;
  else max = depth + hl.get(0);

  res[i] = max;
}
return res;
```

61. . Count Square Submatrices with All Ones - [Medium]

Given a m * n matrix of ones and zeros, return how many square submatrices have all ones.

Input: matrix =

[

[0,1,1,1],

[1,1,1,1],

[0,1,1,1]

]

Output: 15

Explanation:

There are 10 squares of side 1.

There are 4 squares of side 2.

There is 1 square of side 3.

Total number of squares = 10 + 4 + 1 = 15.

```
int ans = 0;
int [][] dp = new int[n][m];

for(int i = 0;i<n;i++){
    dp[i][0] = matrix[i][0];
    ans += matrix[i][0];
}
for(int j = 1;j<m;j++){
     dp[0][j] = matrix[0][j];
     ans += matrix[0][j];
}
for(int i = 1;i<n;i++){
    for(int j = 1;j<m;j++){
        if(matrix[i][j] == 1){
            dp[i][j] = 1 + Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]));
        }
        ans += dp[i][j];
    }
}
return ans;
```

`

62. Pairs with difference k

Difficulty: EasyAccuracy: 22.41%Submissions: 18K+Points: 2

Given an array arr[] of positive integers. Find the number of pairs of integers whose difference equals a given number k.

Note: (a, b) and (b, a) are considered the same. Also, the same numbers at different indices are considered different.

```
HashMap<Integer,Integer> map = new HashMap<>();
int count = 0;
for(int i : arr){
    map.put(i,map.getOrDefault(i,0)+1);

    if(map.containsKey(i - k)) count += map.get(i-k);

    if(map.containsKey(i+k)) count += map.get(i+k);

}
return count;
```

63. Insert in Sorted way in a Sorted DLL

Given a sorted doubly linked list and an element x, you need to insert the element x into the correct position in the sorted Doubly linked list(DLL).

Note: The DLL is sorted in ascending order

```
Node temp = head;
if(head == null) return null;

if(temp.data > x){
    temp.prev = new Node(x);
    temp.prev.next = temp;
    head = temp.prev;
    return head;
}
while(temp.next != null){

    if(temp.next != null && x>= temp.data && x <=temp.next.data
        Node y = temp.next;
        temp.next = new Node(x);
        temp.next.prev = temp;
        temp = temp.next;
        temp.next = y;
        temp.next.prev = temp;
        return head;
    }
    temp = temp.next;
    }
    temp.next = new Node(x);
    temp.next.prev = temp;
    return head;
```

Recursion:

64. Pow(x,n) - Medium - Recursion

```
class Solution {
    public double helper(double x, int n){
        if(x == 0) return 0;
        if(n == 0) return 1;
        double res = helper(x,n/2);
        res = res * res;
        return (n%2 == 0) ? res : x*res;
    }
    public double myPow(double x, int n) {
       double res = helper(x,Math.abs(n));
       return (n >= 0) ? res : 1/res;
    }
}
```

z

65. Count Good Numbers - medium

A digit string is good if the digits (0-indexed) at even indices are even and the digits at odd indices are prime (2, 3, 5, or 7).

For example, "2582" is good because the digits (2 and 8) at even positions are even and the digits (5 and 2) at odd positions are prime. However, "3245" is not good because 3 is at an even index but is not even.

Given an integer n, return the total number of good digit strings of length n. Since the answer may be large, return it modulo 109 + 7.

A digit string is a string consisting of digits 0 through 9 that may contain leading zeros.

Input: n = 1 Output: 5

```
public int countGoodNumbers(long n) {
    long even = (n+1)/2;
    long odd = n/2;
    long first = pow(5,even); //[0,2,4,6,8] - even on e
    long second = pow(4,odd);  //[2,3,5,7] - prime on o
    return (int) ((first * second)% mod);
}
public long pow(long x, long n){
    if(n == 0) return 1;
    if(x == 0) return 0;
    long res = pow(x,n/2);
    res = (res * res)%mod;
    return (n%2 == 0) ? res : (res * x)%mod;
}
```

66. Sort a Stack - Medium

```java
public void sorted(Stack<Integer> s){
    if(s.size() == 0) return;
    int temp = s.pop();
    sorted(s);
    insert(s,temp);
}

public void insert(Stack<Integer> s, int temp){
    if(s.size() == 0 || s.get(s.size() - 1) <= temp ){
    s.push(temp);
    return;
    }
    int x = s.pop();
    insert(s,temp);
    s.push(x);
    return;
}
public Stack<Integer> sort(Stack<Integer> s) {
    // add code here.
    sorted(s);
    return s;
}
```

67. Maximum Sum of Distinct Subarrays With Length K

```java
 long sum = 0;
 long max = 0;
 int j = 0;
for(int i = 0;i<nums.length;i++){
 sum += nums[i];
 map.put(nums[i],map.getOrDefault(nums[i],0)+1);

 if(i - j + 1 > k){
    map.put(nums[j],map.get(nums[j])-1);
    if(map.get(nums[j]) == 0){
        map.remove(nums[j]);
    }
        sum -= nums[j];
        j++;
    }

    if( map.size() == i-j + 1 && i - j + 1 == k){
        max = Math.max(max,sum);
    }
 }
return max;
```

Only taking indexes in map and not count. Different though, skip sames ones.

```
  long sum = 0;
  long max = 0;
  int l = 0;
 for(int i = 0;i<nums.length;i++){
  sum += nums[i];
  if(!map.containsKey(nums[i])){
      map.put(nums[i],-1);
  }
  int j = map.get(nums[i]);
  while(l <= j || i-l+1 > k){
      sum -= nums[l];
      l++;
  }
  if(i - l + 1 == k) max = Math.max(max,sum);
  map.put(nums[i],i);
```

68. Maximum Matrix Sum - Medium

Input: matrix = [[1,-1],[-1,1]]

Output: 4

Explanation: We can follow the following steps to reach sum equals 4:

- Multiply the 2 elements in the first row by -1.

- Multiply the 2 elements in the first column by -1.

```
 int count=0;
 long sum = 0;
 int min = Integer.MAX_VALUE;
for(int i = 0;i<matrix.length;i++){
    for(int j = 0;j<matrix[0].length;j++){
        if(matrix[i][j] < 0) count++;

        sum += (long) Math.abs(matrix[i][j]);
        min = Math.min(min,Math.abs(matrix[i][j]));
    }
 }
 if(count%2 == 0) return sum;
 sum = (long) sum - (long) 2*min;
 return sum;
```

69. Reverse a Stack - Medium

You are given a stack St. You have to reverse the stack using recursion.

```
static void rev(Stack<Integer> s, Stack<Integer> ns){
    if(s.size() == 0) return;
    int x = s.pop();
    ns.push(x);
    rev(s,ns);
}
static void recrev(Stack<Integer> s, Stack<Integer> ns){
    int i =ns.size();
    while(i >0){
        s.push(ns.get(ns.size() - i));
        i--;
    }
}
static void reverse(Stack<Integer> s)
{
    Stack<Integer> ns = new Stack<>();
    rev(s,ns);
    recrev(s,ns);
}
```

70. Generate parentheses - Medium

```
public void rec(int open, int close, List<String> arr,String s, int n){
    if(open == n && close == n) {
    arr.add(s);
    return;
    }
    if(open < n) rec(open+1,close,arr,s+"(",n);
    if(close < open) rec(open,close+1,arr,s+")",n);
}
public List<String> generateParenthesis(int n) {
    List<String> arr = new ArrayList<>();
    rec(0,0,arr,"",n);
    return arr;
}
```

80. 2981. Find Longest Special Substring That Occurs Thrice I

You are given a string s that consists of lowercase English letters.

A string is called special if it is made up of only a single character. For example, the string "abc" is not special, whereas the strings "ddd", "zz", and "f" are special.

Return the length of the longest special substring of s which occurs at least thrice, or -1 if no special substring occurs at least thrice.

A substring is a contiguous non-empty sequence of characters within a string.

Input: s = "aaaa" Output: 2

Explanation: The longest special substring which occurs thrice is "aa": substrings "aaaa", "aaaa", and "aaaa".

It can be shown that the maximum length achievable is 2.

```
List<String> lis = new ArrayList<>();
for(int i = 0;i<s.length();i++){
    int index = i;

    while(index<s.length() && s.charAt(i) == s.charAt(index)){
        lis.add(s.substring(i,index+1));
        index++;
    }}
HashMap<String,Integer> map = new HashMap<>();
for(String sub : lis){
    map.put(sub, map.getOrDefault(sub,0)+1);
}
int max = -1;
for(Map.Entry<String,Integer> e : map.entrySet()){
    String substring = e.getKey();
    int count = e.getValue();
    if(count >2){
        max = Math.max(max,substring.length());
    }
```

81. 2779. Maximum Beauty of an Array After Applying Operation - Medium

You are given a 0-indexed array nums and a non-negative integer k. In one operation, you can do the following: Choose an index i that hasn't been chosen before from the range [0, nums.length - 1]. Replace nums[i] with any integer from the range [nums[i] - k,

nums[i] + k]. The beauty of the array is the length of the longest subsequence consisting of equal elements.

Return the maximum possible beauty of the array nums after applying the operation any number of times.

A subsequence of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the order of the remaining elements.

Input: nums = [4,6,1,2], k = 2

Output: 3

Explanation: In this example, we apply the following operations:

- Choose index 1, replace it with 4 (from range [4,8]), nums = [4,4,1,2].

- Choose index 3, replace it with 4 (from range [0,4]), nums = [4,4,1,4].

After the applied operations, the beauty of the array nums is 3 (subsequence consisting of indices 0, 1, and 3).

It can be proven that 3 is the maximum possible length we can achieve.

```java
public int maximumBeauty(int[] nums, int k) {
    Arrays.sort(nums);
    int n = nums.length;
    int res = 0;
    for(int i = 0;i<n;i++){
        int v = nums[i];
        int maxPos = BS(nums,v+2*k+1);
        int val = maxPos - i;
        res = Math.max(res,val);
    }
    return res;
}
```

```java
public int BS(int [] arr, int upper){
    int l = 0;
    int r = arr.length;
    while(l<r){
        int mid = l + (r-l)/2;
        if(arr[mid] < upper){
            l = mid + 1;
        }
        else r = mid;
    }
    return l;
}
```

82. 2558. Take Gifts From the Richest Pile

You are given an integer array gifts denoting the number of gifts in various piles. Every second, you do the following:

Choose the pile with the maximum number of gifts.

If there is more than one pile with the maximum number of gifts, choose any.

Leave behind the floor of the square root of the number of gifts in the pile. Take the rest of the gifts.

Return the number of gifts remaining after k seconds.

Input: gifts = [25,64,9,4,100], k = 4

Output: 29

Explanation: - In the first second, the last pile is chosen and 10 gifts are left behind.

- Then the second pile is chosen and 8 gifts are left behind.

- After that the first pile is chosen and 5 gifts are left behind.

- Finally, the last pile is chosen again and 3 gifts are left behind.

The final remaining gifts are [5,8,9,4,3], so the total number of gifts remaining is 29.

```java
public long pickGifts(int[] gifts, int k) {
    long sum = 0;
    PriorityQueue<Long> pq = new PriorityQueue<>(Collections.reverseOrder());
    for(int i : gifts)
    pq.add((long) i);
    while(k-- > 0){
        long x = pq.poll();
        long y = (long) Math.sqrt(x);
        pq.add(y);
    }
    for(Long ele : pq){
        sum += ele;
    }
    return sum;
```

83. 2762. Continuous Subarrays - [Medium Sliding]

You are given a 0-indexed integer array nums. A subarray of nums is called continuous if:

Let i, i + 1, ..., j be the indices in the subarray. Then, for each pair of indices i <= i1, i2 <= j, 0 <= |nums[i1] - nums[i2]| <= 2.

Return the total number of continuous subarrays. A subarray is a contiguous non-empty sequence of elements within an array.

Input: nums = [5,4,2,4] Output: 8 Continuous subarray of size 1: [5], [4], [2], [4].

Continuous subarray of size 2: [5,4], [4,2], [2,4].

Continuous subarray of size 3: [4,2,4]. Thereare no subarrys of size 4.

Total continuous subarrays = 4 + 3 + 1 = 8. It can be shown that there are no more continuous subarrays.

