Q1. Morris Inorder Traversal

```
public class Solution {
    public ArrayList<Integer> solve(TreeNode A) {
        ArrayList<Integer> ans = new ArrayList<Integer>();
        TreeNode curr = A;
        while(curr != null){
            if(curr.left == null){     //check left side null or not
                ans.add(curr.val);
                curr= curr.right;
            }
            else{
                TreeNode temp = curr.left;
                while(temp.right != null && temp.right != curr){  //for finding inorder
predecesor
                    temp = temp.right;
                }
                if(temp.right == null){      //link created between inorder predecesor and
curr node
                    temp.right = curr;
                    curr= curr.left;
                }
                else{                       //link deleted after got curr again and print curr
                    temp.right = null;
                    ans.add(curr.val);
                    curr= curr.right;
                }
            }
        }
        return ans;
    }
}
```

Q3. Recover Binary Search Tree

```
public class Solution {
    public ArrayList<Integer> recoverTree(TreeNode A) {
        ArrayList<Integer> ans = new ArrayList<>();
        int first = Integer.MIN_VALUE, second = 0, logic = 1;
        TreeNode curr = A;
        while(curr != null) {
            if(curr.left == null) {
                if(logic == 1) {
                    if(curr.val > first) first = curr.val;
                    else{
                        logic = 0;
                        second = curr.val;
                    }
                }
                else if(curr.val < second) {
                    ans.add(curr.val);
                    ans.add(first);
                    return ans;
                }
                curr = curr.right;
            }
            else {
                TreeNode temp = curr.left;
                while(temp.right != null && temp.right != curr) {
                    temp = temp.right;
                }
                if(temp.right == null) {
                    temp.right = curr;
                    curr = curr.left;
                }
                else{
                    temp.right = null;
                    if(logic == 1) {
                        if(curr.val > first) first = curr.val;
                        else{
                            logic = 0;
                            second = curr.val;
                        }
                    }
                    else if(curr.val < second) {
                        ans.add(curr.val);
                        ans.add(first);
                        return ans;
                    }
                    curr = curr.right;
                }
            }
        }
        ans.add(second);
        ans.add(first);
        return ans;
    }
}
```

Q1. Path Sum

```
public class Solution {
    public int hasPathSum(TreeNode A, int B) {
        if(A == null){
            return 0;
        }
        if(A.left == null && A.right == null){
            if(A.val == B){
                return 1;
            }
        }
        int left= hasPathSum(A.left, B - A.val);
        int right= hasPathSum(A.right, B - A.val);
        if(left == 1 || right == 1){
            return 1;
        }
        return 0;
        // return (hasPathSum(A.left, B - A.val) || hasPathSum(A.right, B - A.val)); this not
work because return type is int
    }
}
```

Q2. Next Pointer Binary Tree

```
//without using space
public class Solution {
    public void connect(TreeLinkNode root) {
        TreeLinkNode curr = root;
        while(curr.left != null){
            TreeLinkNode temp = curr;
            while(temp != null){
                temp.left.next = temp.right;
                if(temp.next != null){
                    temp.right.next = temp.next.left;
                }
                temp= temp.next;
            }
            curr= curr.left;
        }
    }
}

// TC -> O(N)
// Sc -O(1)

//  with queue

// public class Solution {
//     public void connect(TreeLinkNode root) {
//         Queue<TreeLinkNode> q = new LinkedList<>();
//         q.offer(root);
//         while(q.size() != 0){
//             int sz = q.size();
//             for(int i=0; i<sz; i++){
//                 TreeLinkNode temp = q.remove();
//                 if(temp.left != null){
//                     q.offer(temp.left);
//                 }
//                 if(temp.right != null){
//                     q.offer(temp.right);
//                 }
//                 if(i < sz-1){
//                     temp.next = q.peek();
//                 }
//             }
//         }
//     }
// }
```

Q3. Diameter of binary tree

```
public class Solution {
    public int solve(TreeNode A) {
        return getDiameter(A);
    }

    public int getDiameter(TreeNode currNode){
        if(currNode == null){
            return 0;
        }
        int heightLST = getHeight(currNode.left);
        int heightRST = getHeight(currNode.right);
        return Math.max(heightLST+heightRST, Math.max(getDiameter(currNode.left),
getDiameter(currNode.right)));
    }

    public int getHeight(TreeNode currNode){
        if(currNode == null){
            return 0;
        }
        return 1 + Math.max(getHeight(currNode.left), getHeight(currNode.right));
    }
}
```

Q4. Equal Tree Partition

```
public class Solution {
    long Total_count =0;
    int ans = 0;
    public int solve(TreeNode A) {
        //to calculate Total_count we can use any traversal
        Postorder(A);
        //To calculate tree with sum Total_count/2 with post traversal
        if(Total_count % 2 == 0){
            Equal_Tree_Partition(A);
            return ans;
        }
        else{
            return ans;
        }

    }

    public long Equal_Tree_Partition(TreeNode A){
        if(A == null){
            return 0;
        }
        long sum_left = Equal_Tree_Partition(A.left);
        long sum_right = Equal_Tree_Partition(A.right);
        if(sum_left == Total_count/2 || sum_right == Total_count/2){
            ans = 1;
        }
        return (sum_left + sum_right + A.val);
    }
    public void Postorder(TreeNode A){
        if(A == null){
            return;
        }
        Postorder(A.left);
        Postorder(A.right);
        Total_count = Total_count + A.val;
    }
}

// TC -> O(N)
// SC -> O(hight of tree) -> O(N)
```

Q2. Merge K Sorted Lists

```
public class Solution {
    public ListNode mergeKLists(ArrayList<ListNode> a) {
        int N= a.size();
        ListNode anslink= new ListNode(0);
        ListNode ptr= anslink;
        // PriorityQueue<ListNode> pq=  new PriorityQueue<ListNode>();
        // PriorityQueue<ListNode> pq = new PriorityQueue<>((c,b)-> c.val - b.val);
        PriorityQueue<ListNode> pq=new PriorityQueue(new ListNodeComparator());

        for(int i=0; i<N; i++){          //TC->N*logN
            pq.offer(a.get(i));
        }

        while(!pq.isEmpty()){           //TC -> M*logN
            ListNode temp = pq.poll();
            ptr.next = temp;
            if(temp.next != null){
                pq.offer(temp.next);
            }
            ptr= ptr.next;
        }
        return anslink.next;
    }
}

class ListNodeComparator implements Comparator<ListNode> {
    // Method
    // Sorting in ascending order of val
    public int compare(ListNode a, ListNode b)
    {
        return a.val - b.val;
    }
}

// TC -> O(M*logN)  M represents all elements
// SC -> O(M)
```

Q3. Build a Heap

```java
class Solution {
    public int[] buildHeap(int[] A) {

        int N = A.length;
        // int[] A = new int[N];
        for(int i=0; i<N; i++){
            // A[i] = A[i];
            UpHeaptify(A, i);
        }
        return A;

    }
    public void UpHeaptify(int[] A, int i){
        int parent_i = (i-1)/2;
        while(i != 0 && A[parent_i] > A[i]){
            swap(A, parent_i, i);
            i= parent_i;
            parent_i= (i-1)/2;
        }
    }

    public void swap(int[] A, int X, int Y){
        int temp = A[X];
        A[X] = A[Y];
        A[Y] = temp;
    }
}
```

Q4. Heap Queries

```java
public class Solution {
    public ArrayList<Integer> solve(ArrayList<ArrayList<Integer>> A) {
        int N= A.size();
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        for(int i=0; i<N; i++){
            int P = A.get(i).get(0);
            int Q = A.get(i).get(1);
            if(P == 1){
                if(pq.isEmpty()){
                    ans.add(-1);
                }
                else{
                    ans.add(pq.remove());
                }
            }
            else{
                pq.offer(Q);
            }
        }
        return ans;
    }
}

// Time Complexity : O(NlogN)
// Space Complexity : O(N)
```

Q1. K Places Apart

```
public class Solution {
    public int[] solve(int[] A, int B) {

        PriorityQueue<Integer> q = new PriorityQueue<Integer>();
        int N= A.length;
        // int[] ans= new int[N];

        for(int i=0; i<=B; i++){      // TC -> B+1 * log(B+1)
            q.offer(A[i]);
        }

        int index= 0;
        for(int i= B+1; i<N; i++){      // TC -> (N-B)* logB
            A[index]= q.poll();
            index++;
            q.offer(A[i]);
        }

        while(!q.isEmpty()){      // TC -> B*logB
            A[index]= q.poll();
            index++;
        }
        return A;
    }
}

// TC -> O(B + Nlog(B))
// SC -> O(B)
```

Q2. Ath largest element

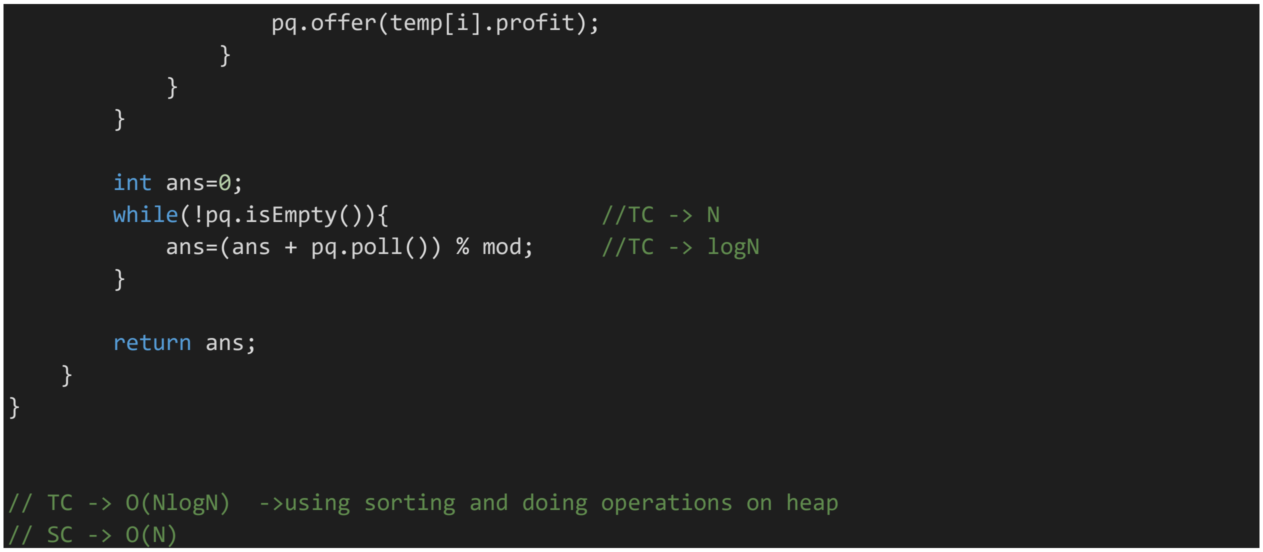
```
public class Solution {
    public ArrayList<Integer> solve(int A, ArrayList<Integer> B) {
        int N= B.size();
        PriorityQueue<Integer> pq= new PriorityQueue<>();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        for(int i=0; i<A; i++){
            pq.offer(B.get(i));
        }

        for(int i=0; i<A-1; i++){
            ans.add(-1);
        }
        ans.add(pq.peek());

        for(int i=A; i<N; i++){
            if(pq.peek() < B.get(i)){
                pq.poll();
                pq.offer(B.get(i));
            }
            ans.add(pq.peek());
        }
        return ans;
    }
}

// TC -> O(N* logA)
// SC -> O(A)
```

Q3. Running Median

```
public class Solution {
    public int[] solve(int[] A) {
        int N= A.length;
        PriorityQueue<Integer> first_max= new
PriorityQueue<Integer>(Collections.reverseOrder());
        PriorityQueue<Integer> second_min= new PriorityQueue<Integer>();
        int[] ans = new int[N];
        first_max.offer(A[0]);
        ans[0]=A[0];
        // int index= 0;
        for(int i=1; i<N; i++){          //TC-> N
            if(A[i] < first_max.peek()){
                first_max.offer(A[i]);
            }
            else{
                second_min.offer(A[i]);
            }
            if((first_max.size() - second_min.size()) > 1){  //rebalance
                second_min.offer(first_max.poll());
            }

            if(second_min.size() > first_max.size()){        //rebalance
                first_max.offer(second_min.poll());
            }

            // if(first_max.size() == second_min.size()){
            //     ans[i]= (first_max.peek() + second_min.peek())/ 2;
            // }
            // else{
            //     ans[i]= first_max.peek();
            // }
            // System.out.println(first_max.size() + " " + second_min.size());
            ans[i]= first_max.peek();   //see note
        }
        return ans;
    }
}

// TC -> O(NlogN)
// SC -> O(N)
```

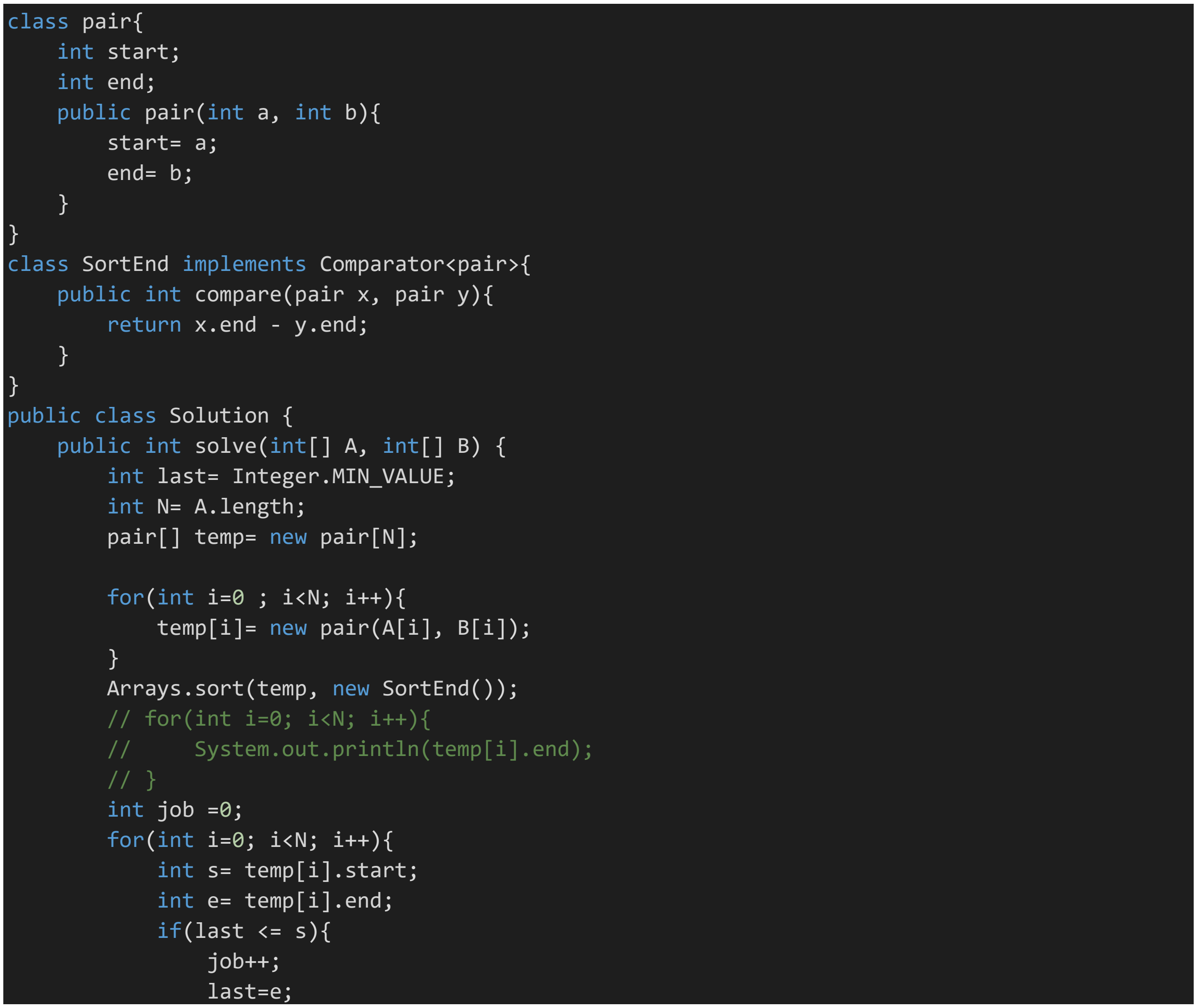
Q1. Flipkart's Challenge in Effective Inventory Management

```
class pair{
    int time;
    int profit;
    public pair(int a, int b){
        time= a;
        profit= b;
    }
}

class sorttime implements Comparator<pair>{
    public int compare(pair a, pair b){
        int f_time= a.time;
        int s_time= b.time;
        return f_time - s_time;
    }
}

public class Solution {
    int mod= 1000000007;
    public int solve(int[] A, int[] B) {
        int N= A.length;
        int M= B.length;
        pair[] temp = new pair[N];
        for(int i=0; i<N; i++){
            temp[i]= new pair(A[i], B[i]);
        }

        Arrays.sort(temp, new sorttime());          //TC -> NlogN
        PriorityQueue<Integer> pq= new PriorityQueue<Integer>();

        int T = 0;
        for(int i=0; i<N; i++){                    //TC -> N
        //by shubham soni approach
            // pq.offer(temp[i].profit);             //TC -> logN
            // if(pq.size() > temp[i].time){         //TC -> logN
            //     pq.poll();
            // }
            //by aayush sir approach
            if(T < temp[i].time){
                pq.offer(temp[i].profit);
                T++;
            }
            else{
                if(pq.peek() < temp[i].profit){
                    pq.poll();
```



```
                    pq.offer(temp[i].profit);
                }
            }
        }

        int ans=0;
        while(!pq.isEmpty()){                //TC -> N
            ans=(ans + pq.poll()) % mod;     //TC -> logN
        }

        return ans;
    }
}


// TC -> O(NlogN)  ->using sorting and doing operations on heap
// SC -> O(N)
```