```
while(r < nums.length){

   maxHeap.add(r);
   minHeap.add(r);

   while(l<r && nums[maxHeap.peek()] - nums[minHeap.peek()] > 2){ l++;

   while(!maxHeap.isEmpty()  && maxHeap.peek() < l ) maxHeap.poll();
   while(!minHeap.isEmpty() && minHeap.peek() < l) minHeap.poll();
   }
   sum += r - l + 1;
   r++;
}
return sum;
```

84. Maximum sum of non-adjacent elements - Medium DP

```
public static int f(int n, ArrayList<Integer> nums, int [] dp){
    if(n == 0) return nums.get(n);
    if(n < 0) return 0;
    if(dp[n] != -1) return dp[n];
    int take = nums.get(n) + f(n-2,nums,dp);
    int not_take = 0 + f(n-1,nums,dp);
    return dp[n] = Math.max(take, not_take);
}
0 references
public static int maximumNonAdjacentSum(ArrayList<Integer> nums) {
    int dp [] = new int[nums.size()];
    for(int i=0;i<dp.length;i++) dp[i] = -1;
    return f(nums.size()-1,nums,dp);
```

```java
int dp [] = new int[nums.size()];
for(int i=0;i<dp.length;i++) dp[i] = -1;

dp[0] = nums.get(0);
for(int i = 1;i<nums.size();i++){

    int take = nums.get(i);
    if(i > 1) take += dp[i-2];
    int not_take = dp[i-1];

    dp[i] = Math.max(take,not_take);
}
return dp[nums.size() -1];
```

85. 1792. Maximum Average Pass Ratio - Medium

```java
public double maxAverageRatio(int[][] classes, int extraStudents) {
    PriorityQueue<double[]> pq = new PriorityQueue<>(
        (a, b) -> Double.compare(b[0], a[0]) );
    double sum = 0;
    for(int i = 0;i<classes.length;i++){
        double val1 = (double) classes[i][0]/classes[i][1];
        double val2 = (double) (classes[i][0]+1)/(classes[i][1]+1);
        double val = val2 - val1;
        pq.add(new double [] {val,i});
        }
```

```java
for(int i = 0;i<extraStudents;i++){
    double [] arr = pq.poll();
    int ind = (int) arr[1];
    classes[ind][0] +=1;
    classes[ind][1] +=1;
    double a = classes[ind][0];
    double b =  classes[ind][1];
    double val1 =  (a+1)/(b+1);
    double val2 = a/b;
    double x = val1 - val2;
    pq.add(new double[] {x, ind});
}
for(int [] c: classes){
    sum += (double) c[0]/c[1];
}
return sum/classes.length;
```

86. 3264. Final Array State After K Multiplication Operations I

Input: nums = [2,1,3,5,6], k = 5, multiplier = 2 Output: [8,4,6,5,6]

```
    PriorityQueue<int []> pq = new PriorityQueue<>(
        (a,b) -> {
           if(a[0] == b[0]) {
            return Integer.compare(a[1],b[1]);  //Compare by index if values are same
           }
           return Integer.compare(a[0],b[0]);
        });
    int [] arr = new int[nums.length];
    for(int i = 0;i<nums.length;i++) pq.add(new int[]{nums[i],i});
while(k-- > 0){
    int [] a = pq.poll();
    int val = a[0];
    int ind = a[1];
    val = val * multiplier;
    pq.add(new int [] {val,ind});
}
while(!pq.isEmpty()){
int []a = pq.poll();
arr[a[1]] = a[0];
}
```

87. 2182. Construct String With Repeat Limit

You are given a string s and an integer repeatLimit. Construct a new string repeatLimitedString using the characters of s such that no letter appears more than repeatLimit times in a row. You do not have to use all characters from s.

A string a is lexicographically larger than a string b if in the first position where a and b differ, string a has a letter that appears later in the alphabet than the corresponding letter in b. If the first min(a.length, b.length) characters do not differ, then the longer string is the lexicographically larger one.

Input: s = "cczazcc", repeatLimit = 3 Output: "zzcccac"

```
int [] freq = new int[26];
for(char c : s.toCharArray()){
    freq[c - 'a']++;
}
PriorityQueue<int[]> pq = new PriorityQueue<>((a,b) -> Integer.compare(b[0],a[0]))
for(int i = 0;i<26;i++){
    if(freq[i]>0){
        pq.add(new int[]{i,freq[i]});
    }
}
```

```
StringBuilder res = new StringBuilder();
while(!pq.isEmpty()){
    int curr [] = pq.poll();
    char c = (char) ('a' + curr[0] );
    int count = Math.min(repeatLimit, curr[1]);
    for(int i = 0;i<count;i++){
        res.append(c);
    }
    curr[1] = curr[1] - count;
    if(curr[1] > 0){
        if(pq.isEmpty()) break;
        int next [] = pq.poll();
        char cc = (char) ('a' + next[0]);
        res.append(cc);
        next[1]--;
        if(next[1] > 0) pq.add(next);
        pq.add(curr);
    }
}
return res.toString();
```

88. 769. Max Chunks To Make Sorted - Medium

You are given an integer array arr of length n that represents a permutation of the integers in the range [0, n - 1].

We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return the largest number of chunks we can make to sort the array.

Input: arr = [4,3,2,1,0] Output: 1

Splitting into two or more chunks will not return the required result.

For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0, 1, 2], which isn't sorted.

Check max at each index, if max == i, inc count. Till that index all the elements can be sorted and are present.

Check the sum till each element and check if the sortedsum == sum, if it equals at any index, inc count. An important observation is that a segment of the array can form a valid chunk if, when sorted, it matches the corresponding segment in the fully sorted version of the array.

```
// int count = 0;
// int max = -1;
// for(int i = 0;i<arr.length;i++){
//     max = Math.max(max,arr[i]);
//     if(i == max){
//         count++;
//     }
// }
// return count;

int sum = 0;
int sortedSum = 0;
int count = 0;
for(int i = 0;i<arr.length;i++){
    sum += arr[i];
    sortedSum += i;
    if(sortedSum == sum) count++;
```

89. 2415. Reverse Odd Levels of Binary Tree - Medium

Given the root of a perfect binary tree, reverse the node values at each odd level of the tree.Return the root of the reversed tree.

A binary tree is perfect if all parent nodes have two children and all leaves are on the same level.

The level of a node is the number of edges along the path between it and the root node.

Input: root = [2,3,5,8,13,21,34] Output: [2,5,3,8,13,21,34]

```java
Queue<TreeNode> q = new LinkedList<>();
q.add(root);
int level = 0;
while(!q.isEmpty()){
    int size = q.size();
    List<Integer> nodes = new ArrayList<>();
    Queue<TreeNode> tempq = new LinkedList<>(q);
    for(int i = 0; i<size;i++){
        TreeNode x = q.poll();
        if(x.left!=null) q.add(x.left);
        if(x.right!=null) q.add(x.right);
        nodes.add(x.val);
    }
    if(level % 2 != 0){
        Collections.reverse(nodes);
        int i = 0;
        while(size-- > 0){
            TreeNode x = tempq.poll();
            x.val = nodes.get(i);
            i++; }}
    level++;
```

90. 2872. Maximum Number of K-Divisible Components - Hard

There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree.

You are also given a 0-indexed integer array values of length n, where values[i] is the value associated with the ith node, and an integer k.

A valid split of the tree is obtained by removing any set of edges, possibly empty, from the tree such that the resulting components all have values that are divisible by k, where the value of a connected component is the sum of the values of its nodes.

Return the maximum number of components in any valid split.

Input: n = 5, edges = [[0,2],[1,2],[1,3],[2,4]], values = [1,8,1,4,4], k = 6

Output: 2

Explanation: We remove the edge connecting node 1 with 2. The resulting split is valid because:

- The value of the component containing nodes 1 and 3 is values[1] + values[3] = 12.
- The value of the component containing nodes 0, 2, and 4 is values[0] + values[2] + values[4] = 6.

It can be shown that no other valid split has more than 2 connected components.

```
public int dfs(ArrayList<ArrayList<Integer>> adj, int [] values, int k,
int [] count, int curr, int p){
    int sum = values[curr];
    for(int x : adj.get(curr)){
        if(x == p) continue;
        sum += dfs(adj,values,k,count,x,curr);}
    sum %= k;
    if(sum == 0) count[0]++;
    return sum;
}
```

```
public int maxKDivisibleComponents(int n, int[][] edges, int[] values, int k) {
    ArrayList<ArrayList<Integer>> arr = new ArrayList<>();
    for(int i = 0;i<values.length;i++){
        arr.add(new ArrayList<>()); }
    for(int i [] : edges){
        int u = i[0];
        int v = i[1];
        arr.get(u).add(v);
        arr.get(v).add(u);}
    int count [] = new int[1];
    dfs(arr,values,k, count, 0, -1);
    return count[0];
}
```

91. 2940. Find Building Where Alice and Bob Can Meet - ?????

```java
int n = heights.length, q = queries.length;
int[] result = new int[q];
Arrays.fill(result, -1);
List<List<int[]>> deferred = new ArrayList<>();
for (int i = 0; i < n; i++) {
    deferred.add(new ArrayList<>());
}
PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));

for (int i = 0; i < q; ++i) {
    int a = queries[i][0], b = queries[i][1];
    if (a > b) {
        int temp = a;
        a = b;
        b = temp;
    }
    if (a == b || heights[a] < heights[b]) result[i] = b;
    else deferred.get(b).add(new int[]{heights[a], i});
}

for (int i = 0; i < n; ++i) {
    for (int[] query : deferred.get(i)) pq.add(query);
    while (!pq.isEmpty() && pq.peek()[0] < heights[i]) {
        result[pq.poll()[1]] = i;
    }
```

92. Subsets - Medium

```java
public void rec(int [] nums, int s, List<Integer> path, List<List<Integer>> res){
    res.add(new ArrayList<>(path));
    for(int i = s;i<nums.length;i++){
        path.add(nums[i]);
        rec(nums,i+1,path,res);
        path.remove(path.size() - 1);
    }
}
public List<List<Integer>> subsets(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    rec(nums,0,new ArrayList<>(), res);
    return res;
}
```

93. Better String

Difficulty: HardAccuracy: 44.5%Submissions: 81K+Points: 8

Given a pair of strings of equal lengths, Geek wants to find the better string. The better string is the string having more number of distinct subsequences.

If both the strings have equal count of distinct subsequence then return str1.

Input:

str1 = "gfg", str2 = "ggg"

Output: "gfg"

Explanation: "gfg" have 6 distinct subsequences whereas "ggg" have 3 distinct subsequences.

```
public static int findsub(String str){
    int n = str.length();
    int [] dp = new int[n+1];
    Map<Character, Integer> map = new HashMap<>();

    dp[0] = 1;

    for(int i = 1;i<=n;i++){
        dp[i] = dp[i-1] * 2;

        if(map.containsKey(str.charAt(i-1))){
            dp[i] = dp[i] - dp[map.get(str.charAt(i-1))];
        }
        map.put(str.charAt(i-1), (i-1));
    }
    return dp[n];

}
public static String betterString(String str1, String str2) {
    // Code here
    int x1 = findsub( str1);
    int x2 = findsub(str2);

    return x1 >= x2 ? str1 : str2;
```

94. Greatest Common Divisor of Strings - Easy

For two strings s and t, we say "t divides s" if and only if s = t + t + t + ... + t + t (i.e., t is concatenated with itself one or more times).

Given two strings str1 and str2, return the largest string x such that x divides both str1 and str2.

Example 1: Input: str1 = "ABCABC", str2 = "ABC" Output: "ABC"

```
    if(!(str1+str2).equals(str2+str1)) return "";
    int l1 = str1.length();
    int l2 = str2.length();

    while(l2!=0){
        int t = l1 % l2;
        l1 = l2;
        l2 = t;
    }
    return str1.substring(0,l1);
```

95 605. Can Place Flowers - Easy

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Input: flowerbed = [1,0,0,0,1], n = 1

Output: true

```
class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        for(int i = 0;i<flowerbed.length;i++){
            boolean left = i == 0 || flowerbed[i-1] == 0;
            boolean right = i == flowerbed.length-1 || flowerbed[i+1] == 0;
            if(left && right && flowerbed[i] == 0){
                flowerbed[i] = 1;
                n--;
            }
        }
        return n<= 0;
    }
```

96. 1462. Course Schedule IV - Medium

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course ai first if you want to take course bi.

For example, the pair [0, 1] indicates that you have to take course 0 before you can take course 1.

Prerequisites can also be indirect. If course a is a prerequisite of course b, and course b is a prerequisite of course c, then course a is a prerequisite of course c.

You are also given an array queries where queries[j] = [uj, vj]. For the jth query, you should answer whether course uj is a prerequisite of course vj or not. Return a boolean array answer, where answer[j] is the answer to the jth query.

Input: numCourses = 2, prerequisites = [[1,0]], queries = [[0,1],[1,0]]

Output: [false,true]

Explanation: The pair [1, 0] indicates that you have to take course 1 before you can take course 0.

Course 0 is not a prerequisite of course 1, but the opposite is true.