Q2. Finish Maximum Jobs



```
class pair{
    int start;
    int end;
    public pair(int a, int b){
        start= a;
        end= b;
    }
}
class SortEnd implements Comparator<pair>{
    public int compare(pair x, pair y){
        return x.end - y.end;
    }
}
public class Solution {
    public int solve(int[] A, int[] B) {
        int last= Integer.MIN_VALUE;
        int N= A.length;
        pair[] temp= new pair[N];

        for(int i=0 ; i<N; i++){
            temp[i]= new pair(A[i], B[i]);
        }
        Arrays.sort(temp, new SortEnd());
        // for(int i=0; i<N; i++){
        //      System.out.println(temp[i].end);
        // }
        int job =0;
        for(int i=0; i<N; i++){
            int s= temp[i].start;
            int e= temp[i].end;
            if(last <= s){
                job++;
                last=e;
            }
        }
        return job;
    }
}

// TC - > O(NlogN)
// SC -> O(N)
```

Q3. Distribute Candy

```
// with left right giving same TC and SC
public class Solution {
    public int candy(int[] A) {
        int N= A.length;
        int[] left = new int[N];
        int[] right = new int[N];

        left[0]= 1;
        for(int i=1; i<N; i++){
            if(A[i-1] < A[i]){
                left[i]= left[i-1] + 1;
            }
            else{
                left[i]= 1;
            }
        }

        right[N-1]= 1;
        for(int i=N-2; i>=0; i--){
            if(A[i] > A[i+1]){
                right[i]= right[i+1] + 1;
            }
            else{
                right[i]= 1;
            }
        }

        int Candy=0;
        for(int i=0; i<N; i++){
            Candy = Candy + Math.max(left[i], right[i]);
        }

        return Candy;
    }
}

// TC -> O(N)
// SC -> O(N)
```

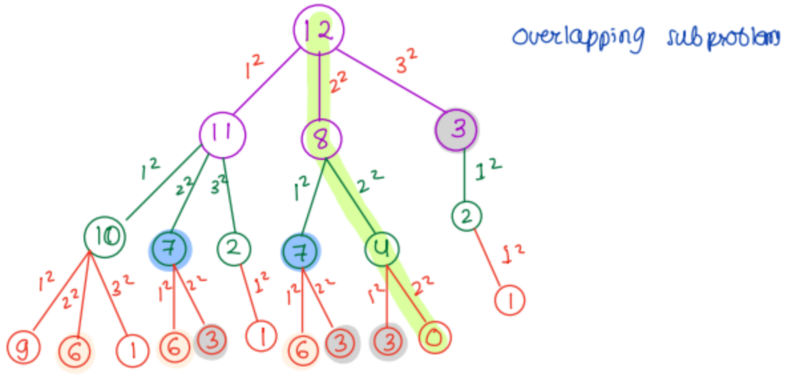
Q1. Stairs

```
//with iterative teqnic also known botton up approach with space utilization (tabulation)
public class Solution {
    public int climbStairs(int A) {
        int mod = 1000000007;
        if(A <= 2){
            return A;
        }
        int a=1;
        int b=2;
        int c= -1;
        for(int i=2; i<A; i++){
            c= (a + b) % mod;
            a=b;
            b=c;
        }
        return c;
    }
}

// TC -> O(N)
// SC -> O(1)


//With DP and Top-Down Approach (memoization)
// public class Solution {
//     int ans;
//     int mod = 1000000007;
//     public int climbStairs(int A) {
//         int[] DP = new int[A+1];
//         for(int i=0; i<A+1; i++){
//             DP[i] = -1;
//         }
//         int val = Fibo(DP,A);
//         return val;
//     }

//     public int Fibo(int[] DP,int A){
//         if(A <= 2){
//             return A;
//         }
//         if(DP[A] != -1){
//             return DP[A];
//         }
//         ans = (Fibo(DP,A-1) + Fibo(DP, A-2)) % mod;
//         DP[A] = ans;
//         return ans;
//     }
// }

// TC -> O(N)
// SC -> O(N)
```

Q2. Fibonacci Number

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

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int A = input.nextInt();
        if(A <= 1){
            System.out.print(A);
        }
        else{
            int a=0;
            int b=1;
            int c=-1;
            for(int i=2; i<=A; i++){
                c = a + b;
                a = b;
                b = c;
            }
            System.out.print(c);
        }
    }
}

// TC -> O(N)
// SC -> O(1)
```

Q3. Minimum Number of Squares



```
//Top-Down Approach (memoization)
public class Solution {
    public int countMinSquares(int A) {
        int[] DP = new int[A+1];
        for(int i=0; i<A+1; i++){
            DP[i] = -1;
        }
        int ans = MinCount(DP, A);
        return ans;
    }

    public int MinCount(int[] DP, int N){
        if(N == 0){
            return N;
        }
        if(DP[N] != -1){
            return DP[N];     //reuse
        }
        int count = Integer.MAX_VALUE;
        for(int i=1; i*i<=N; i++){
            count = Math.min(count , 1 + MinCount(DP, (N-i*i)));
        }
        DP[N] = count;    //store
        return count;
    }
}

// TC -> O(N * (N)^1/2)
// SC -> O(N)
```

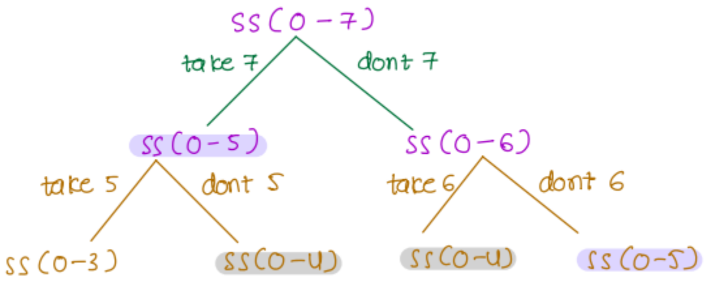
Q4. Max Sum Without Adjacent Elements

0 1 2 3 4 5 6 7

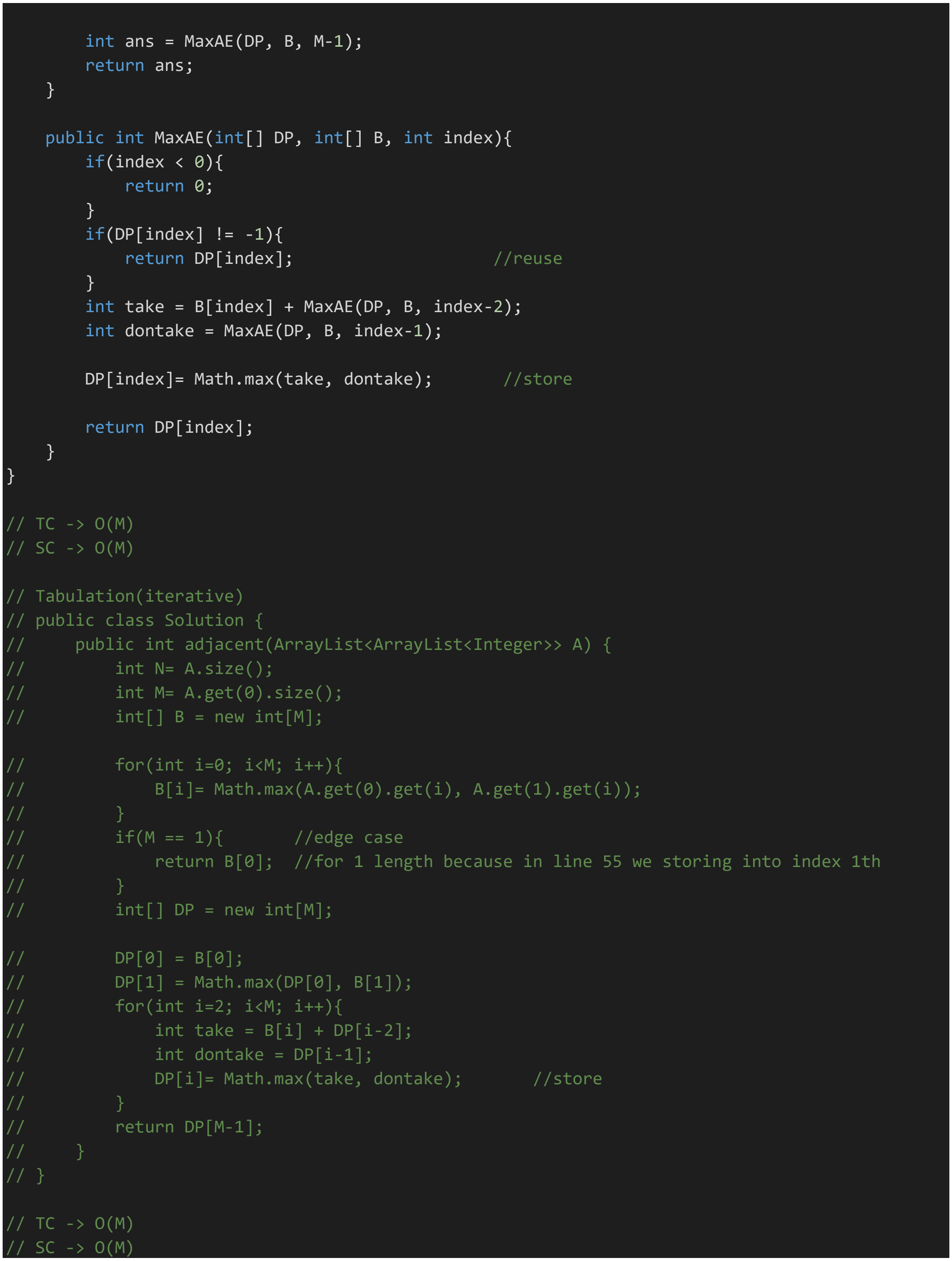
{2, -1, -4, 5, 3, -1, 4, 7}

ss(0-7) → max subsequence sum from 0 to 7

dp state

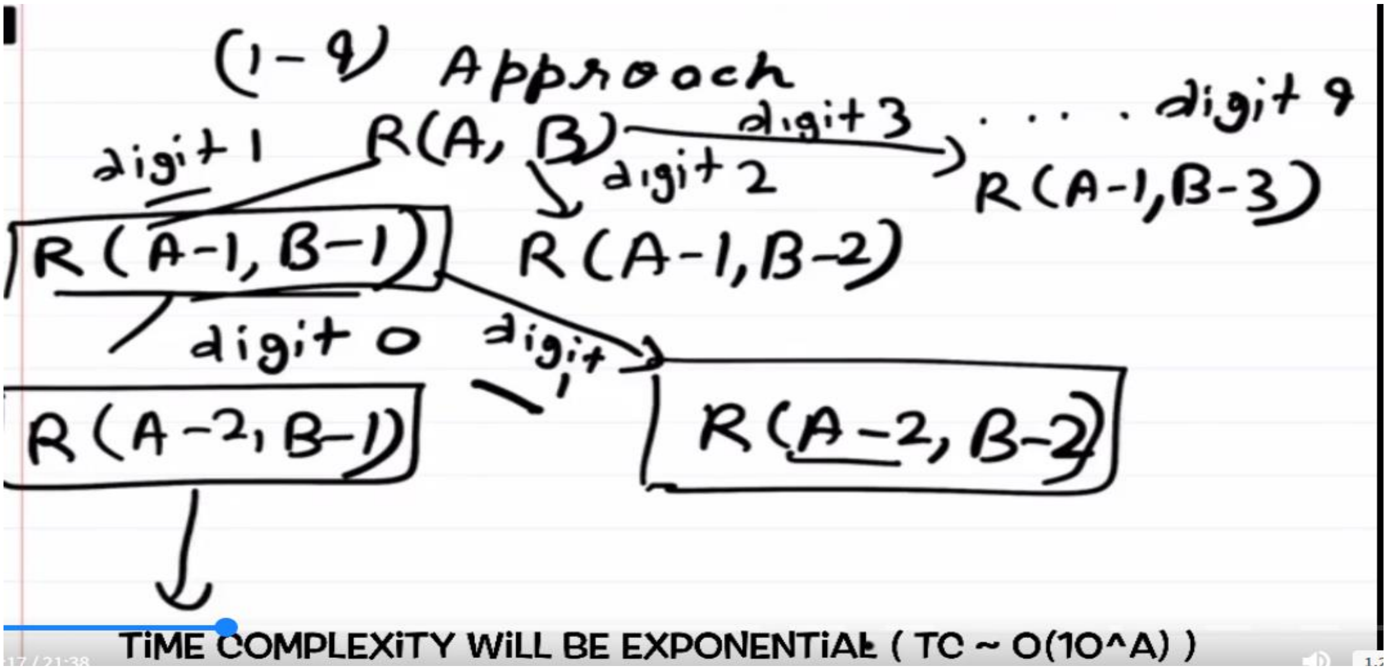


```
// Memoization(recurssive)
public class Solution {
    public int adjacent(ArrayList<ArrayList<Integer>> A) {
        int M=A.get(0).size();
        int[] DP= new int[M];

        for(int i=0; i<M; i++){
            DP[i]= -1;
        }

        int[] B= new int[M];
        for(int i=0; i<M; i++){
            B[i]= Math.max(A.get(0).get(i), A.get(1).get(i));
        }
```

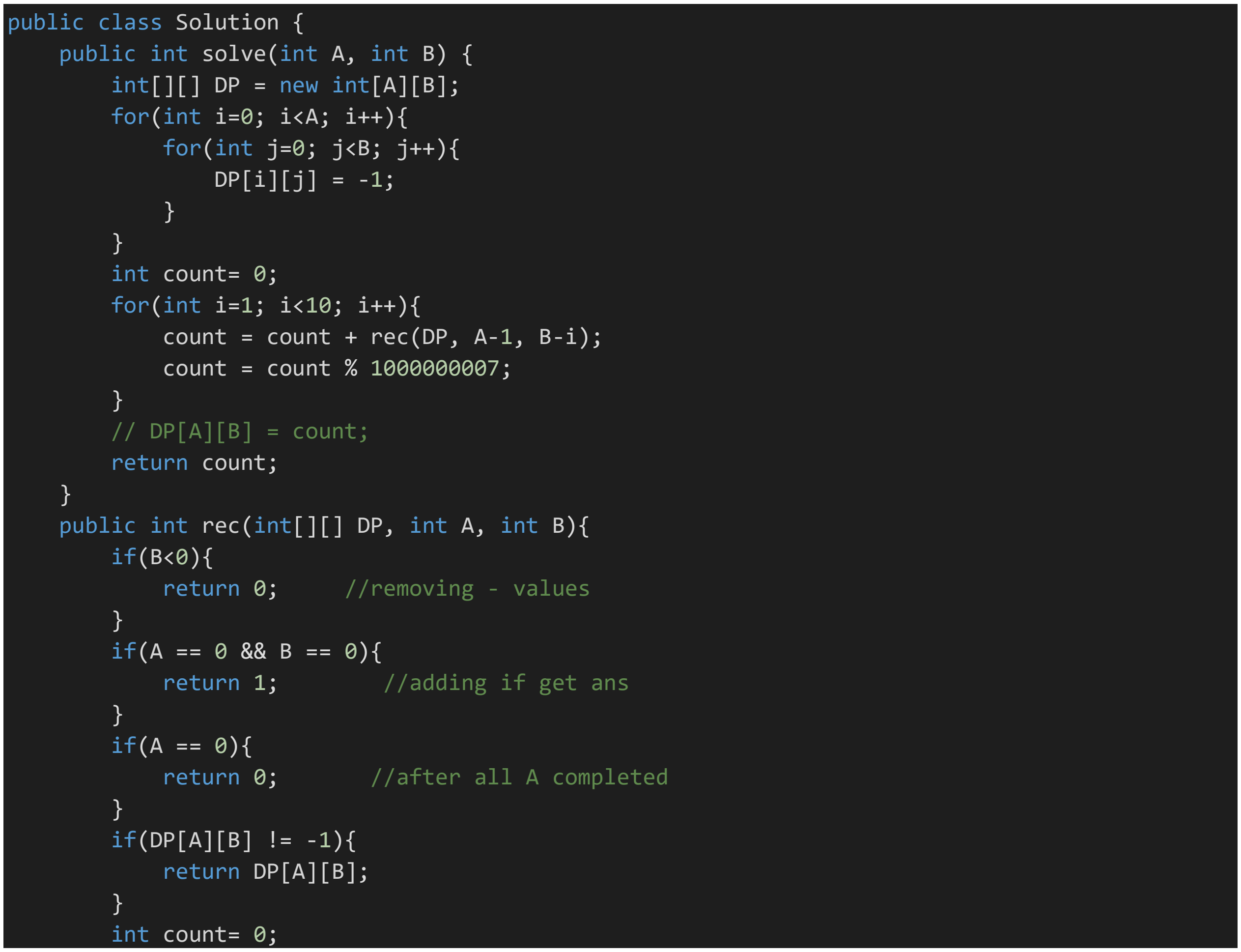