```
boolean [][] graph = new boolean[numCourses][numCourses];

for(int i [] : prerequisites){
    graph[i[0]][i[1]] = true;
}

//Flyod Warshall Algorithm - compute transitive closure
for(int k = 0;k<numCourses;k++){
    for(int i = 0;i<numCourses;i++){
        for(int j = 0;j<numCourses;j++){
            graph[i][j] = graph[i][j] || (graph[i][k] && graph[k][j]);
        }
    }
}
List<Boolean> arr = new ArrayList<>();
for(int i [] : queries){
    arr.add(graph[i[0]][i[1]]);
}
return arr;
```

```java
List<List<Integer>> adj = new ArrayList<>();
int [] indegrees = new int[numCourses];
for(int i = 0;i<numCourses;i++){
 adj.add(new ArrayList<>());
}
for(int i [] : prerequisites){
 adj.get(i[0]).add(i[1]);
 indegrees[i[1]]++;  }
Queue<Integer> queue = new LinkedList<>();
for(int i = 0; i<numCourses;i++){
 if(indegrees[i] == 0){
     queue.offer(i);   }  }
Map<Integer, Set<Integer>> map = new HashMap<>();
for(int i = 0;i<numCourses;i++){
 map.put(i, new HashSet<>());
}
while(!queue.isEmpty()){
 int curr = queue.poll();
 for(int next : adj.get(curr)){
     map.get(next).add(curr);
     map.get(next).addAll(map.get(curr));
     indegrees[next]--;
     if(indegrees[next] == 0){
         queue.offer(next);  } } }
List<Boolean> lis = new ArrayList<>();
for(int [] q: queries){
 lis.add(map.get(q[1]).contains(q[0]));
```

97. 345. Reverse Vowels of a String - Easy

Given a string s, reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.

Input: s = "IceCreAm"

Output: "AceCreIm"

```java
HashSet<Character> set = new HashSet<>();
set.add('a');
set.add('e');
set.add('i');
set.add('o');
set.add('u');
set.add('A');
set.add('E');
set.add('I');
set.add('O');
set.add('U');
StringBuilder ns = new StringBuilder(s);
int i = 0, j = s.length()-1;
while(i<j){
    if(!set.contains(s.charAt(i))){
        i++;
        continue;
    }
    if(!set.contains(s.charAt(j))){
        j--;
        continue;
    }
    else{
        char temp = ns.charAt(i);
        ns.setCharAt(i, ns.charAt(j));
        ns.setCharAt(j, temp); // using char
        i++;
        j--;    } }
return ns.toString();
```

98. 1800. Maximum Ascending Subarray Sum

```java
    int sum = nums[0];
    int max = sum;
    for(int i = 1;i<nums.length;i++){
        if(nums[i-1] < nums[i]){
            sum += nums[i];
        }
        else{
            sum = nums[i];
        }
        max = Math.max(max,sum);
    }
    return max;
}
```

99. 2364. Count Number of Bad Pairs - Medium

You are given a 0-indexed integer array nums. A pair of indices (i, j) is a bad pair if i < j and j - i != nums[j] - nums[i]. Return the total number of bad pairs in nums.

Input: nums = [4,1,3,3]

Output: 5

Explanation: The pair (0, 1) is a bad pair since 1 - 0 != 1 - 4.

The pair (0, 2) is a bad pair since 2 - 0 != 3 - 4, 2 != -1.

The pair (0, 3) is a bad pair since 3 - 0 != 3 - 4, 3 != -1.

The pair (1, 2) is a bad pair since 2 - 1 != 3 - 1, 1 != 2.

The pair (2, 3) is a bad pair since 3 - 2 != 3 - 3, 1 != 0.

There are a total of 5 bad pairs, so we return 5.

```
HashMap<Integer, Integer> map = new HashMap<>();
long count = 0;
for(int i = 0;i<nums.length;i++){
    if(map.containsKey(nums[i] - i)){
        count += map.get(nums[i] - i);
    }
        map.put(nums[i] - i, map.getOrDefault(nums[i] - i, 0)+1);
}
long total = (long) nums.length * (nums.length - 1)/2;
return total - count;
```

100. 2342. Max Sum of a Pair With Equal Sum of Digits - Medium

You are given a 0-indexed array nums consisting of positive integers. You can choose two indices i and j, such that i != j, and the sum of digits of the number nums[i] is equal to that of nums[j]. Return the maximum value of nums[i] + nums[j] that you can obtain over all possible indices i and j that satisfy the conditions.

Input: nums = [18,43,36,13,7]

Output: 54

Explanation: The pairs (i, j) that satisfy the conditions are:

- (0, 2), both numbers have a sum of digits equal to 9, and their sum is 18 + 36 = 54.

- (1, 4), both numbers have a sum of digits equal to 7, and their sum is 43 + 7 = 50.

So the maximum sum that we can obtain is 54.

```
int total = -1;
HashMap<Integer, Integer> map = new HashMap<>();
for(int i = 0;i<nums.length;i++){
    int sum = 0;
    int x = nums[i];
    while(x > 0){
        int r = x%10;
        sum += r;
        x = x/10;
    }
    if(!map.isEmpty() && map.containsKey(sum)){
        int y = nums[map.get(sum)];
        total = Math.max(total, y + nums[i]);
        if(nums[i] > y){
            map.put(sum, i);
        }
    }
    else{
        map.put(sum, i);
    }
}
return total;
```

3066. Minimum Operations to Exceed Threshold Value II - Medium

You are allowed to perform some operations on nums, where in a single operation, you can:

Select the two smallest integers x and y from nums. Remove x and y from nums.

Insert (min(x, y) * 2 + max(x, y)) at any position in the array.

Note that you can only apply the described operation if nums contains at least two elements. Return the minimum number of operations needed so that all elements of the array are greater than or equal to k.

Input: nums = [2,11,10,1,3], k = 10 Output: 2

In the first operation, we remove elements 1 and 2, then add 1 * 2 + 2 to nums. nums becomes equal to [4, 11, 10, 3].

In the second operation, we remove elements 3 and 4, then add 3 * 2 + 4 to nums. nums becomes equal to [10, 11, 10].

At this stage, all the elements of nums are greater than or equal to 10 so we can stop.

It can be shown that 2 is the minimum number of operations needed so that all elements of the array are greater than or equal to 10.

```
    int count = 0;
    PriorityQueue<Long> minHeap = new PriorityQueue<>();
    for(int i : nums){
        minHeap.add( (long) i);
    }
    while(minHeap.peek() < k && minHeap.size() >= 2){
        long x = minHeap.poll();
        long y = minHeap.poll();

        long min = Math.min(x,y);
        long max = Math.max(x,y);
        minHeap.add(min*2 + max);
        count++;
    }

    return count;
}
```

1352. Product of the Last K Numbers - Medium

Design an algorithm that accepts a stream of integers and retrieves the product of the last k integers of the stream.

Implement the ProductOfNumbers class:

ProductOfNumbers() Initializes the object with an empty stream.

void add(int num) Appends the integer num to the stream.

int getProduct(int k) Returns the product of the last k numbers in the current list. You can assume that always the current list has at least k numbers.

The test cases are generated so that, at any time, the product of any contiguous sequence of numbers will fit into a single 32-bit integer without overflowing.

Input

["ProductOfNumbers","add","add","add","add","add","getProduct","getProduct","getProduct","add","getProduct"]

[[],[3],[0],[2],[5],[4],[2],[3],[4],[8],[2]]

Output

[null,null,null,null,null,null,20,40,0,null,32]

Explanation

ProductOfNumbers productOfNumbers = new ProductOfNumbers();

productOfNumbers.add(3); // [3]

productOfNumbers.add(0); // [3,0]

productOfNumbers.add(2); // [3,0,2]

productOfNumbers.add(5); // [3,0,2,5]

productOfNumbers.add(4); // [3,0,2,5,4]

productOfNumbers.getProduct(2); // return 20. The product of the last 2 numbers is 5 * 4 = 20

productOfNumbers.getProduct(3); // return 40. The product of the last 3 numbers is 2 * 5 * 4 = 40

productOfNumbers.getProduct(4); // return 0. The product of the last 4 numbers is 0 * 2 * 5 * 4 = 0

productOfNumbers.add(8); // [3,0,2,5,4,8]

productOfNumbers.getProduct(2); // return 32. The product of the last 2 numbers is 4 * 8 = 32

```java
ArrayList<Integer> arr = new ArrayList<>();
int prod = 1;
public ProductOfNumbers() {}
public void add(int num) {
    if(num == 0){
        arr = new ArrayList<>();
        prod = 1;
        return;
    }
    prod = prod * num;
    arr.add(prod);
}
public int getProduct(int k) {
    if(arr.size() < k) return 0;

    int ans = arr.get(arr.size()-1);
    if(k == arr.size()) return ans;
    return ans / arr.get(arr.size() - 1 - k);
}
```

1780. Check if Number is a Sum of Powers of Three - Medium

Given an integer n, return true if it is possible to represent n as the sum of distinct powers of three. Otherwise, return false.

An integer y is a power of three if there exists an integer x such that y == 3x.

Input: n = 12

Output: true

Explanation: 12 = 31 + 32

Take an arraylist, add the powers of 3 into it < n and then

```java
ublic boolean checkSum(List<Integer> list, int sum, int n){
  if(sum == 0) return true;

  if(n == 0) return false;

  if(list.get(n-1) > sum){
      return checkSum(list, sum , n-1 );
  }

      return checkSum(list, sum - list.get(n-1), n-1) || checkSum(list, sum, n-1);
```

2965. Find Missing and Repeated Values - Easy

You are given a 0-indexed 2D integer matrix grid of size n * n with values in the range [1, n2]. Each integer appears exactly once except a which appears twice and b which is missing. The task is to find the repeating and missing numbers a and b.

Return a 0-indexed integer array ans of size 2 where ans[0] equals to a and ans[1] equals to b.

Example 1:

Input: grid = [[1,3],[2,2]]

Output: [2,4]

Explanation: Number 2 is repeated and number 4 is missing so the answer is [2,4].

```java
public int[] findMissingAndRepeatedValues(int[][] grid) {
    HashSet<Integer> set = new HashSet<>();
    int a = -1;
    int sum = 0;
    for(int arr [] : grid){
        for(int i : arr){
            if(set.contains(i)){
                a = i;
                continue;
            }
            set.add(i);
            sum += i;
        }
    }
    int n = grid.length * grid.length;
    int x  = n * (n+1)/2;
    return new int []{a, x - sum};
```

2523. Closest Prime Numbers in Range - Medium

Given two positive integers left and right, find the two integers num1 and num2 such that:

left <= num1 < num2 <= right .

Both num1 and num2 are prime numbers.

num2 - num1 is the minimum amongst all other pairs satisfying the above conditions.

Return the positive integer array ans = [num1, num2]. If there are multiple pairs satisfying these conditions, return the one with the smallest num1 value. If no such numbers exist, return [-1, -1].

Input: left = 10, right = 19

Output: [11,13]

Explanation: The prime numbers between 10 and 19 are 11, 13, 17, and 19.

The closest gap between any pair is 2, which can be achieved by [11,13] or [17,19].

Since 11 is smaller than 17, we return the first pair.

#Use Sieve of Eratosthenes - 1st step

```
        }
    for(int i = left; i<= right;i++){
        if(primes[i] == true) arr.add(i);
    }
    for(int i = arr.size() - 1; i>0;i-- ){
        int diff = arr.get(i) - arr.get(i-1);
        if(min_diff >= diff){
            min_diff = diff;
            arr1[0] = arr.get(i-1);
            arr1[1] = arr.get(i);
        }
    }
    return arr1;

}
```

2379. Minimum Recolors to Get K Consecutive Black Blocks - Easy

You are given a 0-indexed string blocks of length n, where blocks[i] is either 'W' or 'B', representing the color of the ith block. The characters 'W' and 'B' denote the colors white and black, respectively.

You are also given an integer k, which is the desired number of consecutive black blocks.

In one operation, you can recolor a white block such that it becomes a black block.

Return the minimum number of operations needed such that there is at least one occurrence of k consecutive black blocks.

Input: blocks = "WBBWWBBWBW", k = 7

Output: 3

Explanation:

One way to achieve 7 consecutive black blocks is to recolor the 0th, 3rd, and 4th blocks so that blocks = "BBBBBBBWBW".

It can be shown that there is no way to achieve 7 consecutive black blocks in less than 3 operations.

Therefore, we return 3.

```
    int min = Integer.MAX_VALUE;
    int sum = 0;
    int j = 0;
    for(int i = 0;i<blocks.length();i++){
        if(i < k ){

            if(blocks.charAt(i) == 'W') sum++;
            if(i == k - 1){
                min = Math.min(min,sum);
            }
        }
        else{
            if(blocks.charAt(j) == 'W') sum--;
            j++;
            if(blocks.charAt(i) == 'W') sum++;
            min = Math.min(min, sum);
        }

    }
    return min;
}
```

\

3208. Alternating Groups II - Easy

There is a circle of red and blue tiles. You are given an array of integers colors and an integer k. The color of tile i is represented by colors[i]:

colors[i] == 0 means that tile i is red.

colors[i] == 1 means that tile i is blue.

An alternating group is every k contiguous tiles in the circle with alternating colors (each tile in the group except the first and last one has a different color from its left and right tiles).

Return the number of alternating groups.

Note that since colors represents a circle, the first and the last tiles are considered to be next to each other.

Input: colors = [0,1,0,1,0], k = 3

Output: 3

```java
public int numberOfAlternatingGroups(int[] colors, int k) {
    int n = colors.length;
    int count = 0;
    int left = 0;

    for (int right = 1; right < n + k - 1; right++) {
        if(colors[right%n] == colors[(right - 1)%n]) left = right;

        if(right - left + 1 == k){
            count++;
            left++;
        }

        }
        return count;
```

1358. Number of Substrings Containing All Three Characters - Medium

Given a string s consisting only of characters a, b and c.

Return the number of substrings containing at least one occurrence of all these characters a, b and c.

Input: s = "abcabc"

Output: 10

Explanation: The substrings containing at least one occurrence of the characters a, b and c are "abc", "abca", "abcab", "abcabc", "bca", "bcab", "bcabc", "cab", "cabc" and "abc" (again).

```
int [] abc = new int[3];
Arrays.fill(abc,-1);
int right = 0; int count = 0;
char [] ch = s.toCharArray();
while(right < ch.length){
    abc[ch[right] - 'a'] = right;
    int min = Integer.MAX_VALUE;
    for(int i = 0; i< 3; i++){
        min = Math.min(min, abc[i]);
    }

    count += (min + 1);
    right++;
}
return count;
```

3355. Zero Array Transformation I - Medium

You are given an integer array nums of length n and a 2D array queries, where queries[i] = [li, ri].

For each queries[i]:

Select a subset of indices within the range [li, ri] in nums.

Decrement the values at the selected indices by 1.

A Zero Array is an array where all elements are equal to 0.

Return true if it is possible to transform nums into a Zero Array after processing all the queries sequentially, otherwise return false.

Example 1:

Input: nums = [1,0,1], queries = [[0,2]]

Output: true

For i = 0:

Select the subset of indices as [0, 2] and decrement the values at these indices by 1.

The array will become [0, 0, 0], which is a Zero Array.

```
  int line [] = new int[nums.length + 1];

  for(int [] i : queries){
    line[i[0]]++;
    line[i[1] + 1]--;
  }

  for(int i = 1;i<line.length;i++){
    line[i] += line[i-1];
  }

  for(int i =0;i<nums.length;i++){
    if(line[i] < nums[i]) return false;
  }

  return true;
}
```

2560. House Robber IV - Medium

There are several consecutive houses along a street, each of which has some money inside. There is also a robber, who wants to steal money from the homes, but he refuses to steal from adjacent homes. The capability of the robber is the maximum amount of money he steals from one house of all the houses he robbed.

You are given an integer array nums representing how much money is stashed in each house. More formally, the ith house from the left has nums[i] dollars.

You are also given an integer k, representing the minimum number of houses the robber will steal from. It is always possible to steal at least k houses.

Return the minimum capability of the robber out of all the possible ways to steal at least k houses.

Input: nums = [2,3,5,9], k = 2

Output: 5

Explanation:

There are three ways to rob at least 2 houses:

- Rob the houses at indices 0 and 2. Capability is max(nums[0], nums[2]) = 5.

- Rob the houses at indices 0 and 3. Capability is max(nums[0], nums[3]) = 9.

- Rob the houses at indices 1 and 3. Capability is max(nums[1], nums[3]) = 9.

Therefore, we return min(5, 9, 9) = 5.

```
    int l = Integer.MAX_VALUE;
    int r = Integer.MIN_VALUE;
    for(int i : nums){
        if(l > i) l = i;
        if(r < i) r = i;
    }
    while(l <= r){
        int mid = l + (r-l)/2;

        if(check(nums, mid, k)) r = mid - 1;

        else l = mid + 1;

    }
    return l;
}
```

```
public boolean check(int [] arr, int mid, int k){
    int i = 0;
    int count = 0;

    while(i < arr.length){

        if(arr[i] <= mid){
            count++;
            i+=2;
        }
        else i++;
    }
    return count >= k;

}
```

2594. Minimum Time to Repair Cars - Medium

You are given an integer array ranks representing the ranks of some mechanics. ranksi is the rank of the ith mechanic. A mechanic with a rank r can repair n cars in r * n2 minutes.

You are also given an integer cars representing the total number of cars waiting in the garage to be repaired.

Return the minimum time taken to repair all the cars.

Input: ranks = [4,2,3,1], cars = 10

Output: 16

Explanation:

- The first mechanic will repair two cars. The time required is 4 * 2 * 2 = 16 minutes.

- The second mechanic will repair two cars. The time required is 2 * 2 * 2 = 8 minutes.

- The third mechanic will repair two cars. The time required is 3 * 2 * 2 = 12 minutes.

- The fourth mechanic will repair four cars. The time required is 1 * 4 * 4 = 16 minutes.

It can be proved that the cars cannot be repaired in less than 16 minutes.

```java
public long repairCars(int[] ranks, int cars) {
    long l = 1;
    long r = (long) ranks[0] * cars * cars;

    while( l <= r){
        long mid = l + (r-l)/2;
        if(check(ranks, mid , cars)){
            r = mid - 1;
        }
        else{
            l = mid + 1;
        }
    }
    return l;
}
public boolean check(int [] arr, long mid, int cars){
    long cur = 0;
    for(int i : arr){
        cur += (long) (Math.sqrt((1.0 * mid)/i));
    }
    return (cur >= cars);
}
```

2206. Divide Array Into Equal Pairs

You are given an integer array nums consisting of 2 * n integers.

You need to divide nums into n pairs such that:

Each element belongs to exactly one pair.

The elements present in a pair are equal.

Return true if nums can be divided into n pairs, otherwise return false.

```
int[] freq = new int[501];

for(int i = 0; i<nums.length; i++){
    freq[nums[i]]++;
}

for(int i = 0; i<501; i++){
    if(freq[i]%2 != 0){
        return false;
    }
}

return true;
```

2401. Longest Nice Subarray - Medium

You are given an array nums consisting of positive integers.

We call a subarray of nums nice if the bitwise AND of every pair of elements that are in different positions in the subarray is equal to 0.

Return the length of the longest nice subarray.

A subarray is a contiguous part of an array.

Note that subarrays of length 1 are always considered nice.

Input: nums = [1,3,8,48,10]

Output: 3

Explanation: The longest nice subarray is [3,8,48]. This subarray satisfies the conditions:

- 3 AND 8 = 0.

- 3 AND 48 = 0. - 8 AND 48 = 0.

```
    int used = 0;
    int left = 0;
    int right = 0;
    int max = Integer.MIN_VALUE;
    for(right = 0; right < nums.length; right++){
        while( (used & nums[right]) != 0){
            used ^= nums[left];
            left++;
        }

        used |= nums[right];
        max = Math.max(max, right - left + 1);
    }

return max;
```

1976. Number of Ways to Arrive at Destination - Medium

You are in a city that consists of n intersections numbered from 0 to n - 1 with bi-directional roads between some intersections. The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between any two intersections.

You are given an integer n and a 2D integer array roads where roads[i] = [ui, vi, timei] means that there is a road between intersections ui and vi that takes timei minutes to travel. You want to know in how many ways you can travel from intersection 0 to intersection n - 1 in the shortest amount of time.

Return the number of ways you can arrive at your destination in the shortest amount of time. Since the answer may be large, return it modulo 109 + 7.

```
public int countPaths(int n, int[][] roads) {
    ArrayList<ArrayList<int []>> adj = new ArrayList<>();
    for(int i=0;i<n;i++) adj.add(new ArrayList<>());
    for(int i = 0;i<roads.length;i++){
        adj.get(roads[i][0]).add(new int [] { roads[i][1], roads[i][2]});
        adj.get(roads[i][1]).add(new int [] { roads[i][0], roads[i][2]});
    }
    PriorityQueue<long []> pq = new PriorityQueue<>( (a,b) -> Long.compare(a[0],b[0]));
    int mod = (int) 1e9+7;
    pq.offer(new long [] {0,0});
    long dist [] = new long[n];
    long ways[] = new long[n];
   Arrays.fill(dist, Long.MAX_VALUE);
    dist[0] = 0;
    ways[0] = 1;
    while(!pq.isEmpty()){
        long curr [] = pq.poll();
        long distt = curr[0];
        int node = (int) curr[1];
```

```
    for(int i [] : adj.get(node)){
        int nnode = i[0];
        int ndist = i[1];
        if(distt + ndist  < dist[nnode]){
            dist[nnode] = distt + ndist;
            ways[nnode] = ways[node];
            pq.offer(new long [] { distt + ndist, nnode});
        }
        else if ( ndist + distt == dist[nnode]){
            ways[nnode] = (ways[nnode] + ways[node])%mod;
        }
    }
}
return (int) ways[n-1];
```

3169. Count Days Without Meetings - Medium

You are given a positive integer days representing the total number of days an employee is available for work (starting from day 1). You are also given a 2D array meetings of size n where, meetings[i] = [start_i, end_i] represents the starting and ending days of meeting i (inclusive).

Return the count of days when the employee is available for work but no meetings are scheduled.

Note: The meetings may overlap.

Input: days = 10, meetings = [[5,7],[1,3],[9,10]]

Output: 2

There is no meeting scheduled on the 4th and 8th days.

```
Arrays.sort(meetings, (a,b) -> Integer.compare(a[0], b[0]));
int freedays = 0, lastend = 0;
for(int i [] : meetings){
    int s = i[0], e = i[1];

    if(s > lastend + 1) freedays += s - lastend - 1;

    lastend = Math.max(lastend, e);
}

return freedays + (days - lastend);
```

2033. Minimum Operations to Make a Uni-Value Grid - Medium

You are given a 2D integer grid of size m x n and an integer x. In one operation, you can add x to or subtract x from any element in the grid.

A uni-value grid is a grid where all the elements of it are equal.

Return the minimum number of operations to make the grid uni-value. If it is not possible, return -1.

Input: grid = [[2,4],[6,8]], x = 2 Output: 4

Explanation: We can make every element equal to 4 by doing the following:

- Add x to 2 once.

- Subtract x from 6 once.

- Subtract x from 8 twice.

A total of 4 operations were used.

```
List<Integer> arr = new ArrayList<>();
//You can define arr[grid.length * grid[0].length] also to reduce space
//complexity
 int k = 0;
 for(int i [] : grid){
     for(int ii : i){
         arr.add(ii); }}
 Collections.sort(arr);
 int median = arr.get(arr.size()/2);
 for(int i : arr){
    int y = Math.abs(i - median);
    if(y%x !=0) return -1;
 }
 int count = 0;
 for(int i : arr){
     count += Math.abs((i - median)/x);
 }
 return count;
```

2780. Minimum Index of a Valid Split - Medium

An element x of an integer array arr of length m is dominant if more than half the elements of arr have a value of x.

You are given a 0-indexed integer array nums of length n with one dominant element.

You can split nums at an index i into two arrays nums[0, ..., i] and nums[i + 1, ..., n - 1], but the split is only valid if: 0 <= i < n - 1 and nums[0, ..., i], and nums[i + 1, ..., n - 1] have the same dominant element.

Here, nums[i, ..., j] denotes the subarray of nums starting at index i and ending at index j, both ends being inclusive. Particularly, if j < i then nums[i, ..., j] denotes an empty subarray.

Return the minimum index of a valid split. If no valid split exists, return -1.

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

Explanation: We can split the array at index 2 to obtain arrays [1,2,2] and [2].

In array [1,2,2], element 2 is dominant since it occurs twice in the array and 2 * 2 > 3.

In array [2], element 2 is dominant since it occurs once in the array and 1 * 2 > 1.

Both [1,2,2] and [2] have the same dominant element as nums, so this is a valid split.

It can be shown that index 2 is the minimum index of a valid split.

```
int can = 0;
int count = 0;
for(int i : nums){
    if(count == 0){
        can = i;
    }
    if(can == i) count++;
    else count--;
}
count = 0;
for(int i : nums)
if( i == can) count++;
int f1 = 0;
for(int i = 0; i< nums.size(); i++){
    if(nums.get(i) == can){ f1++;
    int f2 = count - f1;
    boolean len1 = (f1 > (i+1)/2) ? true : false;
    boolean len2 = (f2 > (nums.size() - (i+1))/2) ? true : false;
    if(len1 && len2) return i;
    }
}
return -1;
```

2873. Maximum Value of an Ordered Triplet I - Easy

You are given a 0-indexed integer array nums.

Return the maximum value over all triplets of indices (i, j, k) such that i < j < k. If all such triplets have a negative value, return 0.

The value of a triplet of indices (i, j, k) is equal to (nums[i] - nums[j]) * nums[k].

Input: nums = [12,6,1,2,7]

Output: 77

Explanation: The value of the triplet (0, 2, 4) is (nums[0] - nums[2]) * nums[4] = 77.

It can be shown that there are no ordered triplets of indices with a value greater than 77.