```
        int ans = MaxAE(DP, B, M-1);
        return ans;
    }

    public int MaxAE(int[] DP, int[] B, int index){
        if(index < 0){
            return 0;
        }
        if(DP[index] != -1){
            return DP[index];                    //reuse
        }
        int take = B[index] + MaxAE(DP, B, index-2);
        int dontake = MaxAE(DP, B, index-1);

        DP[index]= Math.max(take, dontake);       //store

        return DP[index];
    }
}

// TC -> O(M)
// SC -> O(M)

// Tabulation(iterative)
// public class Solution {
//     public int adjacent(ArrayList<ArrayList<Integer>> A) {
//         int N= A.size();
//         int M= A.get(0).size();
//         int[] B = new int[M];

//         for(int i=0; i<M; i++){
//             B[i]= Math.max(A.get(0).get(i), A.get(1).get(i));
//         }
//         if(M == 1){       //edge case
//             return B[0];  //for 1 length because in line 55 we storing into index 1th
//         }
//         int[] DP = new int[M];

//         DP[0] = B[0];
//         DP[1] = Math.max(DP[0], B[1]);
//         for(int i=2; i<M; i++){
//             int take = B[i] + DP[i-2];
//             int dontake = DP[i-1];
//             DP[i]= Math.max(take, dontake);       //store
//         }
//         return DP[M-1];
//     }
// }

// TC -> O(M)
// SC -> O(M)
```

Q1. N digit numbers



TIME COMPLEXITY WILL BE EXPONENTIAL (TC ~ O(10^A))



```java
public class Solution {
    public int solve(int A, int B) {
        int[][] DP = new int[A][B];
        for(int i=0; i<A; i++){
            for(int j=0; j<B; j++){
                DP[i][j] = -1;
            }
        }
        int count= 0;
        for(int i=1; i<10; i++){
            count = count + rec(DP, A-1, B-i);
            count = count % 1000000007;
        }
        // DP[A][B] = count;
        return count;
    }
    public int rec(int[][] DP, int A, int B){
        if(B<0){
            return 0;      //removing - values
        }
        if(A == 0 && B == 0){
            return 1;         //adding if get ans
        }
        if(A == 0){
            return 0;        //after all A completed
        }
        if(DP[A][B] != -1){
            return DP[A][B];
        }
        int count= 0;
        for(int i=0; i<10; i++){
            count = count + rec(DP, A-1, B-i);
            count = count % 1000000007;
        }
        DP[A][B] = count;
        return count;
    }
}

// TC -> O(A*B*10)
// SC -> O(A*B)
```

Q2. Unique Paths in a Grid

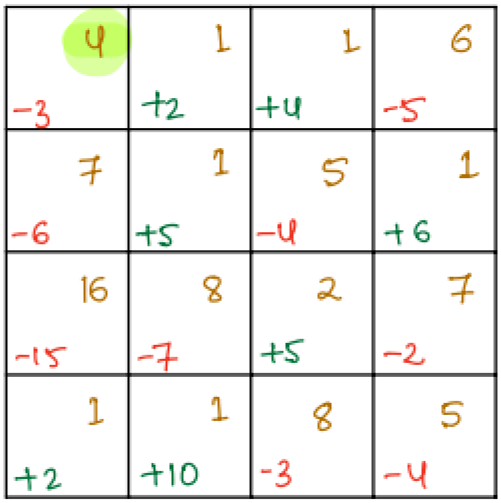
```
Obstacle
```

```
public class Solution {
    public int uniquePathsWithObstacles(int[][] A) {
        int N= A.length;
        int M= A[0].length;
        int[][] DP = new int[N][M];
        for(int i=0; i<N; i++){
            for(int j=0; j<M; j++){
                DP[i][j] = -1;
            }
        }
        int r= N-1;
        int c= M-1;
        int ways = rec(A, DP, r, c);
        return ways;
    }

    public int rec(int[][] A, int[][] DP, int r, int c){
        if(r < 0 || c < 0){
            return 0;
        }
        if(A[r][c] == 1){
            return 0;
        }
        if(r == 0 && c == 0){
            return 1;
        }
        if(DP[r][c] != -1){
            return DP[r][c];
        }
        int top = rec(A,DP,r-1,c);
        int left = rec(A,DP,r,c-1);

        DP[r][c] = top + left;

        return top+left;
    }
}
```

```
// TC -> O(M*N)
// Sc -> O(M*N)
```

Q3. Dungeon Princess

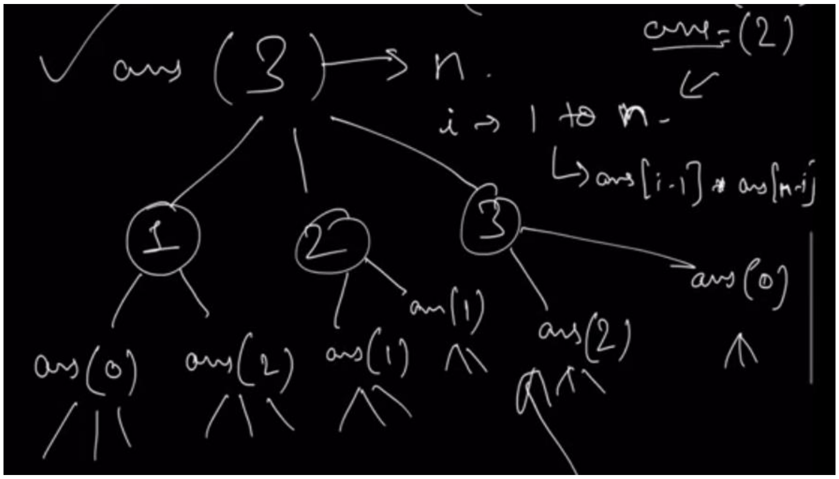
The demons had captured the **princess** and imprisoned her in the **bottom-right** corner of a dungeon. The dungeon consists of **M x N** rooms laid out in a 2D grid. Our valiant **knight** was initially positioned in the **top-left** room and must fight his way through the dungeon to rescue the princess.

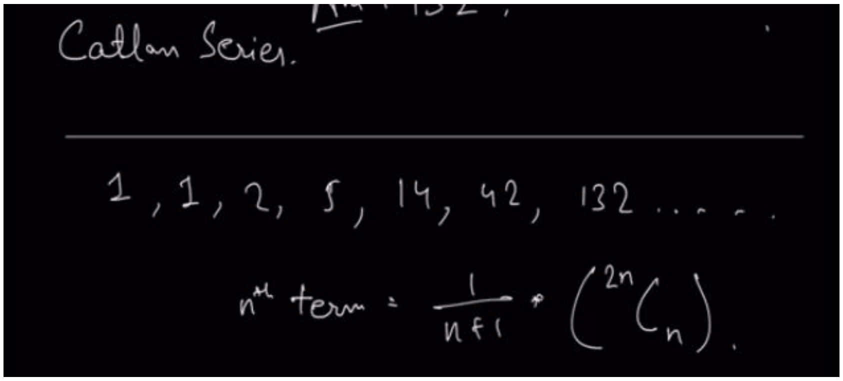


4 -3	1 +2	1 +4	6 -5
7 -6	1 +5	5 -4	1 +6
16 -15	8 -7	2 +5	7 -2
1 +2	1 +10	8 -3	5 -4

```
public class Solution {
    public int calculateMinimumHP(int[][] A) {
        int r= A.length;
        int c= A[0].length;
        int[][] DP = new int[r][c];
        for(int i=0; i<r; i++){
            for(int j=0; j<c; j++){
                DP[i][j] = -1;
            }
        }
        for(int i=r-1; i>=0; i--){
            for(int j=c-1; j>=0; j--){
                if(i == r-1 && j == c-1){
                    DP[i][j] = Math.max(1, 1-A[i][j]);
                }
                else if(i == r-1){
                    DP[i][j] = Math.max(1, DP[i][j+1] - A[i][j]);
                }
                else if(j == c-1){
                    DP[i][j] = Math.max(1, DP[i+1][j] - A[i][j]);
                }
                else{
                    int minHealth = Math.min(DP[i+1][j], DP[i][j+1]);
                    DP[i][j] = Math.max(1, minHealth - A[i][j]);
                }
            }
        }
        return DP[0][0];
    }
}

// TC -> O(r*c)
// SC -> O(r*c)
```

Q4. Unique Binary Search Trees II





Catlan Series.

1, 1, 2, 5, 14, 42, 132 .....

n^{th} term = $\frac{1}{n+1} * \left({}^{2n}C_n\right)$.

```
// tabulation Bottom up approach
// public class Solution {
//     public int numTrees(int A) {
//         int[] DP = new int[A+1];
//         DP[0]= 1;
//         DP[1]= 1;
//         if(A <=1){
//             return DP[A];
//         }
//         // DP[2]= 2;
//         for(int i=2; i<=A; i++){
//             // int total =0;
//             for(int j=1; j<=i; j++){
//                 DP[i] += DP[j-1] * DP[i-j];
//             }
//             // DP[i] =  total;
//         }
//         return DP[A];
//     }
// }

// TC -> O(A^2)
// SC -> O(A)

//formula approach
public class Solution {
    public int numTrees(int A) {
        long ans = 1;
        for(int i=0; i<A; i++){
            ans = ans * (2*A - i);
            ans = ans / (i+1);
        }
        ans = ans / (A+1);
        return (int)ans;
    }
}
// TC -> O(A)
// SC -> O(1)
```

Q1. Fractional Knapsack

```
//greedy approach
class pair{
    int value;
    int weight;
    double ratio;
    public pair(int a, int b, double c){
        value= a;
        weight= b;
        ratio= c;
    }
}
public class Solution {
    public int solve(int[] A, int[] B, int C) {
        int N= A.length;
        pair[] item= new pair[N];

        for (int i = 0; i < N; i++) {
            item[i] = new pair(A[i], B[i], (double) A[i] / B[i]);
        }

        Arrays.sort(item, new Comparator<pair>(){
            public int compare(pair a, pair b){
                if(a.ratio > b.ratio){
                    return -1;
                }
                else if(a.ratio < b.ratio){
                    return 1;
                }
                else{
                    return 0;
                }
            }
        } );

        double ans=0;
        for(int i=0; i<N; i++){
            if(C >= item[i].weight){
                C = C - item[i].weight;

                ans= ans + item[i].value;
            }
            else{
                ans = ans + C * item[i].ratio;
                break;
            }
        }
        ans = ans*1000;
        return (int)(ans/10);
    }
}
// TC -> O(N*logN)
// SC -> O(N)
```

Q2. 0-1 Knapsack

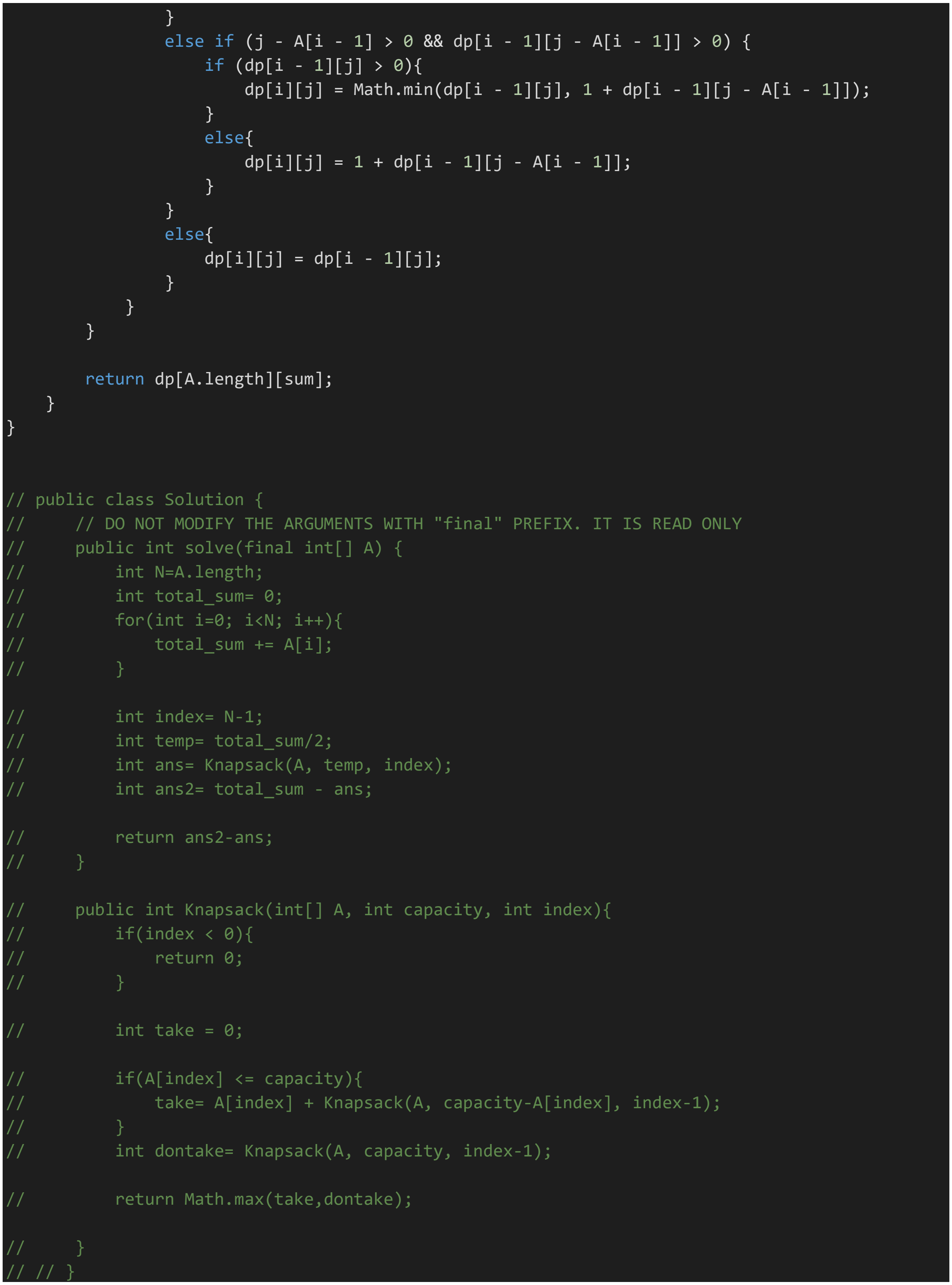
```
public class Solution {
    public int solve(int[] A, int[] B, int C) {
        int N= A.length;

        int[][] DP = new int[N][C+1];
        for(int i=0; i<N; i++){
            for(int j=0; j<C+1; j++){
                DP[i][j] = -1;
            }
        }
        return Knapsack(DP, A, B, C, N-1);

    }

    public int Knapsack(int[][] DP, int[] values, int[] weights, int C, int index){
        //base candition
        if(index < 0){
            return 0;
        }

        if(DP[index][C] != -1){
            return DP[index][C];    //reuse
        }
        int take=0;
        if(C >= weights[index]){
            take = values[index] + Knapsack(DP, values, weights, C-weights[index], index-1);
        }
        int dontake = Knapsack(DP, values, weights, C, index-1);

        DP[index][C] = Math.max(take, dontake);    //store
        return Math.max(take, dontake);
    }
}

// TC -> O(N*C+1) -> (NC)
// SC -> O(N*C+1) -> (NC)
```

Q3. Unbounded Knapsack

```
public class Solution {
    public int solve(int A, int[] B, int[] C) {
        int N= B.length;

        int[][] DP = new int[N][A+1];
        for(int i=0; i<N; i++){
            for(int j=0; j<A+1; j++){
                DP[i][j] = -1;
            }
        }
        return Knapsack(DP, B, C, A, N-1);

    }

    public int Knapsack(int[][] DP, int[] values, int[] weights, int Capacity, int index){
        //base candition
        if(index < 0){
            return 0;
        }

        if(DP[index][Capacity] != -1){
            return DP[index][Capacity];    //reuse
        }

        int take=0;
        if(Capacity >= weights[index]){
            take = values[index] + Knapsack(DP, values, weights, Capacity-weights[index],
index);  //just remove -1 from index
        }
        int dontake = Knapsack(DP, values, weights, Capacity, index-1);

        DP[index][Capacity] = Math.max(take, dontake);   //store
        return Math.max(take, dontake);
    }
}

// TC -> O(N*C+1) -> (NC)
// SC -> O(N*C+1) -> (NC)

// another approach
// public class Solution {
//     public int solve(int A, int[] B, int[] C) {
//         int[] DP = new int[A+1];

//         for(int i=0; i<A+1; i++){
//             DP[i]= -1;
//         }
//         return Unbounded_Knapsack(DP, A, B, C);
//     }

//     public int Unbounded_Knapsack(int[] DP, int capacity, int[] values, int[] weight){
//         if(DP[capacity] != -1){
//             return DP[capacity];
//         }

//         int mx=0;
//         for(int i=0 ; i<values.length; i++){
//             if(capacity >= weight[i]){
//                 mx= Math.max(mx, values[i] + Unbounded_Knapsack(DP, capacity-weight[i],
values, weight));
//             }
//         }

//         DP[capacity]= mx;
//         return mx;
//     }
// }


// TC -> number of unique DP state * TC per state
// TC -> O((C+1) * N -> O(NC)
// SC -> O(C+1) -> O(C)
```

Q4. Flip Array

```
import java.util.stream.*;
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int solve(final int[] A) {
        if (A.length == 1) return 0;
        if (A.length == 2) return 1;
        int sum = IntStream.range(0, A.length).map(i -> A[i]).sum();
        sum /= 2;

        int[][] dp = new int[A.length + 1][sum + 1];

        for (int i = 1; i <= A.length; i++) {
            for (int j = 1; j <= sum; j++) {
                if (j - A[i - 1] == 0){
                    dp[i][j] = 1;
```



```
                }
                else if (j - A[i - 1] > 0 && dp[i - 1][j - A[i - 1]] > 0) {
                    if (dp[i - 1][j] > 0){
                        dp[i][j] = Math.min(dp[i - 1][j], 1 + dp[i - 1][j - A[i - 1]]);
                    }
                    else{
                        dp[i][j] = 1 + dp[i - 1][j - A[i - 1]];
                    }
                }
                else{
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }

        return dp[A.length][sum];
    }
}


// public class Solution {
//     // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
//     public int solve(final int[] A) {
//         int N=A.length;
//         int total_sum= 0;
//         for(int i=0; i<N; i++){
//             total_sum += A[i];
//         }

//         int index= N-1;
//         int temp= total_sum/2;
//         int ans= Knapsack(A, temp, index);
//         int ans2= total_sum - ans;

//         return ans2-ans;
//     }

//     public int Knapsack(int[] A, int capacity, int index){
//         if(index < 0){
//             return 0;
//         }

//         int take = 0;

//         if(A[index] <= capacity){
//             take= A[index] + Knapsack(A, capacity-A[index], index-1);
//         }
//         int dontake= Knapsack(A, capacity, index-1);

//         return Math.max(take,dontake);

//     }
// // }
```

Q1. Cutting a Rod

```
// public class Solution {
//     int[] A;
//     public int solve(int[] A) {
//         this.A= A;
//         int C= A.length;
//         //cpacity = C
//         //happiness = A[i]
//         //weight = i+1
//         int[][] DP = new int[C+1][C+1];
//         for(int i=0; i<C+1; i++){
//             for(int j=0; j<C+1; j++){
//                 DP[i][j] = -1;
//             }
//         }
//         return Knapsack0inf(C, C-1, DP);

//     }

//     public int Knapsack0inf(int C, int index, int[][] DP){
//         if(index < 0){
//             return 0;
//         }
//         if(DP[index][C] != -1){
//             return DP[index][C];
//         }
//         int take = 0;
//         if(C >= (index+1)){
//             take= A[index] + Knapsack0inf(C-(index+1), index, DP);
//         }
//         int dontake = Knapsack0inf(C, index-1, DP);

//         DP[index][C] = Math.max(take, dontake);
//         return Math.max(take, dontake);
//     }
// }

// TC -> O(N*N)  N represent lenght of array and capacity
// SC -> O(N*N)

//another approch
public class Solution {
    int[] A;
    public int solve(int[] A) {
        this.A= A;
        int C= A.length;
        //cpacity = C
        //happiness = A[i]
        //weight = i+1
        int[] DP = new int[C+1];
        for(int i=0; i<C+1; i++){
            DP[i]=-1;
        }
        return maxRod(C, DP);
    }

    public int maxRod(int C, int[] DP){
        if(C == 0){
            return 0;
        }

        if(DP[C] != -1){
            return DP[C];
        }
        int profit= 0;
        for(int cut= 1 ; cut <= C; cut++){
            int index= cut-1;
            profit= Math.max(profit, A[index] +maxRod(C-cut, DP));
        }
        DP[C]= profit;

        return profit;
    }
}

// TC -> O(N*N)  N represent lenght of array and capacity
// SC -> O(N)
```

Q2. Coin Sum Infinite

```
public class Solution {
    int[][] DP;
    int mod = 1000007;
    int[] A;
    public int coinchange2(int[] A, int B) {
        this.A = A;
        int N= A.length;
        DP= new int[N][B+1];
        for(int i=0; i<N; i++){
            for(int j=0; j<B+1; j++){
                DP[i][j]= -1;
            }
        }
        return coinsum0inf(N-1, B);

    }
    public int coinsum0inf(int index, int total){
        if(total == 0){
            return 1;    //when reached last we got 1 way
        }
        if(index < 0){
            return 0;
        }

        if(DP[index][total] != -1){
            return DP[index][total];
        }

        int take= 0;
        if(total >= A[index]){
            take = coinsum0inf(index, total-A[index]);
        }
        int dontake = coinsum0inf(index-1, total);

        int ways = (take + dontake)% mod;
        DP[index][total]= ways;
        return ways;
    }
}

// TC -> O(N * B)
// SC -> O(N * B)
```

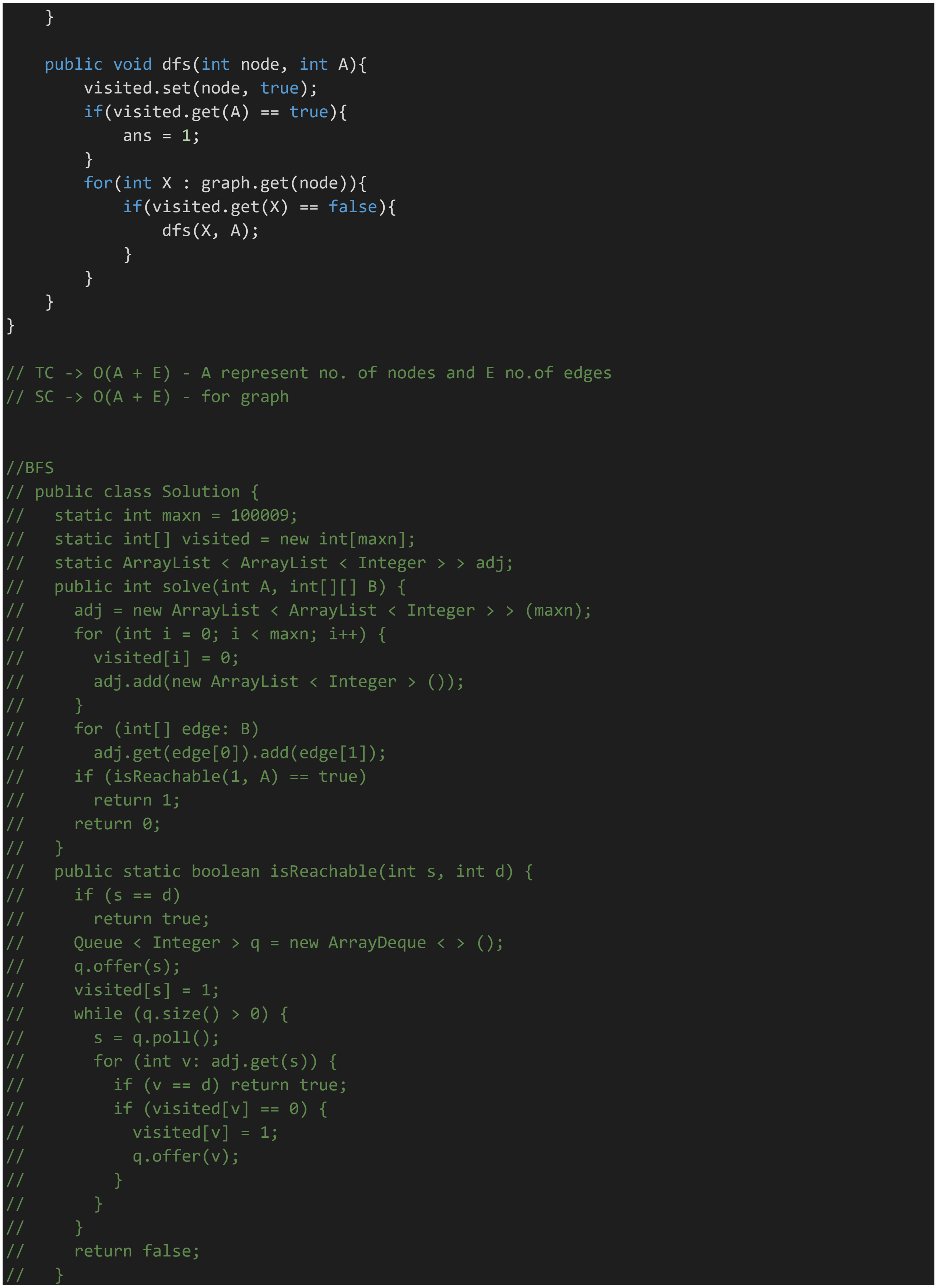
Q3. 0-1 Knapsack II

Given two integer arrays **A** and **B** of size **N** each which
represent **values** and **weights** associated with **N** items respectively.
Also given an integer **C** which represents knapsack capacity.
Find out the **maximum value** subset of **A** such that sum of the weights of this subset is smaller than or equal to **C**.
NOTE: You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

```
//o-1 Knapsack II approach
public class Solution {
    int[] A;
    int[] B;
    int[][] DP;
    public int solve(int[] A, int[] B, int C) {
        this.A=A;
        this.B=B;
        int N= A.length;
        int MH=0;
        for(int i=0; i<N; i++){
            MH += A[i];
        }
        DP= new int[N][MH+1];
        for(int i=0; i<N; i++){
            for(int j=0; j<MH+1; j++){
                DP[i][j]= -1;
            }
        }

        for(int i=MH; i>=0; i--){
            Knapsack2(N-1, i);
            if(DP[N-1][i] <= C){
                return i;
            }
        }
        return -1;
    }

    public int Knapsack2(int index, int MH){
        // System.out.println(index +" "+ MH);
        if(MH == 0){
            return 0;
        }
        if(index < 0){
            return 1000000000;
        }
        if(DP[index][MH] != -1){
            return DP[index][MH];
        }
        // System.out.println(index +" "+ MH);
        int take =1000000000;
        int dontake= Knapsack2(index-1 , MH);
        if(MH - A[index] >= 0){
            take = B[index] + Knapsack2(index-1, MH-A[index]);
        }
        int mincapa= Math.min(take, dontake);
        DP[index][MH]= mincapa;
        return mincapa;
    }
}

// TC ->(N * MH)
// SC ->(N * MH)
```

Q1. Path in Directed Graph

```
//DFS
public class Solution {
    ArrayList<ArrayList<Integer>> graph = new ArrayList<ArrayList<Integer>>();
    ArrayList<Boolean> visited = new ArrayList<Boolean>();
    int ans =0;
    public int solve(int A, ArrayList<ArrayList<Integer>> B) {
        int E = B.size();

        for(int i=0; i<=A; i++){
            graph.add(new ArrayList<Integer>());  //SC->(A)
        }
        for(int i=0; i<E; i++){               //SC -(E)
            int start = B.get(i).get(0);
            int end = B.get(i).get(1);
            graph.get(start).add(end);
        }
        for(int i=0; i<=A; i++){
            visited.add(false);
        }
        dfs(1, A);
        return ans;
```



```
    }

    public void dfs(int node, int A){
        visited.set(node, true);
        if(visited.get(A) == true){
            ans = 1;
        }
        for(int X : graph.get(node)){
            if(visited.get(X) == false){
                dfs(X, A);
            }
        }
    }
}

// TC -> O(A + E) - A represent no. of nodes and E no.of edges
// SC -> O(A + E) - for graph


//BFS
// public class Solution {
//   static int maxn = 100009;
//   static int[] visited = new int[maxn];
//   static ArrayList < ArrayList < Integer > > adj;
//   public int solve(int A, int[][] B) {
//     adj = new ArrayList < ArrayList < Integer > > (maxn);
//     for (int i = 0; i < maxn; i++) {
//       visited[i] = 0;
//       adj.add(new ArrayList < Integer > ());
//     }
//     for (int[] edge: B)
//       adj.get(edge[0]).add(edge[1]);
//     if (isReachable(1, A) == true)
//       return 1;
//     return 0;
//   }
//   public static boolean isReachable(int s, int d) {
//     if (s == d)
//       return true;
//     Queue < Integer > q = new ArrayDeque < > ();
//     q.offer(s);
//     visited[s] = 1;
//     while (q.size() > 0) {
//       s = q.poll();
//       for (int v: adj.get(s)) {
//         if (v == d) return true;
//         if (visited[v] == 0) {
//           visited[v] = 1;
//           q.offer(v);
//         }
//       }
//     }
//     return false;
//   }
// }
```

Q2. Shortest Distance in a Maze

```
//from TA with BFS
public class Solution {
    int maxn = 100009;
    int[] dx = new int[] { -1, 1, 0, 0 };
    int[] dy = new int[] { 0, 0, -1, 1 };
    public boolean inside(int x, int y, int n, int m) {
        return (x >= 0 && x <= n - 1 && y >= 0 && y <= m - 1);
    }
    public int solve(int[][] A, int[] B, int[] C) {
        return findMinDist(A, B, C);
    }
    public int findMinDist(int[][] maze, int[] start, int[] destination) {
        int n = maze.length;
        int m = maze[0].length;
        int sx = start[0];
        int sy = start[1];
        int ex = destination[0];
        int ey = destination[1];
        int[][] v = new int[n][m];  //to store distance
        for (int[] row: v)
            Arrays.fill(row, Integer.MAX_VALUE);  //fills every array slot into
Integer.MAX_VALUE
        Queue < Pair > pq = new LinkedList<>();
        int i;
        int d, d1;
        int x, y;
        int x1, y1;
        int x2, y2;
        pq.offer(new Pair(0, sx, sy));
        while (pq.size() != 0 && v[ex][ey] == Integer.MAX_VALUE) {
            Pair temp = pq.poll();
            x = temp.b;             //current x
            y = temp.c;             //current y
            d = temp.a;             //distance
            if (v[x][y] <= d) {
                continue;
            } else {
                v[x][y] = d;
            }
            for (i = 0; i < 4; ++i) {
                x1 = x;
                y1 = y;
                d1 = 0;
                while (true) {
                    x2 = x1 + dx[i];
                    y2 = y1 + dy[i];
                    if (inside(x2, y2, n, m) == true && maze[x2][y2] == 0) {
                        x1 = x2;
                        y1 = y2;
                        ++d1;
                    } else {
                        break;
                    }
                }
                if (d1 > 0 && v[x1][y1] == Integer.MAX_VALUE) {
                    pq.offer(new Pair(d + d1, x1, y1));
                }
            }
        }
        int res = -1;
        if (v[ex][ey] != Integer.MAX_VALUE)
            res = v[ex][ey];
        return res;
    }
}
class Pair {
    int a, b, c;
    public Pair(int u, int v, int w) {
        a = u;          //distance
        b = v;          //current x
        c = w;          //current y
    }
}
```

Q3. Cycle in Directed Graph

```
public class Solution {
    ArrayList<ArrayList<Integer>> AL = new ArrayList<ArrayList<Integer>>();
    ArrayList<Boolean> visited = new ArrayList<Boolean>();
    ArrayList<Boolean> path = new ArrayList<Boolean>();
    int cycle=0;
    public int solve(int A, ArrayList<ArrayList<Integer>> B) {
        int edges = B.size();
        for(int i=0; i<=A; i++){             //SC->A
            AL.add(new ArrayList<Integer>());
            visited.add(false);
            path.add(false);
        }

        for(int i=0; i<edges; i++){        //Sc-> edge
            int start= B.get(i).get(0);
            int  end= B.get(i).get(1);
            AL.get(start).add(end);
        }

        for(int i=1; i<=A; i++){
            if(visited.get(i) == false){
                DFS(i);
            }
        }
        return cycle;
    }
    public void DFS(int node){
        if(visited.get(node) == false){
            visited.set(node, true);
        }
        path.set(node, true);
        for(int X: AL.get(node)){
            if(path.get(X) == true){
                cycle =1;
            }
            if(visited.get(X) == false){
                DFS(X);
            }
        }
        path.set(node, false);
    }
}

// TC -> O(N+M) N represent no. of nodes and M no.of edges
// SC -> O(V + E)
```

Q4. Number of islands