```
int maxLeft = nums[0];
int maxDiff = Integer.MIN_VALUE;
long result = 0;

for(int j = 1; j<nums.length - 1; j++){
  maxDiff = Math.max(maxDiff, maxLeft - nums[j]);
  maxLeft = Math.max(maxLeft, nums[j]);


  long product = (long) maxDiff*nums[j+1];
  result = Math.max(result, product);
}

return result;
```

1123. Lowest Common Ancestor of Deepest Leaves - Medium

Given the root of a binary tree, return the lowest common ancestor of its deepest leaves.

Recall that:

The node of a binary tree is a leaf if and only if it has no children

The depth of the root of the tree is 0. if the depth of a node is d, the depth of each of its children is d + 1.

The lowest common ancestor of a set S of nodes, is the node A with the largest depth such that every node in S is in the subtree with root A.

Input: root = [3,5,1,6,2,0,8,null,null,7,4]

Output: [2] (2->7->4)

Explanation: We return the node with value 2, colored in yellow in the diagram.

The nodes coloured in blue are the deepest leaf-nodes of the tree.

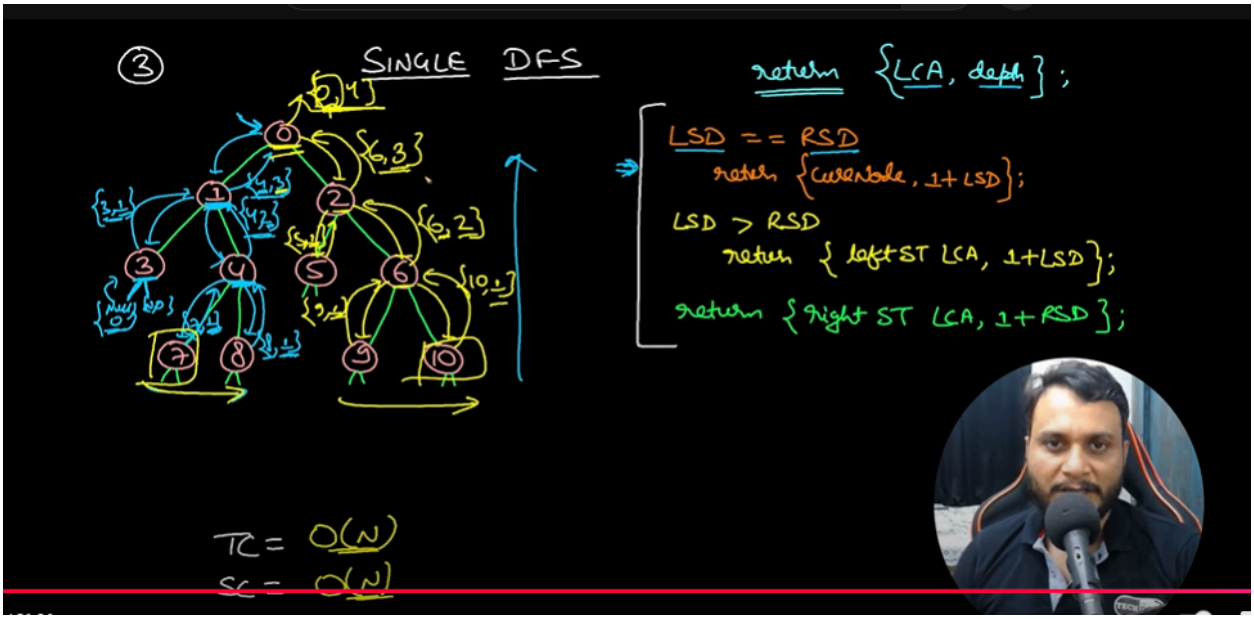
Note that nodes 6, 0, and 8 are also leaf nodes, but the depth of them is 2, but the depth of nodes 7 and 4 is 3.

Approach 1 : Common path method

```
class Solution {
    //find the depth
    private int findMaxDepth(TreeNode root){
        if(root == null) return -1;
        return 1 + Math.max(findMaxDepth(root.left), findMaxDepth(root.right));
    }
    //To get the deepest node paths from root
    public void findpath( TreeNode root, int depth, int maxDepth, ArrayList<TreeNode> path, List<List<TreeNode>> paths){

        if(root == null) return;
        path.add(root);

        if(root.left == null && root.right == null && depth == maxDepth) paths.add(new ArrayList<TreeNode>(path));

        findpath(root.left, depth + 1, maxDepth, path, paths);
        findpath(root.right, depth + 1, maxDepth, path, paths);

        path.remove(path.size() - 1);
    }
```

```
    //To get the last common node between all the paths
    public TreeNode getDeepest(List<List<TreeNode>> paths){
        TreeNode lca = null;
        for(int i = 0; ; i++){
            TreeNode curr = null;
            for(List<TreeNode> path : paths){
                if(i >= path.size()) return lca;
                if(curr == null) curr = path.get(i);
                else if(curr != path.get(i)) return lca;
            }
            lca = curr;
        }

    }

    public TreeNode lcaDeepestLeaves(TreeNode root) {
      int maxDepth = findMaxDepth(root);
      List<List<TreeNode>> paths = new ArrayList<>();
      findpath(root, 0, maxDepth, new ArrayList<TreeNode>(), paths );
      return getDeepest(paths);
    }
}
```

Approach 2: Reverse DFS




```java
static class Pair{
    TreeNode node;
    int depth;

    Pair(TreeNode node, int depth){
        this.node = node;
        this.depth = depth;
    }
}

public Pair findDeepestLeaves(TreeNode curr){
    if(curr == null) return new Pair(null, 0);

    Pair left = findDeepestLeaves(curr.left);
    Pair right = findDeepestLeaves(curr.right);

    if(left.depth == right.depth) return new Pair(curr, 1 + left.depth);
    else if(left.depth > right.depth) return new Pair(left.node, 1 + left.depth);

    else return new Pair(right.node, 1 + right.depth);
}
public TreeNode lcaDeepestLeaves(TreeNode root) {
  Pair p = findDeepestLeaves(root);
  return p.node;
}
```

1863. Sum of All Subset XOR Totals - Easy

The XOR total of an array is defined as the bitwise XOR of all its elements, or 0 if the array is empty.

For example, the XOR total of the array [2,5,6] is 2 XOR 5 XOR 6 = 1.

Given an array nums, return the sum of all XOR totals for every subset of nums.

Note: Subsets with the same elements should be counted multiple times.

An array a is a subset of an array b if a can be obtained from b by deleting some (possibly zero) elements of b.

Input: nums = [1,3]

Output: 6

Explanation: The 4 subsets of [1,3] are:

- The empty subset has an XOR total of 0.

- [1] has an XOR total of 1.

- [3] has an XOR total of 3.

- [1,3] has an XOR total of 1 XOR 3 = 2.

0 + 1 + 3 + 2 = 6

```java
public int Xor(ArrayList<Integer> path){
    int i = 0;
    for(int x : path){
        i = i ^ x;
    }
    return i;
}
public int countSubsets( int [] nums, int x, ArrayList<Integer> path){
    int total = Xor(path);

    for(int i = x; i < nums.length; i++ ){
        path.add(nums[i]);
        total += countSubsets(nums, i + 1, path);
        path.remove(path.size() - 1);
    }
    return total;
}
public int subsetXORSum(int[] nums) {

    return countSubsets(nums, 0, new ArrayList<>());
```

3396. Minimum Number of Operations to Make Elements in Array Distinct - Easy

You are given an integer array nums. You need to ensure that the elements in the array are distinct. To achieve this, you can perform the following operation any number of times:

Remove 3 elements from the beginning of the array. If the array has fewer than 3 elements, remove all remaining elements.

Note that an empty array is considered to have distinct elements. Return the minimum number of operations needed to make the elements in the array distinct.

Example 1:

Input: nums = [1,2,3,4,2,3,3,5,7]

Output: 2

Explanation:

In the first operation, the first 3 elements are removed, resulting in the array [4, 2, 3, 3, 5, 7].In the second operation, the next 3 elements are removed, resulting in the array [3, 5, 7], which has distinct elements.

Therefore, the answer is 2.

```
List<Integer> list = new ArrayList<>();
for(int i : nums) list.add(i);
int count = 0;
while(true){
    Set<Integer> seen = new HashSet<>();
    boolean hasdup = false;

    for(int i : list){
        if(!seen.add(i)){
            hasdup = true;
            break;
        }
    }
    if(!hasdup) break;
    int removelen = Math.min(3, list.size());
    for(int j = 0;j<removelen;j++){
        list.remove(0);
    }
    count++;
}
return count;
```

Greedy

```java
public int minimumOperations(int[] nums) {
    Set<Integer> set = new HashSet<>();

    for(int i = nums.length - 1; i>=0; i--){
        if(!set.add(nums[i])){
            return i/3 + 1;
        }
        set.add(nums[i]);
    }
    return 0;
}
```

1922. Count Good Numbers Medium

A digit string is good if the digits (0-indexed) at even indices are even and the digits at odd indices are prime (2, 3, 5, or 7). For example, "2582" is good because the digits (2 and 8) at even positions are even and the digits (5 and 2) at odd positions are prime. However, "3245" is not good because 3 is at an even index but is not even. Given an integer n, return the total number of good digit strings of length n. Since the answer may be large, return it modulo 109 + 7. A digit string is a string consisting of digits 0 through 9 that may contain leading zeros.

Input: n = 1

Output: 5

Explanation: The good numbers of length 1 are "0", "2", "4", "6", "8".

Example 2:

Input: n = 4

Output: 400

Example 3:

Input: n = 50

Output: 564908303

```java
private long mod = 1_000_000_007;
public int countGoodNumbers(long n) {
  long even = (n + 1)/2;
  long odd = n/2;
  return (int) ((pow(5,even) * pow(4,odd))%mod);
}
public long pow(int x, long n){
    if(n == 0) return 1;
    if(x == 0) return 0;

    long r = pow(x, n/2);
    long res = (r * r)%mod;

    if(n %2 == 1){
        res = (res * (long) x) % mod;
    }
    return res; // return x * pow(x,n-1);
}
```

1534. Count Good Triplets - Easy

Given an array of integers arr, and three integers a, b and c. You need to find the number of good triplets.

A triplet (arr[i], arr[j], arr[k]) is good if the following conditions are true:

0 <= i < j < k < arr.length

|arr[i] - arr[j]| <= a

|arr[j] - arr[k]| <= b

|arr[i] - arr[k]| <= c

Where |x| denotes the absolute value of x.

Return the number of good triplets.

Input: arr = [3,0,1,1,9,7], a = 7, b = 2, c = 3

Output: 4

Explanation: There are 4 good triplets: [(3,0,1), (3,0,1), (3,1,1), (0,1,1)].

O(n^3) time complexity

```java
int count = 0;
for(int i = 0; i<arr.length-2;i++){
    for(int j = i + 1; j<arr.length-1;j++){
        for(int k = j+1;k<arr.length;k++){
            int ax = Math.abs(arr[i] - arr[j]);
            int bx = Math.abs(arr[j] - arr[k]);
            int cx = Math.abs(arr[i] - arr[k]);

            if(ax <= a && bx <= b && cx <= c){
                count++;
            }
```

2537. Count the Number of Good Subarrays - Medium

Given an integer array nums and an integer k, return the number of good subarrays of nums.

A subarray arr is good if there are at least k pairs of indices (i, j) such that i < j and arr[i] == arr[j].

A subarray is a contiguous non-empty sequence of elements within an array.

Input: nums = [1,1,1,1,1], k = 10

Output: 1

Explanation: The only good subarray is the array nums itself.

Example 2:

Input: nums = [3,1,4,3,2,2,4], k = 2

Output: 4

Explanation: There are 4 different good subarrays:

- [3,1,4,3,2,2] that has 2 pairs.

- [3,1,4,3,2,2,4] that has 3 pairs.

- [1,4,3,2,2,4] that has 2 pairs.

- [4,3,2,2,4] that has 2 pairs.

```
long count = 0;
int pairs = 0;
int left = 0;
Map<Integer, Integer> map = new HashMap<>();
for(int i = 0;i<nums.length;i++){
   pairs += map.getOrDefault(nums[i],0);
   map.put(nums[i], map.getOrDefault(nums[i],0)+1);

   while( pairs>= k){
       map.put(nums[left], map.get(nums[left]) - 1);
       pairs -= map.get(nums[left]);
       left++;
   }
count+= left;
}
return count;
}
```

2176. Count Equal and Divisible Pairs in an Array - Easy

Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i * j) is divisible by k.

Input: nums = [3,1,2,2,2,1,3], k = 2

Output: 4

Explanation:

- nums[0] == nums[6], and 0 * 6 == 0, which is divisible by 2.

- nums[2] == nums[3], and 2 * 3 == 6, which is divisible by 2.

- nums[2] == nums[4], and 2 * 4 == 8, which is divisible by 2.

- nums[3] == nums[4], and 3 * 4 == 12, which is divisible by 2.

```
public int check(int []nums, int k, int i){
    if(i >= nums.length) return 0;
    int count = 0;
    for(int j = i+1; j<nums.length;j++){
        if(nums[i] == nums[j])
            if((i*j)%k == 0) count++;
    }

    return count + check(nums, k , i+1);
}
public int countPairs(int[] nums, int k) {
  return check(nums, k , 0);
```

38. Count and Say - Medium

countAndSay(1) = "1"

countAndSay(n) is the run-length encoding of countAndSay(n - 1).

Run-length encoding (RLE) is a string compression method that works by replacing consecutive identical characters (repeated 2 or more times) with the concatenation of the character and the number marking the count of the characters (length of the run). For example, to compress the string "3322251" we replace "33" with "23", replace "222" with "32", replace "5" with "15" and replace "1" with "11". Thus the compressed string becomes "23321511".

Given a positive integer n, return the nth element of the count-and-say sequence.

Example 1:

Input: n = 4

Output: "1211"

countAndSay(1) = "1"

countAndSay(2) = RLE of "1" = "11"

countAndSay(3) = RLE of "11" = "21"

countAndSay(4) = RLE of "21" = "1211"

```
if( n == 1) return "1";
if(n == 2) return "11";
String a = "1";
for(int i = 1;i<n;i++){
     String b = "";
     int count = 1;
     char cur = a.charAt(0);
    for(int j = 1; j<a.length(); j++){
        if(a.charAt(j) == cur){
            count++;
        }
        else {
        b += Integer.toString(count) + cur;
        cur = a.charAt(j);
        count = 1;

    }
    }
    b += Integer.toString(count) + cur;
    a = b;
}

return a;
```

2563. Count the Number of Fair Pairs - Medium

Given a 0-indexed integer array nums of size n and two integers lower and upper, return the number of fair pairs.

A pair (i, j) is fair if:

0 <= i < j < n, and

lower <= nums[i] + nums[j] <= upper

Input: nums = [0,1,7,4,4,5], lower = 3, upper = 6

Output: 6

Explanation: There are 6 fair pairs: (0,3), (0,4), (0,5), (1,3), (1,4), and (1,5).

Input: nums = [1,7,9,2,5], lower = 11, upper = 11

Output: 1

Explanation: There is a single fair pair: (2,3).

```java
public long count(int [] nums, int sum){
 long res = 0;

 for(int i = 0,j = nums.length -1; i<j; i++){
    while( i< j && nums[i] + nums[j] > sum) j--;

    res += j - i;
 }
 return res;
}
public long countFairPairs(int[] nums, int lower, int upper) {
    Arrays.sort(nums);
    return count(nums, upper) - count(nums, lower - 1);
}
```

```java
public int countlow(int [] nums, int l, int h, int sum){
    int low = l;
    int high = h;

    while(low <high){
        int mid = low + (high - low)/2;

        if(nums[mid] < sum) low = mid + 1;
        else high = mid;
    }
    return low;
}
   public int counthigh(int [] nums, int l, int h, int sum){
    int low = l;
    int high = h;

    while(low<high){
        int mid = low + (high - low)/2;

        if(nums[mid] <= sum) low = mid + 1;
        else high = mid;
    }
    return low;
}
   public long countFairPairs(int[] nums, int lower, int upper) {
    Arrays.sort(nums);
    long ans = 0;

    for(int i = 0;i+1 < nums.length; i++){
        int min = lower - nums[i];
        int max = upper - nums[i];
        int l = countlow(nums, i+1, nums.length,min);
        int h = counthigh(nums, i+1, nums.length, max);
        ans += (h - l);
    }
    return ans;
}
```

781. Rabbits in Forest - medium

There is a forest with an unknown number of rabbits. We asked n rabbits "How many rabbits have the same color as you?" and collected the answers in an integer array answers where answers[i] is the answer of the ith rabbit.

Given the array answers, return the minimum number of rabbits that could be in the forest.

Input: answers = [1,1,2]

Output: 5

Explanation:

The two rabbits that answered "1" could both be the same color, say red.

The rabbit that answered "2" can't be red or the answers would be inconsistent.

Say the rabbit that answered "2" was blue.

Then there should be 2 other blue rabbits in the forest that didn't answer into the array.

The smallest possible number of rabbits in the forest is therefore 5: 3 that answered plus 2 that didn't.

```
HashMap<Integer, Integer> map = new HashMap<>();
int rb = 0;
for(int i : answers){
    if(i == 0){
        rb++;
        continue;
    }
    map.put(i, map.getOrDefault(i, 0)+1);
    }
    for(Map.Entry<Integer, Integer> m : map.entrySet()){
        int k = m.getKey();
        int v = m.getValue();

        int groupsize = k + 1;
        int groups = (v+k)/groupsize;
        rb += groups*groupsize;
    }
    return rb;
}
```

1399. Count Largest Group - Easy

You are given an integer n.

Each number from 1 to n is grouped according to the sum of its digits.

Return the number of groups that have the largest size.

Input: n = 13

Output: 4

Explanation: There are 9 groups in total, they are grouped according sum of its digits of numbers from 1 to 13:

[1,10], [2,11], [3,12], [4,13], [5], [6], [7], [8], [9].

There are 4 groups with largest size.

```
class Solution {
public int countSum(int x){
    int ans = 0;
    while(x>0){
        int res = x%10;
        ans += res;
        x = x/10;
    }
    return ans;
}
public int countLargestGroup(int n) {
   int [] sums = new int[37];
   for(int i = 1; i<= n; i++){
    sums[countSum(i)]++;
   }
   int max= 0, count = 0;
   for(int i : sums){
    if(i > max){
        max = i;
        count = 1;
    }
    else if(i == max){
        count++;
    }
   }
```

2962. Count Subarrays Where Max Element Appears at Least K Times - MEDIUM

You are given an integer array nums and a positive integer k.

Return the number of subarrays where the maximum element of nums appears at least k times in that subarray.

A subarray is a contiguous sequence of elements within an array.

Example 1:

Input: nums = [1,3,2,3,3], k = 2

Output: 6

Explanation: The subarrays that contain the element 3 at least 2 times are: [1,3,2,3], [1,3,2,3,3], [3,2,3], [3,2,3,3], [2,3,3] and [3,3].

Example 2:

Input: nums = [1,4,2,1], k = 3

Output: 0

Explanation: No subarray contains the element 4 at least 3 times.

```java
public long countSubarrays(int[] nums, int k) {
    // HashMap<Integer,Integer> map = new HashMap<>();
    // int max = Integer.MIN_VALUE;
    // for(int i : nums) max = Math.max(max, i);    ---- HASHMAP ---

    // long res = 0;
    // int left = 0;
    // for(int i = 0;i<nums.length;i++){
    //     map.put(nums[i], map.getOrDefault(nums[i],0)+1);

    //     while(left < nums.length && map.getOrDefault(max, 0) >= k){
    //         map.put(nums[left], map.get(nums[left])-1);
    //         left++;
    //     }

    //     res += (long) left;
    // }
    // return res;
    long res = 0;
    int left = 0;
    int maxocc = 0;
     int max = Integer.MIN_VALUE;
    for(int i : nums) max = Math.max(max, i);   //SLIDING WINDOW
    for(int i = 0;i<nums.length;i++){
        if(nums[i] == max) maxocc++;
        while(maxocc >= k){
            if(nums[left] == max) maxocc--;
            left++;
        }
        res += (long) left;
    }
    return res;
}
```

2302. Count Subarrays With Score Less Than K - hard

The score of an array is defined as the product of its sum and its length.

For example, the score of [1, 2, 3, 4, 5] is (1 + 2 + 3 + 4 + 5) * 5 = 75.

Given a positive integer array nums and an integer k, return the number of non-empty subarrays of nums whose score is strictly less than k.

A subarray is a contiguous sequence of elements within an array.

Example 1:

Input: nums = [2,1,4,3,5], k = 10

Output: 6

Explanation:

The 6 subarrays having scores less than 10 are:

- [2] with score 2 * 1 = 2.

- [1] with score 1 * 1 = 1.

- [4] with score 4 * 1 = 4.

- [3] with score 3 * 1 = 3.

- [5] with score 5 * 1 = 5.

- [2,1] with score (2 + 1) * 2 = 6.

Note that subarrays such as [1,4] and [4,3,5] are not considered because their scores are 10 and 36 respectively, while we need scores strictly less than 10.

```java
public long countSubarrays(int[] nums, long k) {
    int left = 0;
    long sum = 0;
    long count = 0;

    for(int right = 0; right < nums.length; right++){
     sum += nums[right];

     while( sum * (right - left + 1) >= k){
         sum -= nums[left++];
     }
     count += (right - left + 1);
    }
    return count;
}
```

1295. Find Numbers with Even Number of Digits

Given an array nums of integers, return how many of them contain an even number of digits.

Example 1:

Input: nums = [12,345,2,6,7896]

Output: 2

```java
public int findNumbers(int[] nums) {
    return find_count(nums, 0);
}
public int find_count(int [] nums, int i){
    if(i == nums.length) return 0;
    int count = 0;
    int x = (int) Math.log10(nums[i]) + 1;
    if(x%2 == 0) count++;
    return count + find_count(nums, i+1);
}
```

2071. Maximum Number of Tasks You Can Assign - HARD

You have n tasks and m workers. Each task has a strength requirement stored in a 0-indexed integer array tasks, with the ith task requiring tasks[i] strength to complete. The strength of each worker is stored in a 0-indexed integer array workers, with the jth worker having workers[j] strength. Each worker can only be assigned to a single task and must have a strength greater than or equal to the task's strength requirement (i.e., workers[j] >= tasks[i]).

Additionally, you have pills magical pills that will increase a worker's strength by strength. You can decide which workers receive the magical pills, however, you may only give each worker at most one magical pill.

Given the 0-indexed integer arrays tasks and workers and the integers pills and strength, return the maximum number of tasks that can be completed.

Input: tasks = [3,2,1], workers = [0,3,3], pills = 1, strength = 1

Output: 3

Explanation:

We can assign the magical pill and tasks as follows:

- Give the magical pill to worker 0.

- Assign worker 0 to task 2 (0 + 1 >= 1)

- Assign worker 1 to task 1 (3 >= 2)

- Assign worker 2 to task 0 (3 >= 3)

```java
public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {
    Arrays.sort(tasks);
    Arrays.sort(workers);

    int l = 0;
    int h  = Math.min(tasks.length, workers.length);
    int assigned = 0;

    while(l <= h){
        int mid = l + (h - l)/2;
        if(canAssign(mid, tasks, workers, pills, strength)){
            assigned = mid;
            l = mid + 1;
        }
        else h = mid - 1;
    }
return assigned;
```

```java
public boolean canAssign(int mid, int [] tasks,int [] workers, int p, int s){
   Deque<Integer> dq = new ArrayDeque<>();
   int w = workers.length - 1;
   int freep = p;

   for(int t = mid - 1; t >= 0; t--){
    int task = tasks[t];

    if(!dq.isEmpty() && dq.peekFirst() >= task){
        dq.pollFirst();
    }
    else if( w>= 0 && workers[w] >= task){
        w--;
    }
    else {
        while( w>= 0 && workers[w] + s >= task){
            dq.addLast(workers[w--]);
        }
        if( dq.isEmpty() || p <= 0) return false;

        dq.pollLast();
        p--;
    }
   }
        return true;
}
```

838. Push Dominoes - Medium

There are n dominoes in a line, and we place each domino vertically upright. In the beginning, we simultaneously push some of the dominoes either to the left or to the right.

After each second, each domino that is falling to the left pushes the adjacent domino on the left. Similarly, the dominoes falling to the right push their adjacent dominoes standing on the right.

When a vertical domino has dominoes falling on it from both sides, it stays still due to the balance of the forces.

For the purposes of this question, we will consider that a falling domino expends no additional force to a falling or already fallen domino.

You are given a string dominoes representing the initial state where:

dominoes[i] = 'L', if the ith domino has been pushed to the left,

dominoes[i] = 'R', if the ith domino has been pushed to the right, and

dominoes[i] = '.', if the ith domino has not been pushed.

Return a string representing the final state.

Example 1:

Input: dominoes = "RR.L"

Output: "RR.L"

Explanation: The first domino expends no additional force on the second domino.