```
public class Solution {

    public int solve(ArrayList<ArrayList<Integer>> A) {
        int N= A.size();
        int M= A.get(0).size();
        int islands = 0;
        ArrayList<ArrayList<Integer>> visited = new ArrayList<ArrayList<Integer>>();
        for(int i=0; i<N; i++){
            visited.add(new ArrayList<Integer>());
        }
        for(int i=0; i<N; i++){
            for(int j=0; j<M; j++){
                visited.get(i).add(0);
            }
        }
        ArrayList<ArrayList<Integer>> Direction = new ArrayList<ArrayList<Integer>>();

        for(int i=-1; i<2; i++){
            for(int j=-1; j<2; j++){
                if(i == 0 && j == 0){
                    continue;
                }
                Direction.add(new ArrayList<>(Arrays.asList(i,j)));
            }
        }
        // Direction.add(new ArrayList<>(Arrays.asList(-1, 0)));
        // Direction.add(new ArrayList<>(Arrays.asList(0, -1)));
        // Direction.add(new ArrayList<>(Arrays.asList(1, 0)));
        // Direction.add(new ArrayList<>(Arrays.asList(0, 1)));
        // Direction.add(new ArrayList<>(Arrays.asList(-1, -1)));
        // Direction.add(new ArrayList<>(Arrays.asList(1, 1)));
        // Direction.add(new ArrayList<>(Arrays.asList(-1, 1)));
        // Direction.add(new ArrayList<>(Arrays.asList(1, -1)));

        for(int R=0; R<N; R++){
            for(int C=0; C<M; C++){
                if(visited.get(R).get(C) == 0 && A.get(R).get(C) == 1){
                    visited.get(R).set(C, 1);
                    DFS(R, C, A, visited, Direction, N, M);
                    islands++;
                }
            }
        }
        return islands;
    }

    public void DFS(int R, int C, ArrayList<ArrayList<Integer>> A,
ArrayList<ArrayList<Integer>> visited, ArrayList<ArrayList<Integer>> Direction, int N, int M){
        for(int i=0; i<8; i++){
            int nR = R + Direction.get(i).get(0);
            int nC = C + Direction.get(i).get(1);
            if(0 <= nR && nR < N  && 0 <= nC && nC < M){
                if(visited.get(nR).get(nC) == 0 && A.get(nR).get(nC) == 1){
                    visited.get(nR).set(nC, 1);
                    DFS(nR, nC, A, visited, Direction, N, M);
                }
            }
        }
    }
}

// TC -> O(R*C)
// SC -> O(R*C)
```

Q1. Possibility of Finishing

```
//Kahn's Algo (with BFS Methos)
public class Solution {
    public int solve(int A, ArrayList<Integer> B, ArrayList<Integer> C) {
        ArrayList<ArrayList<Integer>> AL = new ArrayList<ArrayList<Integer>>();

        int edges= B.size();
        int[] indgree = new int[A+1];

        for(int i=0; i<=A; i++){              //SC->A
            AL.add(new ArrayList<Integer>());
        }

        for(int i=0; i<edges; i++){         //SC -> V+E
            int start = B.get(i);
            int end = C.get(i);
            AL.get(start).add(end);
            indgree[end]++;
        }

        Queue<Integer> q =  new LinkedList<Integer>();      //SC->A
        ArrayList<Integer> ans = new ArrayList<Integer>();

        for(int i=1; i<=A; i++){
            if(indgree[i] == 0){
                q.offer(i);
            }
        }

        while(q.size() > 0){
            int remove = q.poll();
            ans.add(remove);
            for(int X: AL.get(remove)){
                indgree[X]--;
                if(indgree[X] == 0){
                    q.offer(X);
                }
            }
        }
        if(ans.size() == A){
            return 1;
        }
        return 0;
    }
}

// TC -> O(V + E)
// SC -> O(V + E)
```

Q2. Shortest Distance in a Maze

```
//from TA with BFS
public class Solution {
    int maxn = 100009;
    int[] dx = new int[] { -1, 1, 0, 0 };
    int[] dy = new int[] { 0, 0, -1, 1 };
    public boolean inside(int x, int y, int n, int m) {
        return (x >= 0 && x <= n - 1 && y >= 0 && y <= m - 1);
    }
    public int solve(int[][] A, int[] B, int[] C) {
        return findMinDist(A, B, C);
    }
    public int findMinDist(int[][] maze, int[] start, int[] destination) {
        int n = maze.length;
        int m = maze[0].length;
        int sx = start[0];
        int sy = start[1];
        int ex = destination[0];
        int ey = destination[1];
        int[][] v = new int[n][m];  //to store distance
        for (int[] row: v)
            Arrays.fill(row, Integer.MAX_VALUE);  //fills every array slot into 
Integer.MAX_VALUE
        Queue < Pair > pq = new LinkedList<>();
        int i;
        int d, d1;
        int x, y;
        int x1, y1;
        int x2, y2;
        pq.offer(new Pair(0, sx, sy));
        while (pq.size() != 0 && v[ex][ey] == Integer.MAX_VALUE) {
            Pair temp = pq.poll();
            x = temp.b;             //current x
            y = temp.c;             //current y
            d = temp.a;             //distance
            if (v[x][y] <= d) {
                continue;
            } else {
                v[x][y] = d;
            }
            for (i = 0; i < 4; ++i) {
                x1 = x;
                y1 = y;
                d1 = 0;
                while (true) {
                    x2 = x1 + dx[i];
                    y2 = y1 + dy[i];
                    if (inside(x2, y2, n, m) == true && maze[x2][y2] == 0) {
                        x1 = x2;
                        y1 = y2;
                        ++d1;
                    } else {
                        break;
                    }
                }
                if (d1 > 0 && v[x1][y1] == Integer.MAX_VALUE) {
                    pq.offer(new Pair(d + d1, x1, y1));
                }
            }
        }
        int res = -1;
        if (v[ex][ey] != Integer.MAX_VALUE)
            res = v[ex][ey];
        return res;
    }
}
class Pair {
    int a, b, c;
    public Pair(int u, int v, int w) {
        a = u;          //distance
        b = v;          //current x
        c = w;          //current y
    }
}
```

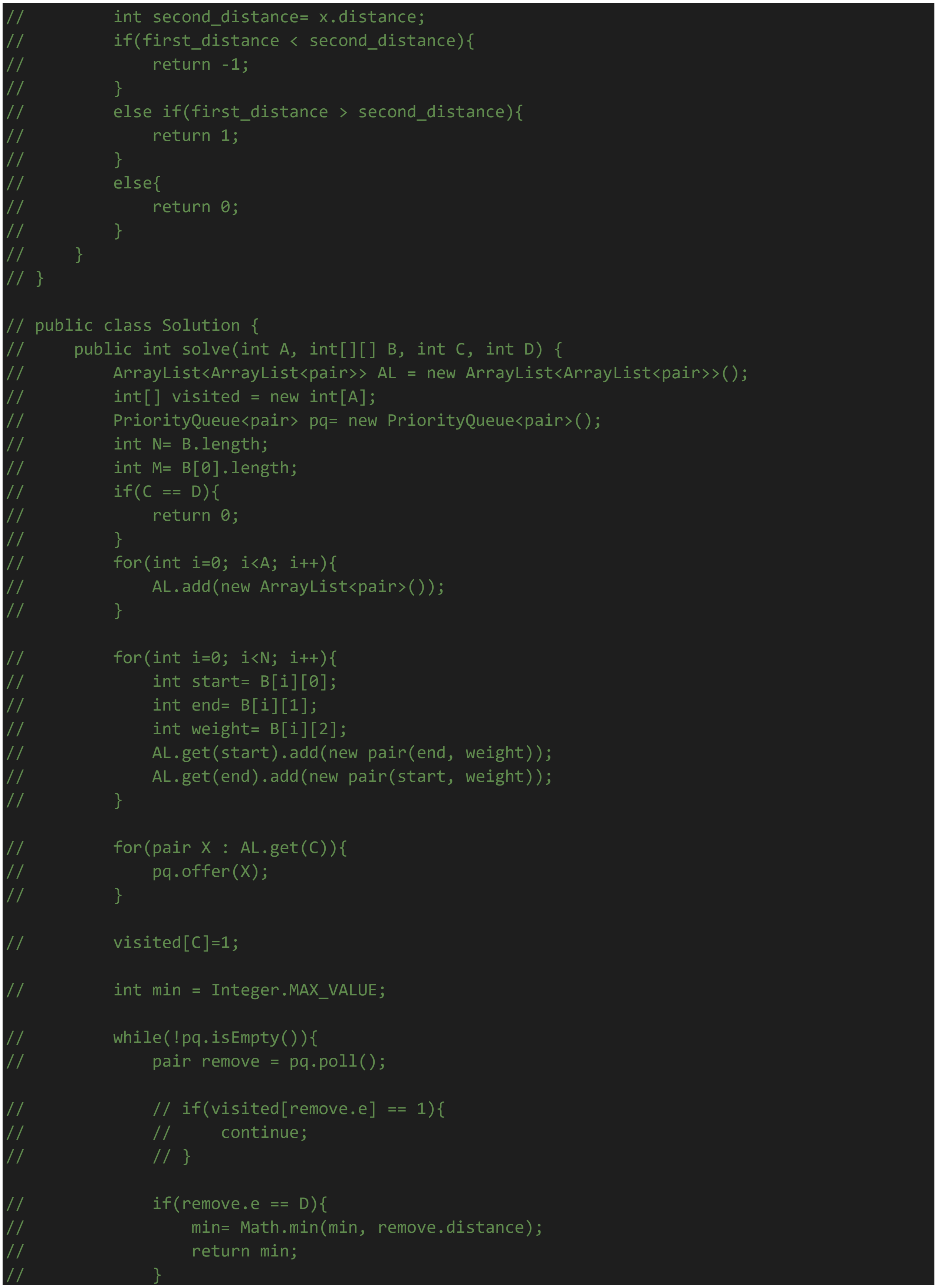
Q3. Rotten Oranges

```java
class Pair{
    int time;
    int R;
    int C;
    Pair(int X, int Y, int Z){
        time= X;
        R= Y;
        C= Z;
    }
}
public class Solution {
    public int solve(ArrayList<ArrayList<Integer>> A) {
        int N= A.size();
        int M= A.get(0).size();
        ArrayList<ArrayList<Integer>> Direction = new ArrayList<ArrayList<Integer>>();

        Queue<Pair> q = new LinkedList<>();
        Direction.add(new ArrayList<>(Arrays.asList(-1,0)));
        Direction.add(new ArrayList<>(Arrays.asList(0,-1)));
        Direction.add(new ArrayList<>(Arrays.asList(1,0)));
        Direction.add(new ArrayList<>(Arrays.asList(0,1)));

        int mintime =0;
        for(int R=0; R<N; R++){
            for(int C=0; C<M; C++){
                if(A.get(R).get(C) == 2){
                    q.offer(new Pair(0,R,C));  //at start we added rotten in queue
                }
            }
        }

        while(q.size() > 0){
            Pair rotten = q.poll();
            mintime = rotten.time;

            for(int i= 0; i<Direction.size(); i++){          //move to every direction
                int nR = rotten.R + Direction.get(i).get(0);
                int nC = rotten.C + Direction.get(i).get(1);
                if(0<= nR && nR <N && 0<=nC && nC<M){
                    if(A.get(nR).get(nC) == 1){
                        A.get(nR).set(nC, 2);
                        q.offer(new Pair(rotten.time +1, nR, nC));
                    }
                }
            }
        }

        for(int R=0; R<N; R++){
            for(int C=0; C<M; C++){
                if(A.get(R).get(C) == 1){
                    return -1;
                }
            }
        }
        return mintime;

    }
}

// TC -> O(N*M)
// SC -> O(N*M)
```

Q4. Another BFS

Given a weighted undirected graph having A nodes, a source node C and destination node D.
Find the shortest distance from C to D and if it is impossible to reach node D from C then return -1.
You are expected to do it in Time Complexity of O(A + M).
Note:
There are no self-loops in the graph.
No multiple edges between two pair of vertices.
The graph may or may not be connected.
Nodes are Numbered from 0 to A-1.
Your solution will run on multiple testcases. If you are using global variables make sure to clear them.

```
//Normal BFS approach after moving 2 to 1 , 1
class pair{
    int Destination;
    int Distance;
    pair(int a, int b){
        Destination= a;
        Distance= b;
    }
}
public class Solution {
    public int solve(int A, int[][] B, int C, int D) {
        int maxn = 2*A;    //if we have 10 nodes then maximumit can go till 19(9+10)
        ArrayList<ArrayList<pair>> AL= new ArrayList<ArrayList<pair>>(maxn);
        Queue<pair> q= new LinkedList<>();
        if(C == D){
            return 0;
        }

        int N= B.length;
        int M= B[0].length;
        for(int i=0; i<maxn; i++){
            AL.add(new ArrayList<pair>());
        }

        for(int i=0; i<N; i++){
            int start= B[i][0];
            int end= B[i][1];
            int weight= B[i][2];
            if(weight == 1){
                AL.get(start).add(new pair(end, weight));
                AL.get(end).add(new pair(start, weight));
            }
            else{
                AL.get(start).add(new pair(start+A, 1));
                AL.get(start+A).add(new pair(end, 1));
                AL.get(end).add(new pair(end+A, 1));
                AL.get(end+A).add(new pair(start, 1));
            }
        }
// 0  X  5  Y  0(this 0 is from start, we using 2 dummy nodes(single direction))

        for(pair X: AL.get(C)){
            q.offer(X);
        }
        int[] visited = new int[maxn];
        visited[C]= 1;

        while(!q.isEmpty()){
            pair remove = q.poll();
            if(remove.Destination == D){
                return remove.Distance;
            }
            // visited[remove.Destination]= 1;
            for(pair X: AL.get(remove.Destination)){
                if(visited[X.Destination] == 0){
                    q.offer(new pair(X.Destination, remove.Distance +1));
                    visited[X.Destination]= 1;
                }
            }
        }
        return -1;
    }
}

// TC -> O(V + E)
// SC -> O(2A)



//Dijkstra
// class pair implements Comparable<pair>{
//     int e;
//     int distance;
//     pair(int a, int b){
//         e= a;
//         distance= b;
//     }
//     public int compareTo(pair x){
//         int first_distance= this.distance;
```



```
//         int second_distance= x.distance;
//         if(first_distance < second_distance){
//             return -1;
//         }
//         else if(first_distance > second_distance){
//             return 1;
//         }
//         else{
//             return 0;
//         }
//     }
// }

// public class Solution {
//     public int solve(int A, int[][] B, int C, int D) {
//         ArrayList<ArrayList<pair>> AL = new ArrayList<ArrayList<pair>>();
//         int[] visited = new int[A];
//         PriorityQueue<pair> pq= new PriorityQueue<pair>();
//         int N= B.length;
//         int M= B[0].length;
//         if(C == D){
//             return 0;
//         }
//         for(int i=0; i<A; i++){
//             AL.add(new ArrayList<pair>());
//         }

//         for(int i=0; i<N; i++){
//             int start= B[i][0];
//             int end= B[i][1];
//             int weight= B[i][2];
//             AL.get(start).add(new pair(end, weight));
//             AL.get(end).add(new pair(start, weight));
//         }

//         for(pair X : AL.get(C)){
//             pq.offer(X);
//         }

//         visited[C]=1;

//         int min = Integer.MAX_VALUE;

//         while(!pq.isEmpty()){
//             pair remove = pq.poll();

//             // if(visited[remove.e] == 1){
//             //     continue;
//             // }

//             if(remove.e == D){
//                 min= Math.min(min, remove.distance);
//                 return min;
//             }
//             visited[remove.e] = 1;
//             for(pair X : AL.get(remove.e)){
//                 int d= remove.distance + X.distance;
//                 if(visited[X.e] == 0){
//                     pq.offer(new pair(X.e, d));
//                 }
//             }
//         }
//         return -1;
//     }
// }

// TC -> O(V + E + ElogE)
// or
// TC -> O(V + E + ElogV)
// SC -> O(V + E)

// or other trick is
// N= V+E;
// TC -> O(NlogN)
```

Q5. Topological Sort

```
// Multi source BFS
public class Solution {
    public ArrayList<Integer> solve(int A, ArrayList<ArrayList<Integer>> B) {
        ArrayList<ArrayList<Integer>> AL = new ArrayList<ArrayList<Integer>>();
        boolean[] visited = new boolean[A+1];
        ArrayList<Integer> ans = new ArrayList<Integer>();
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
        int[] indgree = new int[A+1];

        for(int i=0; i<=A; i++){
            AL.add(new ArrayList<Integer>());
        }

        int edges= B.size();
        for(int i=0; i<edges; i++){           //TC -> A+edges
            int start= B.get(i).get(0);
            int end= B.get(i).get(1);
            AL.get(start).add(end);
            indgree[end]++;
        }
        for(int i=1; i<=A; i++){
            if(indgree[i] == 0){
                pq.offer(i);
            }
        }

        while(!pq.isEmpty()){             //TC -> ((A*logA) + edges)
            int node= pq.poll();
            ans.add(node);
            for(int X : AL.get(node)){
                indgree[X]--;
                if(indgree[X] == 0){
                    pq.offer(X);
                }
            }
        }
        if(ans.size() != A){
            return new ArrayList<Integer>();
        }
        return ans;
    }
}
// TC -> O((A*logA) + edges)
// SC -> O(V + E)
```

Q1. Commutable Islands