```
public void doublepush(int last_r, int pos, StringBuilder s){
    while(last_r < pos){
        s.setCharAt(last_r++,'R');
        s.setCharAt(pos--,'L');
    }
}
public void leftpush(int s , int e, StringBuilder st){
    while(s <= e){
        st.setCharAt(s++, 'L');
    }
}
public void rightpush(int last_r, int pos, StringBuilder s){
    while(last_r <= pos) s.setCharAt(last_r++, 'R');
}
```

```java
public String pushDominoes(String dominoes) {
StringBuilder res = new StringBuilder(dominoes);

int last_r = -1;
int last_l = -1;

for(int i = 0;i < dominoes.length(); i++){
    if(dominoes.charAt(i) == 'L'){
        if(last_r > last_l){
            doublepush(last_r, i, res);
        }
        else if( last_l > last_r || last_l == -1){
            leftpush(last_l + 1, i, res);
        }
        last_l = i;
    }
    else if(dominoes.charAt(i) == 'R'){
        if(last_r > last_l){
            rightpush(last_r, i, res);
        }
        last_r = i;
    }
}
if(last_r > last_l) rightpush(last_r, dominoes.length() -1 , res);
return res.toString();
}
```

1128. Number of Equivalent Domino Pairs

Given a list of dominoes, dominoes[i] = [a, b] is equivalent to dominoes[j] = [c, d] if and only if either (a == c and b == d), or (a == d and b == c) - that is, one domino can be rotated to be equal to another domino.

Return the number of pairs (i, j) for which 0 <= i < j < dominoes.length, and dominoes[i] is equivalent to dominoes[j].

Input: dominoes = [[1,2],[2,1],[3,4],[5,6]]

Output: 1

Input: dominoes = [[1,2],[1,2],[1,1],[1,2],[2,2]]

Output: 3

```
int max [] = new int[100];
int res = 0;

for(int i [] : dominoes){
    int val = i[0] < i[1] ? i[0]*10 + i[1] : i[1] * 10 + i[0];
    res += max[val]++;
}
return res;
```

1920. Build Array from Permutation

Given a zero-based permutation nums (0-indexed), build an array ans of the same length where ans[i] = nums[nums[i]] for each 0 <= i < nums.length and return it.

A zero-based permutation nums is an array of distinct integers from 0 to nums.length - 1 (inclusive).

Input: nums = [0,2,1,5,3,4]

Output: [0,1,2,4,5,3]

Explanation: The array ans is built as follows:

ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]

= [nums[0], nums[2], nums[1], nums[5], nums[3], nums[4]]

= [0,1,2,4,5,3]

```
public int[] buildArray(int[] nums) {
   for(int i = 0; i< nums.length; i++){
    nums[i] += ( 1020 * (nums[nums[i]] % 1020));
   }
   for(int i=0;i<nums.length;i++) nums[i] /= 1020;

   return nums;
}
```

O(1) space complexity

3341. Find Minimum Time to Reach Last Room I - MEDIUM

There is a dungeon with n x m rooms arranged as a grid.

You are given a 2D array moveTime of size n x m, where moveTime[i][j] represents the minimum time in seconds when you can start moving to that room. You start from the room (0, 0) at time t = 0 and can move to an adjacent room. Moving between adjacent rooms takes exactly one second.

Return the minimum time to reach the room (n - 1, m - 1).

Two rooms are adjacent if they share a common wall, either horizontally or vertically.

Input: moveTime = [[0,4],[4,4]]

Output: 6

Explanation:

The minimum time required is 6 seconds.

At time t == 4, move from room (0, 0) to room (1, 0) in one second.

At time t == 5, move from room (1, 0) to room (1, 1) in one second.

Dijkstra → BFS + HEAP

```
private static final int [] dir = { -1, 0, 1 , 0, -1};

static class Node {
    int r;
    int c;
    int time;

    Node(int r, int c, int time){
        this.r = r;
        this.c = c;
        this.time = time;
    }
}

private  boolean isValid(int r, int c, int m, int n){
    return ( r>= 0 && r< m && c>= 0 && c<n);
}

public int minTimeToReach(int[][] moveTime) {
    int m = moveTime.length;
    int n = moveTime[0].length;

    PriorityQueue<Node> pq = new PriorityQueue<>(
        (p1,p2) -> Integer.compare(p1.time, p2.time)
    );
    pq.offer(new Node(0,0,0));
    boolean vis[][] = new boolean[m][n];
    vis[0][0] = true;
```

```java
while(!pq.isEmpty()){
    Node curr = pq.poll();

    if(curr.r == m-1 && curr.c == n-1){
        return curr.time;
    }

    for(int i = 0;i <4;i++){
        int nr = curr.r + dir[i];
        int nc = curr.c + dir[i+1];

        if(isValid(nr, nc, m, n) && !vis[nr][nc]){
            Node newNode = new Node( nr, nc, 1 + Math.max(curr.time, moveTime[nr][nc]));
            pq.offer(newNode);
            vis[nr][nc] = true;
        }
    }
}
return -1;
```

3342. Find Minimum Time to Reach Last Room II -Medium

There is a dungeon with n x m rooms arranged as a grid.

You are given a 2D array moveTime of size n x m, where moveTime[i][j] represents the minimum time in seconds when you can start moving to that room. You start from the room (0, 0) at time t = 0 and can move to an adjacent room. Moving between adjacent rooms takes one second for one move and two seconds for the next, alternating between the two.

Return the minimum time to reach the room (n - 1, m - 1).

Two rooms are adjacent if they share a common wall, either horizontally or vertically.

Input: moveTime = [[0,4],[4,4]]

Output: 7

Explanation

The minimum time required is 7 seconds.

At time t == 4, move from room (0, 0) to room (1, 0) in one second.

At time t == 5, move from room (1, 0) to room (1, 1) in two seconds.

Either cost can be +1 or +2 (alternates)

```java
private static final int[] dir = { -1, 0, 1, 0 ,-1};

    static class Node{
        int r;
        int c;
        int time;
        int cost;

        Node(int r, int c, int time, int cost){
            this.r = r;
            this.c = c;
            this.time = time;
            this.cost = cost;
        }
    }

    public static boolean isValid(int r, int c, int m, int n){
        return (r >= 0 && r< m && c >= 0 && c< n);
    }


public int minTimeToReach(int[][] moveTime) {
    int m = moveTime.length;
    int n = moveTime[0].length;

    PriorityQueue<Node> pq = new PriorityQueue<>( (p1,p2) -> Integer.compare(p1.time, p2.time));

    pq.offer(new Node(0,0,0,0));

    boolean vis[][] = new boolean[m][n];
    vis[0][0] = true;

    while(!pq.isEmpty()){
     Node curr = pq.poll();
```

```java
    if(curr.r == m-1 && curr.c == n-1) return curr.time;

    for(int i = 0; i< 4; i++){
        int nr = curr.r + dir[i];
        int nc = curr.c + dir[i+1];
        int cost = curr.cost;
        cost = (cost == 1) ? 2 : 1;

        if(isValid(nr, nc, m, n) && !vis[nr][nc]){
            Node newNode = new Node( nr, nc, cost + Math.max(curr.time, moveTime[nr][nc]), cost);
            pq.offer(newNode);
            vis[nr][nc] = true;
        }
    }
   }
   return -1;
```

2918. Minimum Equal Sum of Two Arrays After Replacing Zeros - medium

You are given two arrays nums1 and nums2 consisting of positive integers.

You have to replace all the 0's in both arrays with strictly positive integers such that the sum of elements of both arrays becomes equal.

Return the minimum equal sum you can obtain, or -1 if it is impossible.

Example 1:

Input: nums1 = [3,2,0,1,0], nums2 = [6,5,0]

Output: 12

Explanation: We can replace 0's in the following way:

- Replace the two 0's in nums1 with the values 2 and 4. The resulting array is nums1 = [3,2,2,1,4].

- Replace the 0 in nums2 with the value 1. The resulting array is nums2 = [6,5,1].

Both arrays have an equal sum of 12. It can be shown that it is the minimum sum we can obtain.

```
long num1Z = 0;
long num2Z = 0;
long minsum1 = 0;
long minsum2 = 0;

for(int i : nums1){
    if(i == 0) num1Z++;
    minsum1 += i;
}
for(int i : nums2){
    if(i == 0) num2Z++;
    minsum2 += i;
}

long min1 = num1Z + minsum1;
long min2 = num2Z + minsum2;

if(num1Z == 0 && num2Z == 0) return minsum1 == minsum2 ? minsum2 : -1;

else if ( num2Z == 0) return min1 <= minsum2 ? minsum2 : -1;

else if( num1Z == 0) return min2 <= minsum1 ? minsum1 : -1;

return Math.max(min1, min2);
```

2094. Finding 3-Digit Even Numbers

You are given an integer array digits, where each element is a digit. The array may contain duplicates.

You need to find all the unique integers that follow the given requirements:

The integer consists of the concatenation of three elements from digits in any arbitrary order.

The integer does not have leading zeros.

The integer is even.

For example, if the given digits were [1, 2, 3], integers 132 and 312 follow the requirements

Return a sorted array of the unique integers.

Input: digits = [2,1,3,0]

Output: [102,120,130,132,210,230,302,310,312,320]

Explanation: All the possible integers that follow the requirements are in the output array.

Notice that there are no odd integers or integers with leading zeros.

```
int freq[] = new int[10];
for(int i : digits) freq[i]++;

ArrayList<Integer> res = new ArrayList<>();

for(int i = 1; i <= 9; i++){
    if(freq[i] == 0 ) continue;

    freq[i]--;
    for(int j = 0; j<= 9; j++ ){
        if(freq[j] == 0) continue;

        freq[j]--;

        for(int k = 0; k<=8; k+=2){
            if(freq[k] == 0) continue;

            freq[k]--;
            res.add(i*100 + j*10 + k );
            freq[k]++;
        }
        freq[j]++;
    }
    freq[i]++;
}

return res.stream().mapToInt(Integer::intValue).toArray();
```

3335. Total Characters in String After Transformations I - Medium

You are given a string s and an integer t, representing the number of transformations to perform. In one transformation, every character in s is replaced according to the following rules:

If the character is 'z', replace it with the string "ab".

Otherwise, replace it with the next character in the alphabet. For example, 'a' is replaced with 'b', 'b' is replaced with 'c', and so on.

Return the length of the resulting string after exactly t transformations.

Since the answer may be very large, return it modulo 109 + 7.

Example 1:

Input: s = "abcyy", t = 2

Output: 7

Explanation:

First Transformation (t = 1):

'a' becomes 'b'

'b' becomes 'c'

'c' becomes 'd'

'y' becomes 'z'

'y' becomes 'z'

String after the first transformation: "bcdzz"

Second Transformation (t = 2):

'b' becomes 'c'

'c' becomes 'd'

'd' becomes 'e'

'z' becomes "ab"

'z' becomes "ab"

String after the second transformation: "cdeabab"

Final Length of the string: The string is "cdeabab", which has 7 characters.

Freq loads the counts of characters, for each character count in freq is assigned to its next characters count in freqn (i+1) (if a - 1, then b will also be 1). For count of z, freqn[0] and freq[1] is added with stored count of z in freq[25].

```
private int mod = 1_000_000_007;
public int lengthAfterTransformations(String s, int t) {
   int freq[] = new int[26];
   for(char i : s.toCharArray()){
    freq[i - 'a']++;
   }
   while( t-- > 0){
    int freqn[] = new int[26];
    for(int i = 0; i<25;i++){

        freqn[i+1] = (freq[i]);
    }

        freqn[0] = (freqn[0] + freq[25])%mod;
        freqn[1] = (freqn[1] + freq[25])%mod;

    freq = freqn;
   }
   long count = 0;
   for(int i : freq){
    count = (i + count) % mod;
   }
   return (int) count;
```

3337. Total Characters in String After Transformations II

You are given a string s consisting of lowercase English letters, an integer t representing the number of transformations to perform, and an array nums of size 26. In one transformation, every character in s is replaced according to the following rules:

Replace s[i] with the next nums[s[i] - 'a'] consecutive characters in the alphabet. For example, if s[i] = 'a' and nums[0] = 3, the character 'a' transforms into the next 3 consecutive characters ahead of it, which results in "bcd".

The transformation wraps around the alphabet if it exceeds 'z'. For example, if s[i] = 'y' and nums[24] = 3, the character 'y' transforms into the next 3 consecutive characters ahead of it, which results in "zab".

Return the length of the resulting string after exactly t transformations.

Since the answer may be very large, return it modulo 109 + 7.

Exmple 1:

Input: s = "abcyy", t = 2, nums = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2]

Output: 7

Explanation:

First Transformation (t = 1):

'a' becomes 'b' as nums[0] == 1

'b' becomes 'c' as nums[1] == 1

'c' becomes 'd' as nums[2] == 1

'y' becomes 'z' as nums[24] == 1

'y' becomes 'z' as nums[24] == 1

String after the first transformation: "bcdzz"

Second Transformation (t = 2):

'b' becomes 'c' as nums[1] == 1

'c' becomes 'd' as nums[2] == 1

'd' becomes 'e' as nums[3] == 1

'z' becomes 'ab' as nums[25] == 2

'z' becomes 'ab' as nums[25] == 2

String after the second transformation: "cdeabab"

Final Length of the string: The string is "cdeabab", which has 7 characters.

Stores the frequencies of the string s first, then calculates tm matrix where the frequency is applied using nums. Then takes that matrix and calculates its exponent matrix value. Multiplies intial_freq matrix and tm matrix.