```
// Minimum Spanning Tree (MST)
//Prim's Algo
class pair implements Comparable<pair>{
    int cost;
    int destination;
    pair(int a, int b){
        cost= a;
        destination= b;
    }

    public int compareTo(pair X){
        int first_cost = this.cost;
        int second_cost = X.cost;
        if(first_cost < second_cost){
            return -1;
        }
        else if(first_cost > second_cost){
            return 1;
        }
        else{
            return 0;
        }
    }
}
public class Solution {
    public int solve(int A, int[][] B) {
        ArrayList<ArrayList<pair>> AL = new ArrayList<ArrayList<pair>>();
        int[] visited = new int[A+1];
        PriorityQueue<pair> pq = new PriorityQueue<pair>();

        for(int i=0; i<=A; i++){
            AL.add(new ArrayList<pair>());
        }

        int edges= B.length;
        for(int i= 0; i<edges; i++){
            int start = B[i][0];
            int end = B[i][1];
            int w = B[i][2];
            AL.get(start).add(new pair(w, end));
            AL.get(end).add(new pair(w, start));
        }

        for(pair X: AL.get(1)){
            pq.offer(X);
        }

        visited[1]= 1;
        int totalmincost= 0;
        while(!pq.isEmpty()){
            pair remove = pq.poll();
            if(visited[remove.destination] == 1){
                continue;
            }
            totalmincost += remove.cost;
            visited[remove.destination]= 1;
            for(pair X: AL.get(remove.destination)){
                if(visited[X.destination] == 0){
                    pq.offer(X);
                }
            }
        }
        return totalmincost;
    }
}


// TC -> O(V+E+ E*logE)
// TC -> O(E*logV) <-base on chat gpt
// SC -> O(V+E)
```

Q2. Dijkstra

Given a **weighted undirected graph** having **A nodes** and **M weighted edges**, and a **source node C**.
You have to **find an integer array D of size A** such that:

- D[i]: Shortest distance from the **C node** to **node i**.
- If **node i** is not reachable from C then **-1**.

Note:

- There are no self-loops in the graph.
- There are no multiple edges between two pairs of vertices.
- The graph may or may not be connected.
- Nodes are numbered from 0 to A-1.
- Your solution will run on multiple test cases. If you are using global variables, make sure to clear them.

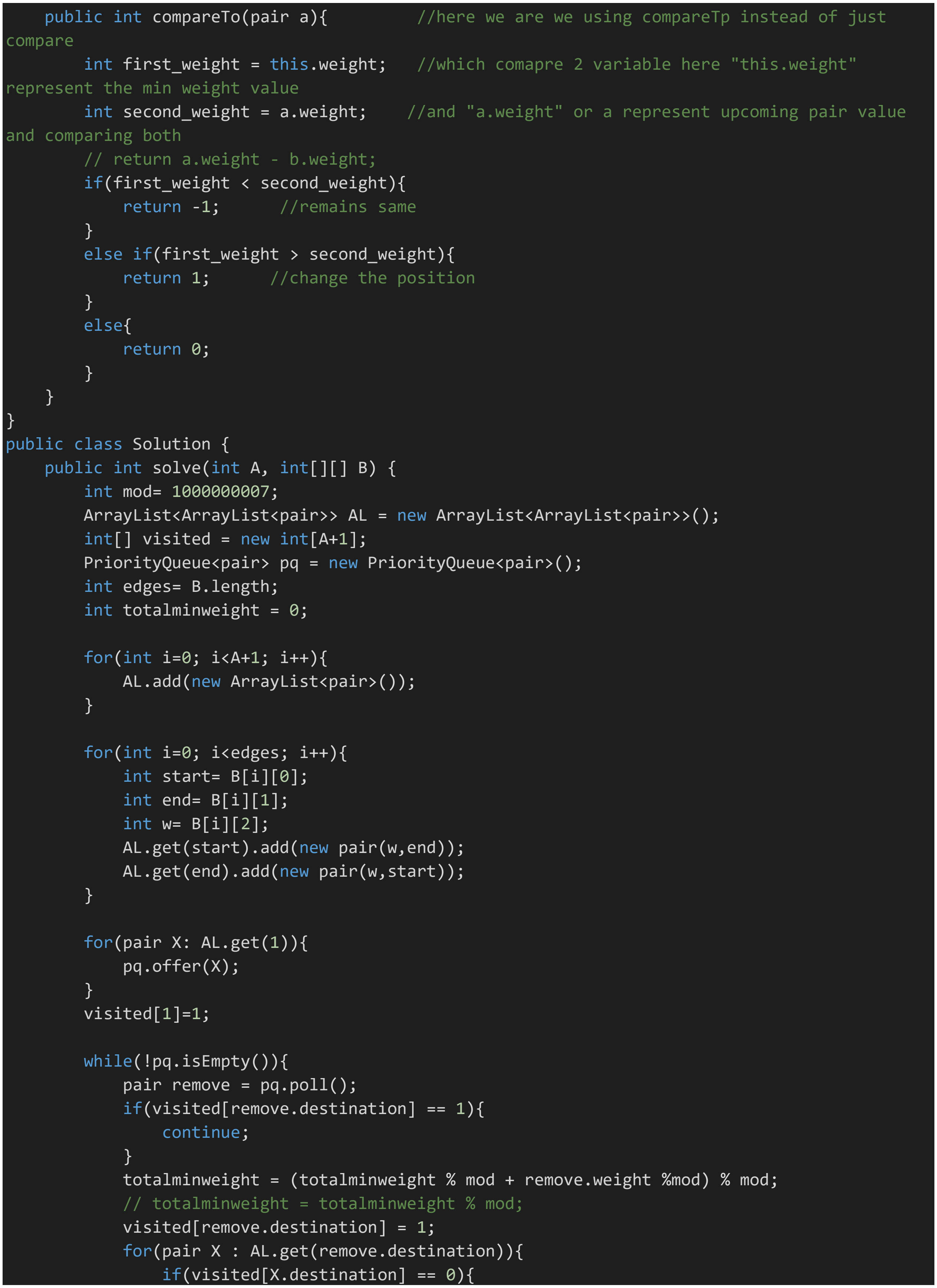
-
  ```
  //Dijkstra Algo
  ```

```
class pair implements Comparable<pair>{  //this is representation of pair's with min heap
    int Weight;
    int Destination;
    pair(int X, int Y){
        Weight= X;
        Destination= Y;
    }
    public int compareTo(pair a){       //here we are we using compareTp instead of just compare
        int first_weight= this.Weight;  //which comapre 2 variable here "this.weight" represent the min weight value
        int second_weight= a.Weight;    //and "a.weight" or a represent upcoming pair value and comparing both
        if(first_weight < second_weight){
            return -1;     //remains same
        }
        else if(first_weight > second_weight){
            return 1;      //change the position
        }
        else{
            return 0;
        }
    }
}
public class Solution {
    public int[] solve(int A, int[][] B, int C) {
        ArrayList<ArrayList<pair>> AL =  new ArrayList<ArrayList<pair>>();
        int[] DistArray = new int[A];
        PriorityQueue<pair> pq = new PriorityQueue<pair>();
        int edges=B.length;

        for(int i=0; i<A; i++){
            AL.add(new ArrayList<pair>());
        }

        for(int i=0; i<edges; i++){
            int start = B[i][0];
            int end = B[i][1];
            int w = B[i][2];
            AL.get(start).add(new pair(w, end));
            AL.get(end).add(new pair(w, start));
        }

        for(int i=0; i<A; i++){
            DistArray[i] = Integer.MAX_VALUE;
        }

        DistArray[C] = 0;

        pq.offer(new pair(0,C));   //just needs to add start node with 0 distance

        while(!pq.isEmpty()){
            pair remove = pq.poll();
            for(pair X: AL.get(remove.Destination)){
                int New_Dest = remove.Weight + X.Weight;  //added previous node weight and going forward
                if(New_Dest < DistArray[X.Destination]){  //if new dist smaller than old replace that
                    DistArray[X.Destination] = New_Dest;
                    pq.offer(new pair(New_Dest, X.Destination));  //add in heap with min new dist and node
                }
            }
        }

        for(int i=0; i<A; i++){
            if(DistArray[i] == Integer.MAX_VALUE){
                DistArray[i] = -1;
            }
        }

        return DistArray;
    }
}

// TC -> O(V + E + ElogE)
// or
// TC -> O(V + E + ElogV)
// SC -> O(V + E)

// or other trick is
// N= V+E;
// TC -> O(NlogN)
```

Q3. Construction Cost

Given a graph with **A** nodes and **C** weighted edges. Cost of constructing the graph is the sum of weights of all the edges in the graph.
Find the **minimum cost** of constructing the graph by selecting some given edges such that we can reach every other node from the **1st** node.
NOTE: Return the answer modulo **10^9+7** as the answer can be large.

```
// Minimum Spanning Tree (MST)
//Prim's Algo
class pair implements Comparable<pair>{  //this is representation of pair's with min heap
    int weight;
    int destination;
    public pair(int x, int y){
        weight = x;
        destination = y;
    }
    @Override
```



```
    public int compareTo(pair a){         //here we are we using compareTp instead of just
compare
        int first_weight = this.weight;   //which comapre 2 variable here "this.weight"
represent the min weight value
        int second_weight = a.weight;    //and "a.weight" or a represent upcoming pair value
and comparing both
        // return a.weight - b.weight;
        if(first_weight < second_weight){
            return -1;      //remains same
        }
        else if(first_weight > second_weight){
            return 1;      //change the position
        }
        else{
            return 0;
        }
    }
}
public class Solution {
    public int solve(int A, int[][] B) {
        int mod= 1000000007;
        ArrayList<ArrayList<pair>> AL = new ArrayList<ArrayList<pair>>();
        int[] visited = new int[A+1];
        PriorityQueue<pair> pq = new PriorityQueue<pair>();
        int edges= B.length;
        int totalminweight = 0;

        for(int i=0; i<A+1; i++){
            AL.add(new ArrayList<pair>());
        }

        for(int i=0; i<edges; i++){
            int start= B[i][0];
            int end= B[i][1];
            int w= B[i][2];
            AL.get(start).add(new pair(w,end));
            AL.get(end).add(new pair(w,start));
        }

        for(pair X: AL.get(1)){
            pq.offer(X);
        }
        visited[1]=1;

        while(!pq.isEmpty()){
            pair remove = pq.poll();
            if(visited[remove.destination] == 1){
                continue;
            }
            totalminweight = (totalminweight % mod + remove.weight %mod) % mod;
            // totalminweight = totalminweight % mod;
            visited[remove.destination] = 1;
            for(pair X : AL.get(remove.destination)){
                if(visited[X.destination] == 0){
                    pq.offer(X);
                }
            }
        }
        return (int)totalminweight;
    }
}

// TC -> O(V+E+ E*logE)
// TC -> O(E*logV) <-base on chat gpt
// SC -> O(V+E)
```

Q1. Connect ropes

```
public class Solution {
    public int solve(ArrayList<Integer> A) {
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
        for(int X : A){
            pq.offer(X);
        }
        int cost =0 ;
        while(pq.size() > 1){
            int first = pq.remove();
            int second = pq.remove();
            cost = cost + (first+second);
            pq.offer(first+second);
        }
        return cost;
    }
}

// TC -> O(NlogN) -> for insertion and deletion logn and for iterating over elements N
// SC -> O(N)
```

List

Q1. Palindrome List

```
public class Solution {
    public int lPalin(ListNode A) {

        if(A.next == null){
            return 1;
        }

        ListNode mid = findmid(A);
        ListNode B = reverse(mid.next);

        ListNode temp1= A;
        ListNode temp2= B;
        while(temp1 != null && temp2 != null){
            if(temp1.val != temp2.val){
                return 0;
            }
            temp1= temp1.next;
            temp2= temp2.next;
        }
        return 1;
    }

    public ListNode findmid(ListNode A){
        ListNode slow= A;
        ListNode fast= A;
        while(fast.next != null && fast.next.next != null){
            slow= slow.next;
            fast= fast.next.next;
        }
        return slow;
    }

    public ListNode reverse(ListNode head){
        ListNode h1= head;
        ListNode h2= head.next;
        ListNode temp;
        while(h2 != null){
            temp = h2.next;
            h2.next= h1;
            h1=h2;
            h2=temp;
        }
        head.next=null;
        head = h1;
        return h1;
    }
}

// TC -> O(N);
// SC -> O(1);
```

Q2. Reverse Linked List

```
public class Solution {
    public ListNode reverseList(ListNode A) {
        ListNode temp= A;
        ListNode h1=temp;
        ListNode h2=temp.next;
        while(h2 != null){
            temp=h2.next;
            h2.next=h1;
            h1=h2;
            h2=temp;
        }
        A.next=null;
        A=h1;
        return A;
    }
}

// TC -> O(N);
// SC -> O(1);
```

Q1. Middle element of linked list

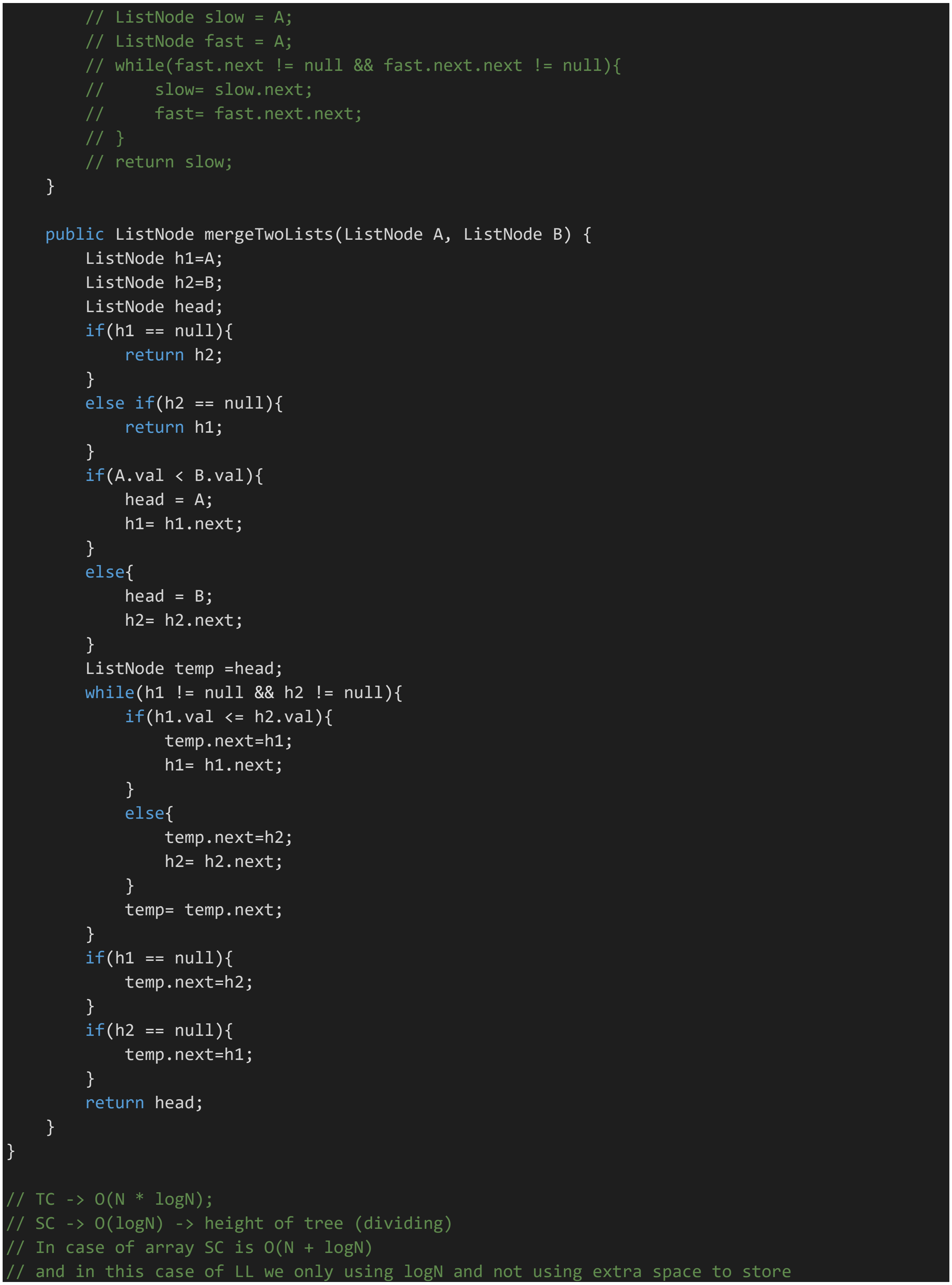
```
public class Solution {
    public int solve(ListNode A) {
        ListNode slow=A;
        ListNode fast=A;
        if (A.next == null){
            return A.val;
        }

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

        }

        return slow.val;
    }
}

// TC -> O(N)
// SC -> O(1)
```

Q2. Merge Two Sorted Lists

```
public class Solution {
    public ListNode mergeTwoLists(ListNode A, ListNode B) {
        ListNode h1=A;
        ListNode h2=B;
        ListNode head;
        if(h1 == null){
            return h2;
        }
        else if(h2 == null){
            return h1;
        }
        if(A.val < B.val){
            head = A;
            h1= h1.next;
        }
        else{
            head = B;
            h2= h2.next;
        }
        ListNode temp =head;
        while(h1 != null && h2 != null){
            if(h1.val <= h2.val){
                temp.next=h1;
                h1= h1.next;
            }
            else{
                temp.next=h2;
                h2= h2.next;
            }
            temp= temp.next;
        }
        if(h1 == null){
            temp.next=h2;
        }
        if(h2 == null){
            temp.next=h1;
        }
        return head;
    }
}


// TC -> O(N1+N2)
// SC -> O(1)
```

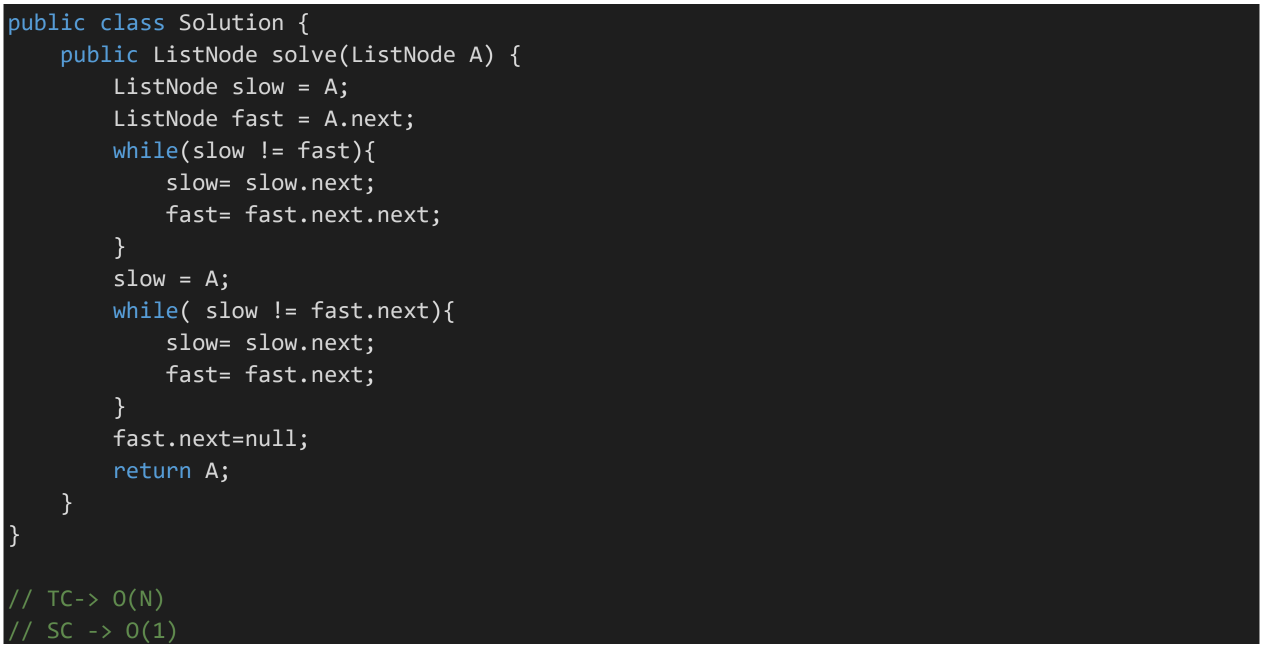
Q3. Sort List

```
public class Solution {
    public ListNode sortList(ListNode A) {
        if(A == null || A.next == null){
            return A;
        }
        ListNode mid = Mid_element(A);
        ListNode h2 = mid.next;
        // ListNode h1 = A;
        mid.next = null;
        ListNode h1 = A;
        h1 = sortList(A);
        h2 = sortList(h2);
        return mergeTwoLists(h1,h2);
        // return Meargesort(A);
    }

    public ListNode Mid_element(ListNode A) {
        ListNode slow=A;
        ListNode fast=A.next;
        while(fast != null && fast.next != null){
            slow= slow.next;
            fast= fast.next.next;
        }
        return slow;
```



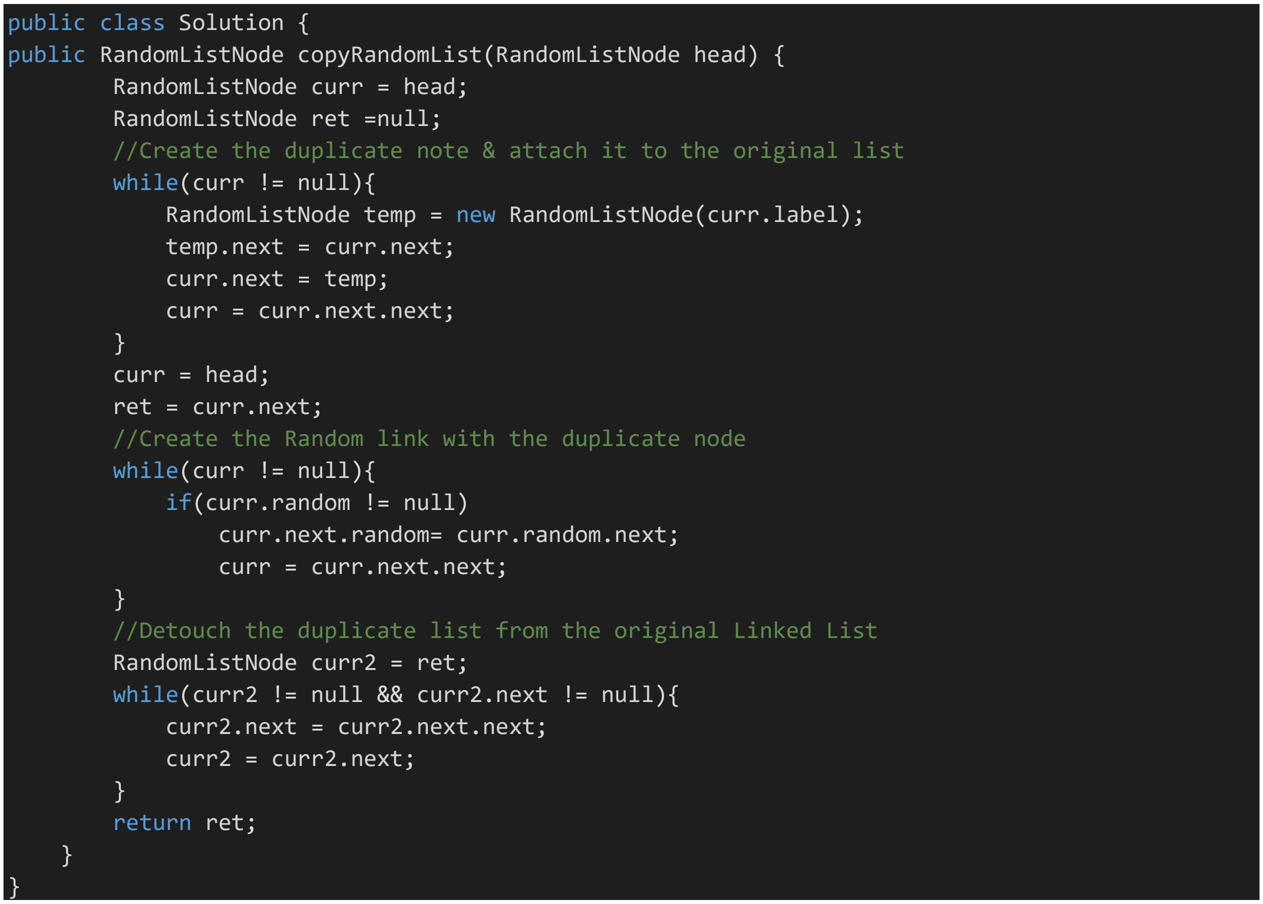
```
        // ListNode slow = A;
        // ListNode fast = A;
        // while(fast.next != null && fast.next.next != null){
        //     slow= slow.next;
        //     fast= fast.next.next;
        // }
        // return slow;
    }

    public ListNode mergeTwoLists(ListNode A, ListNode B) {
        ListNode h1=A;
        ListNode h2=B;
        ListNode head;
        if(h1 == null){
            return h2;
        }
        else if(h2 == null){
            return h1;
        }
        if(A.val < B.val){
            head = A;
            h1= h1.next;
        }
        else{
            head = B;
            h2= h2.next;
        }
        ListNode temp =head;
        while(h1 != null && h2 != null){
            if(h1.val <= h2.val){
                temp.next=h1;
                h1= h1.next;
            }
            else{
                temp.next=h2;
                h2= h2.next;
            }
            temp= temp.next;
        }
        if(h1 == null){
            temp.next=h2;
        }
        if(h2 == null){
            temp.next=h1;
        }
        return head;
    }
}

// TC -> O(N * logN);
// SC -> O(logN) -> height of tree (dividing)
// In case of array SC is O(N + logN)
// and in this case of LL we only using logN and not using extra space to store
```

Q4. Remove Loop from Linked List



```
public class Solution {
    public ListNode solve(ListNode A) {
        ListNode slow = A;
        ListNode fast = A.next;
        while(slow != fast){
            slow= slow.next;
            fast= fast.next.next;
        }
        slow = A;
        while( slow != fast.next){
            slow= slow.next;
            fast= fast.next;
        }
        fast.next=null;
        return A;
    }
}

// TC-> O(N)
// SC -> O(1)
```

Q1. Copy List



```
public class Solution {
public RandomListNode copyRandomList(RandomListNode head) {
        RandomListNode curr = head;
        RandomListNode ret =null;
        //Create the duplicate note & attach it to the original list
        while(curr != null){
            RandomListNode temp = new RandomListNode(curr.label);
            temp.next = curr.next;
            curr.next = temp;
            curr = curr.next.next;
        }
        curr = head;
        ret = curr.next;
        //Create the Random link with the duplicate node
        while(curr != null){
            if(curr.random != null)
                curr.next.random= curr.random.next;
                curr = curr.next.next;
        }
        //Detouch the duplicate list from the original Linked List
        RandomListNode curr2 = ret;
        while(curr2 != null && curr2.next != null){
            curr2.next = curr2.next.next;
            curr2 = curr2.next;
        }
        return ret;
    }
}
```

Q2. LRU Cache

```
//disscusion
public class Solution {

    int capacity=0;
    LRU cache;

    class LRU {
        int size=0;
        Node head,tail;
        HashMap<Integer,Node> hm= new HashMap();
        LRU(){
            head=new Node(-1,-1);
            tail=new Node(-1,-1);
            head.next=tail;
            tail.prev=head;
        }
    }

    class Node {
        public int val,key;
        public Node next,prev;
        Node(int x,int y) { val = x;key=y; next = null; }
    }

    public Solution(int capacity) {
        this.capacity=capacity;
        this.cache=new LRU();
    }

    public int get(int key) {
        if(cache.hm.containsKey(key)){
            Node curr=cache.hm.get(key);
            delFromPos(curr);
            insertAfterHead(cache.head,curr);
            return cache.hm.get(key).val;
        } else {
            return -1;
        }
    }

    public void set(int key, int value) {

         if(cache.hm.containsKey(key)){
            Node curr=cache.hm.get(key);
            curr.val=value;
            delFromPos(curr);
            insertAfterHead(cache.head,curr);
        } else {
            if(cache.hm.size()==capacity){
                Node delnode=cache.tail.prev;
                cache.hm.remove(delnode.key);
               // cache.hm.entrySet().removeIf(entry -> (delnode.val==entry.getValue().val));
```

```
                delFromPos(delnode);
                Node newNode=new Node(value,key);
                cache.hm.put(key,newNode);
                insertAfterHead(cache.head,newNode);
            } else {
                Node newNode=new Node(value,key);
                cache.hm.put(key,newNode);
                insertAfterHead(cache.head,newNode);
            }

        }

    }

    public void delFromPos(Node node){
        node.prev.next=node.next;
        node.next.prev=node.prev;
    }

    public void insertAfterHead(Node head,Node node){
        node.next=head.next;
        head.next=node;
        node.prev=head;
        node.next.prev=node;
    }
}
```

Stack

Q1. Passing game

```
public class Solution {
    public int solve(int A, int B, ArrayList<Integer> C) {
        Stack<Integer> st = new Stack<Integer>();
        st.push(B);
        int N=C.size();
        for(int i=0; i<N; i++){
            if(C.get(i) != 0){
                st.push(C.get(i));
            }
            else{
                st.pop();
            }
        }
        return st.peek();
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q2. Balanced Paranthesis

```
//using charAt(i)
public class Solution {
    public int solve(String A) {
        HashMap < Character, Character > mp = new HashMap < Character, Character > ();
        Stack < Character > st = new Stack < Character > ();
        mp.put(')', '(');
        mp.put('}', '{');
        mp.put(']', '[');
        for (int i = 0; i < A.length(); i++) {
            char c = A.charAt(i);
            if (c == '(' || c == '[' || c == '{') {
                // push any opening bracket into the stack
                st.push(c);
            } else if (st.empty() || st.peek() != mp.get(c)) {
                // check if the last unpaired opening bracket is of the same type
                // as the current closing bracket
                return 0;
            } else {
                st.pop();
            }
        }
        // checks if all the opening brackets are paired
        if (st.empty())
            return 1;
        return 0;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q3. Double Character Trouble

```
public class Solution {
    public String solve(String A) {
        Stack<Character> st= new Stack<Character>();
        int N=A.length();
        for(int i=0; i<N; i++){
            char ch=A.charAt(i);
            if(st.isEmpty() || st.peek()!=ch){
                st.push(ch);
            }
            else{
                st.pop();
            }
        }
        StringBuilder ans =new StringBuilder();
        while(st.isEmpty() != true){
            ans.append(st.peek());
            st.pop();
        }
        ans.reverse();
        return ans.toString();
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q4. Evaluate Expression

```
public class Solution {
    public int evalRPN(ArrayList<String> A) {
        Stack<Integer> st = new Stack<Integer>();
        int N= A.size();
        int a,b;
        int ans =0;
        if(N == 1){
            return Integer.parseInt(A.get(0));
        }
        for(int i=0; i<N; i++){
            if(A.get(i).equals("+")){
                b= st.pop();
                a= st.pop();
                ans= a + b;
                st.push(ans);
            }
            else if(A.get(i).equals("-")){
                b= st.pop();
                a= st.pop();
                ans= a - b;
                st.push(ans);
            }
            else if(A.get(i).equals("*")){
                b= st.pop();
                a= st.pop();
                ans= a * b;
                st.push(ans);
            }
            else if(A.get(i).equals("/")){
                b= st.pop();
                a= st.pop();
                ans= a / b;
                st.push(ans);
            }
            else{
                st.push(Integer.parseInt(A.get(i)));
            }
        }
        return ans;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q1. Nearest Smaller Element

```
public class Solution {
    public ArrayList<Integer> prevSmaller(ArrayList<Integer> A) {
        int N= A.size();
        Stack<Integer> st= new Stack<Integer>();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        // ans.add(-1);
        // st.push(A.get(0));
        for(int i=0; i<N; i++){
            while(!st.isEmpty() && A.get(i) <= st.peek()){
                st.pop();
            }
            if(st.isEmpty()){
                ans.add(-1);
            }
            else{
                ans.add(st.peek());
            }
            st.push(A.get(i));
        }
        return ans;
    }
}

// TC -> O(N);
// SC -> O(N);
```

Q2. Largest Rectangle in Histogram

```
public class Solution {
    public int largestRectangleArea(int[] A) {
        int N= A.length;
        int[] NESL = new int[N];
        int[] NESR = new int[N];
        NESL[0]= -1;
        NESR[N-1]=-1;
        Stack<Integer> st1 = new Stack<Integer>();
        Stack<Integer> st2 = new Stack<Integer>();
        st1.push(0);
        st2.push(N-1);

        if(N == 1){
            return A[0];
        }
        for(int i=1; i<N; i++){
            while(!st1.isEmpty() && A[i] <= A[st1.peek()]){
                st1.pop();
            }
            if(st1.isEmpty()){
                NESL[i]= -1;
            }
            else{
                NESL[i]= st1.peek();
            }
            st1.push(i);
        }

        for(int i=N-2; i>=0; i--){
            while(!st2.isEmpty() && A[i] <= A[st2.peek()]){
                st2.pop();
            }
            if(st2.isEmpty()){
                NESR[i]= -1;
            }
            else{
                NESR[i]= st2.peek();
            }
            st2.push(i);
        }

        int ans=Integer.MIN_VALUE;
        int area, s, e;
        for(int i=0; i<N; i++){
            if(NESL[i]== -1){
                s= 0;
            }
            else{
                s= NESL[i]+1;
            }

            if(NESR[i] == -1){
                e= N-1;
            }
            else{
                e= NESR[i]-1;
            }

            area= A[i] * (e -s +1);
            ans = Math.max(ans,area);
        }
        return ans;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q3. MAX and MIN

Given an array of integers **A**.