```
private int mod = 1_000_000_007;

private int[][] matrixmul(int [][] A, int [][] B){
    int res [][] = new int[26][26];

    for(int i = 0; i<26;i++){
        for(int j = 0; j<26; j++){
            for(int k = 0; k<26;k++){
                res[i][j] = (int)((res[i][j] + (1L * A[i][k] * B[k][j]) % mod) % mod);
            }
        }
    }
    return res;
}
// matrix exponentiation
private int[][] matrixExp(int [][] tm , int t){
    int res [][] = new int[26][26];
    for(int i = 0; i<26;i++){
        res[i][i] = 1;
    }

    while( t > 0){
        if( (t & 1) == 1){
            res = matrixmul(tm, res);
        }

            tm = matrixmul(tm, tm);
            t >>=1;
    }
    return res;
}
```

```
public int lengthAfterTransformations(String s, int t, List<Integer> nums) {
    int initial_freq[] = new int[26];
    for(char c : s.toCharArray()) initial_freq[c - 'a']++;

    int tm[][] = new int[26][26];
    for(int i = 0; i< 26; i++){
        for(int j = i+1; j<= i + nums.get(i) ; j++){
            tm[j%26][i]++;
        }
    }

    int tex [][] = matrixExp(tm, t);
    int final1[] = new int [26];

    for(int i = 0; i< 26;i++){
        for(int j = 0; j<26; j++ ){
            final1[i] = (int) ((final1[i] + ( 1L * tex[i][j] * initial_freq[j])% mod)% mod);
        }
    }

    int count = 0;
    for(int i : final1){
        count = ( count + i)% mod;
    }

   return count;
}
```

2901. Longest Unequal Adjacent Groups Subsequence II - Medium

You are given a string array words, and an array groups, both arrays having length n.

The hamming distance between two strings of equal length is the number of positions at which the corresponding characters are different.

You need to select the longest subsequence from an array of indices [0, 1, ..., n - 1], such that for the subsequence denoted as [i0, i1, ..., ik-1] having length k, the following holds:

For adjacent indices in the subsequence, their corresponding groups are unequal, i.e., groups[ij] != groups[ij+1], for each j where 0 < j + 1 < k.

words[ij] and words[ij+1] are equal in length, and the hamming distance between them is 1, where 0 < j + 1 < k, for all indices in the subsequence.

Return a string array containing the words corresponding to the indices (in order) in the selected subsequence. If there are multiple answers, return any of them.

Note: strings in words may be unequal in length.

Input: words = ["bab","dab","cab"], groups = [1,2,2]

Output: ["bab","cab"]

Explanation: A subsequence that can be selected is [0,2].

groups[0] != groups[2]

words[0].length == words[2].length, and the hamming distance between them is 1.

So, a valid answer is [words[0],words[2]] = ["bab","cab"].

Another subsequence that can be selected is [0,1].

groups[0] != groups[1]

words[0].length == words[1].length, and the hamming distance between them is 1.

So, another valid answer is [words[0],words[1]] = ["bab","dab"].

It can be shown that the length of the longest subsequence of indices that satisfies the conditions is 2.

Taking dp and parent as arrays, processing through the words list and checking the 3 conditions: differ by 1 Hamming Distance, having different group numbers and dp[i] < dp[j] + 1. If satisfied, we updated the dp[i] with dp[j]+1 and parent[i] = j. We also track the max dp[i].

While making the ArrayList, we first go to the max dp word, then using parent, we track its parents and add in the ArrayList.

```
private boolean differ(String w1, String w2){
    if(w1.length() != w2.length()) return false;

    int count = 0;
    for(int i = 0; i< w1.length(); i++){
        if(w1.charAt(i) != w2.charAt(i)) count++;
    }

    return count == 1;
}
public List<String> getWordsInLongestSubsequence(String[] words, int[] groups) {
    int [] dp = new int[words.length];
    int [] p = new int[groups.length];

    Arrays.fill(dp, 1);
    Arrays.fill(p, -1);
```

```
    int max = 0;
    for(int i = 0; i < words.length; i++){
        for(int j = 0; j< i; j++){
            if(groups[i] != groups[j] &&
                differ(words[i], words[j]) &&
                dp[i] < dp[j] + 1){
                    dp[i] = dp[j]  + 1;
                    p[i] = j;
                }
        }
        if (dp[i] > max ) max = dp[i];
    }

    List<String> arr = new ArrayList<>();
    for(int i = 0; i< words.length; i++){
        if(dp[i] == max){
            while( i != -1){
                arr.add(words[i]);
                i = p[i];
            }
            break;
        }
    }

    arr = arr.reversed();
    return arr;
}
```

3362. Zero Array Transformation III - Medium

You are given an integer array nums of length n and a 2D array queries where queries[i] = [li, ri].

Each queries[i] represents the following action on nums:

Decrement the value at each index in the range [li, ri] in nums by at most 1.

The amount by which the value is decremented can be chosen independently for each index. A Zero Array is an array with all its elements equal to 0.

Return the maximum number of elements that can be removed from queries, such that nums can still be converted to a zero array using the remaining queries. If it is not possible to convert nums to a zero array, return -1.

Input: nums = [2,0,2], queries = [[0,2],[0,2],[1,1]] Output: 1

After removing queries[2], nums can still be converted to a zero array. Using queries[0], decrement nums[0] and nums[2] by 1 and nums[1] by 0. Using queries[1], decrement nums[0] and nums[2] by 1 and nums[1] by 0.

```java
public int maxRemoval(int[] nums, int[][] queries) {
    PriorityQueue<Integer> usedQ = new PriorityQueue<>();
    PriorityQueue<Integer> availQ = new PriorityQueue<>(Collections.reverseOrder());
    int qp = 0;
    int avail_count = 0;
    Arrays.sort(queries, (a,b) -> Integer.compare(a[0], b[0]));

    for(int i = 0; i< nums.length; i++){
        // Add the queries starting at i in availQ
        while(qp < queries.length && queries[qp][0] == i){
            availQ.offer(queries[qp][1]);
            qp++;
        }
        // Subtract the available number of queries present in usedQ
        nums[i] -= usedQ.size();

        // add the queries into usedQ until nums[i] == 0
        while(nums[i] > 0 && !availQ.isEmpty() && availQ.peek() >= i){
            int end = availQ.poll();
            usedQ.offer(end);
            nums[i]--;
            avail_count++;
        }
        //  If nums[i] after subtracting > 0 then
        if(nums[i] >0) return -1;
        // Remove queries that are already used, 'r' is == i query is ended
        while(!usedQ.isEmpty() && usedQ.peek() == i){
            usedQ.poll();
        }
    }

    return queries.length - avail_count;
}
```

Using a difference array to store and simulate workloads through nums.

```
 PriorityQueue<Integer> availQ = new PriorityQueue<>(Collections.reverseOrder());
 int qp = 0;
int work[] = new int[nums.length + 1];
 Arrays.sort(queries, (a,b) -> Integer.compare(a[0], b[0]));

 // prefix difference array to simulate workloads over time.
 for(int i = 0; i< nums.length; i++){
    if( i > 0){
     work[i] += work[i-1];
    }

    while( qp < queries.length && queries[qp][0] == i){
     availQ.offer(queries[qp][1]);
     qp++;
    }

    while( work[i] < nums[i]){
     if(availQ.isEmpty() || availQ.peek() < i){
         return -1;
     }

     work[i]++;
     work[availQ.poll() + 1]--;
    }
 }
 return availQ.size();
```

118. Pascal's Triangle - Easy

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

```
    List<List<Integer>> res = new ArrayList<>();

    if( numRows == 0){
        return res;
    }

    if(numRows == 1){
        List<Integer> ar = new ArrayList<>();
        ar.add(1);
        res.add(ar);
        return res;
    }

    res = generate(numRows - 1);
    List<Integer> pr = res.get(numRows - 2);
    List<Integer> cr = new ArrayList<>();

    cr.add(1);
    for(int i = 1; i < numRows - 1; i++){
        cr.add(pr.get(i-1) + pr.get(i));
    }
    cr.add(1);
    res.add(cr);
    return res;
}
```

135. Candy - HARD

There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy.

Children with a higher rating get more candies than their neighbors.

Return the minimum number of candies you need to have to distribute the candies to the children.

Input: ratings = [1,0,2]

Output: 5

Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Left to right pass to track if i-1 < i. Right to left pass to check if i-1 > i

```
    int [] arr = new int[ratings.length];
    Arrays.fill(arr, 1);
    int count = 0;

    for(int i = 1; i< ratings.length; i++){
        if(ratings[i-1] < ratings[i]){
            arr[i] = arr[i-1] + 1;
        }
    }

    for(int i = ratings.length - 1; i> 0; i--){
        if( ratings[i - 1] > ratings[i]){
            arr[i-1] = Math.max(arr[i-1], arr[i] + 1);
        }

        count += arr[i-1];
    }

    return count + arr[ratings.length - 1];
```

3403. Find the Lexicographically Largest String From the Box I - Medium

You are given a string word, and an integer numFriends.

Alice is organizing a game for her numFriends friends. There are multiple rounds in the game, where in each round:

word is split into numFriends non-empty strings, such that no previous round has had the exact same split.

All the split words are put into a box.

Find the lexicographically largest string from the box after all the rounds are finished.

Input: word = "dbca", numFriends = 2

Output: "dbc"

Explanation:

All possible splits are:

"d" and "bca".

"db" and "ca".

"dbc" and "a".

To find the lexicographically largest substring with numFriends splits, we have a range from i to i + word.length() - numFriends + 1. First we find the marCharacter in the string, then for each position of that maxChar we find the substring and compute the maximum largest lexicographic substring.

```
if(numFriends == 1) return word;
int len = word.length() - numFriends + 1;
char max = 'a';
String result = "";

for(char i : word.toCharArray()){
    if(i > max) max = i;
}

for(int i = 0; i<word.length(); i++){
    if(word.charAt(i) == max){
        if( i + len <= word.length() ){
            String sub = word.substring(i, i + len);
            if(sub.compareTo(result) > 0) result = sub;
        }
        else {
            String sub = word.substring(i);
            if(sub.compareTo(result) > 0) result = sub;
        }
    }
}
return result;
```

1695. Maximum Erasure Value

You are given an array of positive integers nums and want to erase a subarray containing unique elements. The score you get by erasing the subarray is equal to the sum of its elements.

Return the maximum score you can get by erasing exactly one subarray.

An array b is called to be a subarray of a if it forms a contiguous subsequence of a, that is, if it is equal to a[l],a[l+1],...,a[r] for some (l,r).

Example 1:

Input: nums = [4,2,4,5,6]

Output: 17

Explanation: The optimal subarray here is [2,4,5,6].

Example 2:

Input: nums = [5,2,1,2,5,2,1,2,5]

Output: 8

Explanation: The optimal subarray here is [5,2,1] or [1,2,5].

```java
public int maximumUniqueSubarray(int[] nums) {
    HashSet<Integer> set = new HashSet<>();

    int sum = 0;
    int max = Integer.MIN_VALUE;
    int k = 0;
    for(int i = 0; i< nums.length; i++){
        while(set.contains(nums[i])){
            sum -= nums[k];
            set.remove(nums[k++]);

        }
        set.add(nums[i]);
        sum += nums[i];
        max = Math.max(max, sum);

    }
    return max;
}
```

```java
int[] lastIndex = new int[10001];
for (int i = 0; i < lastIndex.length; i++) {
    lastIndex[i] = -1;
}
int l = -1, sum = 0;
int[] prefixSum = new int[nums.length+1];

for (int r = 0; r < nums.length; r++) {
    prefixSum[r+1] = nums[r] + prefixSum[r];
    if (lastIndex[nums[r]] >= 0) {
        l = Math.max(l, lastIndex[nums[r]]);
    }
    sum = Math.max(sum, prefixSum[r+1] - prefixSum[l+1]);
    lastIndex[nums[r]] = r;
}

return sum;
```

2348. Number of Zero-Filled Subarrays

Given an integer array nums, return the number of subarrays filled with 0.

A subarray is a contiguous non-empty sequence of elements within an array

Example 1:

Input: nums = [1,3,0,0,2,0,0,4]

Output: 6

Explanation:

There are 4 occurrences of [0] as a subarray.

There are 2 occurrences of [0,0] as a subarray.

There is no occurrence of a subarray with a size more than 2 filled with 0. Therefore, we return 6.

```java
public long zeroFilledSubarray(int[] nums) {
    long count = 0;
    long subsum = 0;

    for(int i = 0; i<nums.length; i++){
        if(nums[i] == 0){
            count++;
        }
        else{
            subsum += (count * (count + 1)/2);
            count = 0;
        }
    }
    if(count > 0){
     subsum += (count * (count + 1)/2);
    }
    return subsum;

}
```

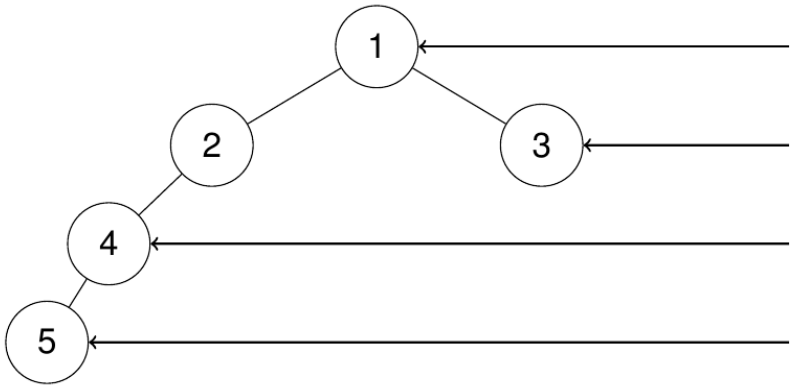
199. Binary Tree Right Side View - Medium

Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example 1:

Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]



O(n) approach - recursive DFS, with a stop condition of length of level and length of arr. If the length is equal, then we append the right val as we are calling the root.right first.

```java
public void rec(TreeNode root, List<Integer> arr, int level){
    if( root == null)
        return;
    if(level == arr.size()){
        arr.add(root.val);
    }
    rec(root.right, arr, level + 1);
    rec(root.left, arr, level + 1);
}

public List<Integer> rightSideView(TreeNode root) {
    if(root == null){
        return new ArrayList<Integer>();
    }
    List<Integer> arr = new ArrayList<Integer>();
    int level = 0;
    rec(root, arr, 0);
    return arr;
```