The value of an array is computed as the difference between the **maximum** element in the array and the **minimum** element in the array **A**.
Calculate and return the sum of values of all possible subarrays of A **modulo 10^9+7**.

```
public class Solution {
    public int solve(int[] A) {
        Stack<Integer> st1 = new Stack<Integer>();
        Stack<Integer> st2 = new Stack<Integer>();
        Stack<Integer> st3 = new Stack<Integer>();
        Stack<Integer> st4 = new Stack<Integer>();
        int N = A.length;
        int[] NSEL = new int[N];
        int[] NSER = new int[N];
        int[] NGEL = new int[N];
        int[] NGER = new int[N];
        // 8 4 10 4 8
        // 0 1 2  3 4
        // st -   1  3    4<=4
        // nsel - -1 -1 1 1(3) 3
        //for NSEL
        for (int i = 0; i < N; i++) {
            while (!st1.isEmpty() && A[i] <= A[st1.peek()]) {  //for removing we should take = 
also
                st1.pop();
            }
            if (st1.isEmpty()) {
                NSEL[i] = -1;
            } else {
                NSEL[i] = st1.peek();
            }
            st1.push(i);
        }
        //for NSER
        for (int i = N - 1; i >= 0; i--) {
            while (!st2.isEmpty() && A[i] <= A[st2.peek()]) {
                st2.pop();
            }
            if (st2.isEmpty()) {
                NSER[i] = N ;
            } else {
                NSER[i] = st2.peek();
            }
            st2.push(i);
        }
        //for NGEL
        for (int i = 0; i < N; i++) {
            while (!st3.isEmpty() && A[i] >= A[st3.peek()]) {
                st3.pop();
            }
            if (st3.isEmpty()) {
                NGEL[i] = -1;
            } else {
                NGEL[i] = st3.peek();
            }
            st3.push(i);
        }
        //for NGER
        for (int i = N - 1; i >= 0; i--) {
            while (!st4.isEmpty() && A[i] >= A[st4.peek()]) {
                st4.pop();
            }
            if (st4.isEmpty()) {
                NGER[i] = N ;
            } else {
                NGER[i] = st4.peek();
            }
            st4.push(i);
        }

        long subArray_Min, subArray_Max, ans;
        long sum = 0;
        int mod = 1000000007;
        for (int i = 0; i < N; i++) {
            subArray_Min = ((long)(i - NSEL[i]) * (NSER[i] - i)) ; //NSEL+1 to i and i to
NSER-1
            subArray_Max = ((long)(i - NGEL[i]) * (NGER[i] - i)) ;
            ans = ((long)A[i] * (subArray_Max - subArray_Min)) ;
            sum = (sum + ans) % mod;
            sum = (sum + mod) % mod;// to change -ve to +ve
            // (-3+7)%7
            // 4%7
            // 4
        }
        return (int)sum;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Queues

Q2. Perfect Numbers

```
public class Solution {
    public String solve(int A) {
        Queue<String> q = new LinkedList<String>();
        if(A == 1){
            return "11";
        }
        if(A == 2){
            return "22";
        }
        q.offer("1");
        q.offer("2");
        String ans = "";
        int count=2;
        String a= "";
        String b= "";
        while(count < A){
            StringBuilder sb= new StringBuilder(q.remove());
            sb.append("1");
            q.offer(sb.toString());
            if(count+1 == A){
                ans= sb.toString();
            }
            sb.deleteCharAt(sb.length()-1);
            sb.append("2");
            q.offer(sb.toString());
            if(count+2 == A){
                ans= sb.toString();
            }
            count= count+2;
        }
        StringBuilder temp = new StringBuilder(ans);
        return ans+temp.reverse();
    }
}

// TC ->  O(A)
// SC -> O(A)
```

Q3. Parking Ice Cream Truck

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public ArrayList<Integer> slidingMaximum(final List<Integer> A, int B) {
        Deque<Integer> dq =new ArrayDeque<>();
        int N= A.size();
        ArrayList<Integer> ans = new ArrayList<Integer>();

        for(int i=0; i<B; i++){
            while(!dq.isEmpty() && dq.getLast()<A.get(i)){
                dq.removeLast();
            }
            dq.addLast(A.get(i));
        }
        ans.add(dq.getFirst());

        int s=1;
        int e=B;
        while(e < N){
            if(dq.getFirst() == A.get(s-1)){
                dq.removeFirst();
            }

            while(!dq.isEmpty() && dq.getLast()<A.get(e)){
                dq.removeLast();
            }
            dq.addLast(A.get(e));

            ans.add(dq.getFirst());
            s++;
            e++;
        }
    return ans;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Tree

Q1. Preorder Traversal

```
public class Solution {
    public ArrayList<Integer> preorderTraversal(TreeNode A) {
        Stack<TreeNode> st = new Stack<TreeNode>();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        if(A == null){
            return ans;
        }
        st.push(A);
        while(!st.empty()){
            TreeNode n = st.pop();
            ans.add(n.val);
            if(n.right != null){
                st.push(n.right);
            }
            if(n.left != null){
                st.push(n.left);
            }
        }
        return ans;
    }
}
```

Q2. Inorder Traversal

```
public class Solution {
    public ArrayList<Integer> inorderTraversal(TreeNode A) {
        Stack<TreeNode> st = new Stack<TreeNode>();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        TreeNode curr = A;
        while(curr != null || !st.empty()){
            if(curr != null){
                st.push(curr);
                curr = curr.left;
            }
            else{
                curr = st.pop();
                ans.add(curr.val);
                curr= curr.right;
            }
        }
        return ans;
    }
}

// public class Solution {
//     public ArrayList<Integer> inorderTraversal(TreeNode A) {
//         Stack<TreeNode> st = new Stack<TreeNode>();
//         ArrayList<Integer> ans = new ArrayList<Integer>();
//         TreeNode current = A;

//         while (current != null || !st.isEmpty()) {
//             // Reach the leftmost node of the current node
//             while (current != null) {
//                 st.push(current);
//                 current = current.left;
//             }
//             // Current must be null at this point, so pop the top of the stack
//             current = st.pop();
//             ans.add(current.val);
//             // We have visited the node and its left subtree, now visit the right subtree
//             current = current.right;
//         }
//         return ans;
//     }
// }

// TC -> O(N)
// SC -> O(N)
```

Q3. Binary Tree From Inorder And Postorder

```
// In this question what we are already given is the Inorder traversal and post-order
traversal of a tree
// now we are expected to create the tree from these 2 ArrayLists

// So, there are a few observations to make :
// 1) the inorder traversal is L N R (left node right)
// 2) the postorder traversal is L R N (left right node)
// 3) For example if we have a tree as shown below

//      1
//     / \
//    6   2
//   /   / \
//  7   3  4
// the inorder traversal will be like -> 7 6 1 3 2 4
// the postorder traversal will be like -> 7 6 3 4 2 1

// Now if we observe the sequence, we see that :
// 1) the root is the last element in our postorder traversal
// 2) if we see the inorder traversal, the element on the left of 1 are in the left sub tree
and to its right are in the right subtree

// So to create the tree we can use this specific observation by using recursion

public class Solution {
    HashMap<Integer, Integer> hm = new HashMap<>();

    public TreeNode buildTree(ArrayList<Integer> A, ArrayList<Integer> B) {
        for(int i=0; i<A.size(); i++){
            hm.put(A.get(i) , i);
        }
        return constructFromInorderPostorder(A, B, 0, A.size()-1, B.size()-1);

        }

        public TreeNode constructFromInorderPostorder(ArrayList<Integer> io,
ArrayList<Integer> po, int st_in, int en_in, int en_po ){
            if(st_in > en_in) return null;
            TreeNode root = new TreeNode(po.get(en_po));
            int root_in = hm.get(root.val);
            int cnt_left = root_in - st_in;
            int cnt_right = en_in - root_in;
            root.right = constructFromInorderPostorder(io, po, root_in +1, en_in, en_po-1);
            root.left = constructFromInorderPostorder(io, po,st_in, root_in -1, en_po-
cnt_right-1);
            return root;
    }
}
```

Q1. Level Order

```
public class Solution {
    public ArrayList<ArrayList<Integer>> solve(TreeNode A) {
        ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(A);
        while(q.size() > 0){
            int sz = q.size();
            ArrayList<Integer> temp = new ArrayList<Integer>();
            for(int i=0; i<sz; i++){
                TreeNode node = q.remove();
                temp.add(node.val);
                if(node.left != null){
                    q.offer(node.left);
                }
                if(node.right != null){
                    q.offer(node.right);
                }
            }
            ans.add(temp);
        }
        return ans;
    }
}


// TC -> O(N)
// Sc -> O(N)
```

Q2. Right View of Binary tree

```
public class Solution {
    public ArrayList<Integer> solve(TreeNode A) {
        ArrayList<Integer> rview = new ArrayList<Integer>();
        Queue<TreeNode> q = new LinkedList<>();
        q.offer(A);
        while(q.size() > 0){
            int sz=q.size();
            for(int i=0; i<sz; i++){
                TreeNode node = q.remove();
                if(i == sz-1){
                    rview.add(node.val);
                }
                if(node.left != null){
                    q.offer(node.left);
                }
                if(node.right != null){
                    q.offer(node.right);
                }
            }
        }
        return rview;
    }
}

// TC -> O(N)
// SC ->O(N)
```

Q3. Vertical Order traversal

```
class pair{
    int vi;
    TreeNode node;
    pair(int x, TreeNode y){
        vi=x;
        node=y;
    }
```

```java
}
public class Solution {
    public ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode A) {
        Queue<pair> q= new LinkedList<>();
        HashMap<Integer, ArrayList<Integer>> hm= new HashMap<Integer, ArrayList<Integer>>();
        int min =0;
        int max =0;
        // pair rpair = new pair(0,A);
        q.offer(new pair(0,A));
        while(q.size()>0){
            pair temp = q.remove();
            min = Math.min(min,temp.vi);
            max = Math.max(max,temp.vi);
            if(hm.containsKey(temp.vi)){
                hm.get(temp.vi).add(temp.node.val);
            }
            else{
                ArrayList<Integer> NewArr = new ArrayList<Integer>();
                NewArr.add(temp.node.val);
                hm.put(temp.vi, NewArr);
                // hm.put(temp.vi, new ArrayList<TreeNode>(temp.node));
            }
            if(temp.node.left  != null){
                q.offer(new pair(temp.vi-1 , temp.node.left));
            }
            if(temp.node.right != null){
                q.offer(new pair(temp.vi+1 , temp.node.right));
            }
        }
        ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();
        for(int i=min ; i<=max; i++){
            ArrayList<Integer> sub = new ArrayList<Integer>();
            for(int j=0; j<hm.get(i).size(); j++){
                sub.add(hm.get(i).get(j));
            }
            ans.add(sub);
        }
        return ans;
    }
}

// TC -> O(N)
// SC -> O(N*k) k represent constant
```

Q4. Balanced Binary Tree

```java
public class Solution {
    int ans = 1;
    public int isBalanced(TreeNode A) {
        Hight(A);
        return ans;
    }
    public int Hight(TreeNode A){
        if(A == null){
            return -1;
        }
        int l = Hight(A.left);
        int r = Hight(A.right);

        if( Math.abs(l-r) > 1){
            ans =0;
        }
        return Math.max(l, r)+1;
    }
}

// For the type of question where we have to calculate
// something that is dependent upon some other
// parameter then those questions can be sorted by
// a technique known as travel change
// TC -> O(N)
// Sc -> O(N)
```

Q1. Search in BST

```
public class Solution {
    public int solve(TreeNode A, int B) {
        if(A == null){
            return 0;
        }
        if(A.val == B){
            return 1;
        }
        else if(A.val > B){
            return solve(A.left, B);
        }
        else{
            return solve(A.right, B);
        }
        // return 0;
    }
}

// T.C. -> O(Hight of tree)
// worst O(N) - in case of skrewed tree
// S.C -> same
```

Q2. Delete a node in BST

```
public class Solution {
    public TreeNode solve(TreeNode A, int B) {
        if(A == null){
            return null;
        }
        if(A.val == B){
            if(A.left == null && A.right == null){
                return null;
            }
            else if(A.left == null){
                return A.right;
            }
            else if(A.right == null){
                return A.left;
            }
            else{
                TreeNode temp = A;
                temp= temp.left;
                while(temp.right != null){
                    temp = temp.right;
                }
                Swap(A,temp);
                A.left = solve(A.left, B);
            }
        }
        else if(A.val > B){
            A.left = solve(A.left, B);
        }
        else{
            A.right = solve(A.right, B);
        }
        return A;
    }
    public void Swap(TreeNode A, TreeNode temp){
        int t;
        t= A.val;
        A.val= temp.val;
        temp.val= t;
    }
}
```

Q3. Sorted Array To Balanced BST

```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public TreeNode sortedArrayToBST(final int[] A) {
        int s=0;
        int e=A.length-1;
        return Link(A,s,e);
    }
    public TreeNode Link(int[] A, int s, int e){
        if(s > e){
            return null;
        }
        int mid= s+ (e-s)/2;
        TreeNode root= new TreeNode(A[mid]);
        root.left = Link(A,s,mid-1);
        root.right = Link(A,mid+1,e);
        return root;
    }
}

// TC: O(N)
// SC: O(Height of BBST)
```

Q4. Valid Binary Search Tree

```
public class Solution {
    public int isValidBST(TreeNode A) {
        long min= Long.MIN_VALUE;
        long max= Long.MAX_VALUE;
        if(Ischeck(A, min, max)){
            return 1;
        }
        else{
            return 0;
        }
    }
    public boolean Ischeck(TreeNode A, long min, long max){
        if(A == null){
            return true;
        }
        if(min >= A.val || A.val >= max){
            return false;
        }
        boolean left = Ischeck(A.left, min, A.val);
        boolean right = Ischeck(A.right, A.val, max);
        return left && right;
    }
}


// TC -> O(N)
// SC -> O(N) or hight of tree

// in this we use preorder traversal
// but in lecture -> "Lecture | DSA: Trees 4: LCA + Morris Inorder Traversal" we use postorder
// please do with that also
```

Q1. Morris Inorder Traversal

```
 // Morris Inorder Traversal
public class Solution {
    public ArrayList<Integer> solve(TreeNode A) {
        ArrayList<Integer> ans = new ArrayList<Integer>();
        TreeNode curr = A;
        while(curr != null){
            if(curr.left == null){     //check left side null or not
                ans.add(curr.val);
                curr= curr.right;
            }
            else{
                TreeNode temp = curr.left;
                while(temp.right != null && temp.right != curr){   //for finding inorder 
predecesor
                    temp = temp.right;
                }
                if(temp.right == null){      //link created between inorder predecesor and 
curr node
                    temp.right = curr;
                    curr= curr.left;
                }
                else{                       //link deleted after got curr again and print curr
                    temp.right = null;
                    ans.add(curr.val);
                    curr= curr.right;
                }
            }
        }
        return ans;
    }
}

// TC -> O(N)
// SC -> O(1)
```

Q2. Kth Smallest Element In BST

```
public class Solution {

    public int kthsmallest(TreeNode A, int B) {
        int count =0;
        int kth = -1;
        TreeNode curr = A;
        while(curr != null){
            if(curr.left == null){     //check left side null or not
                count++;
                if(B == count){
                    kth= curr.val;
                    break;
                }
                curr= curr.right;
            }
            else{
                TreeNode temp = curr.left;
                while(temp.right != null && temp.right != curr){   //for finding inorder 
predecesor
                    temp = temp.right;
                }
                if(temp.right == null){      //link created between inorder predecesor and 
curr node
                    temp.right = curr;
                    curr= curr.left;
                }
                else{                       //link deleted after got curr again and print curr
                    temp.right = null;
                    count++;
                    if(B == count){
                        kth= curr.val;
                        break;
                    }
                    curr= curr.right;
                }
            }
        }
        return kth;
    }
}

// TC -> O(N)
// SC -> O(1)
```

Q3. Recover Binary Search Tree

```
public class Solution {
    public ArrayList<Integer> recoverTree(TreeNode A) {
        ArrayList<Integer> ans = new ArrayList<>();
        int first = Integer.MIN_VALUE, second = 0, logic = 1;
        TreeNode curr = A;
        while(curr != null) {
            if(curr.left == null) {
                if(logic == 1) {
                    if(curr.val > first) first = curr.val;
                    else{
                        logic = 0;
                        second = curr.val;
                    }
                }
                else if(curr.val < second) {
                    ans.add(curr.val);
                    ans.add(first);
                    return ans;
                }
                curr = curr.right;
            }
            else {
                TreeNode temp = curr.left;
                while(temp.right != null && temp.right != curr) {
                    temp = temp.right;
                }
                if(temp.right == null) {
                    temp.right = curr;
                    curr = curr.left;
                }
                else{
                    temp.right = null;
                    if(logic == 1) {
                        if(curr.val > first) first = curr.val;
                        else{
                            logic = 0;
                            second = curr.val;
                        }
                    }
                    else if(curr.val < second) {
                        ans.add(curr.val);
                        ans.add(first);
                        return ans;
                    }
                    curr = curr.right;
                }
            }
        }
        ans.add(second);
        ans.add(first);
        return ans;
    }
}
```

Q1. Path Sum

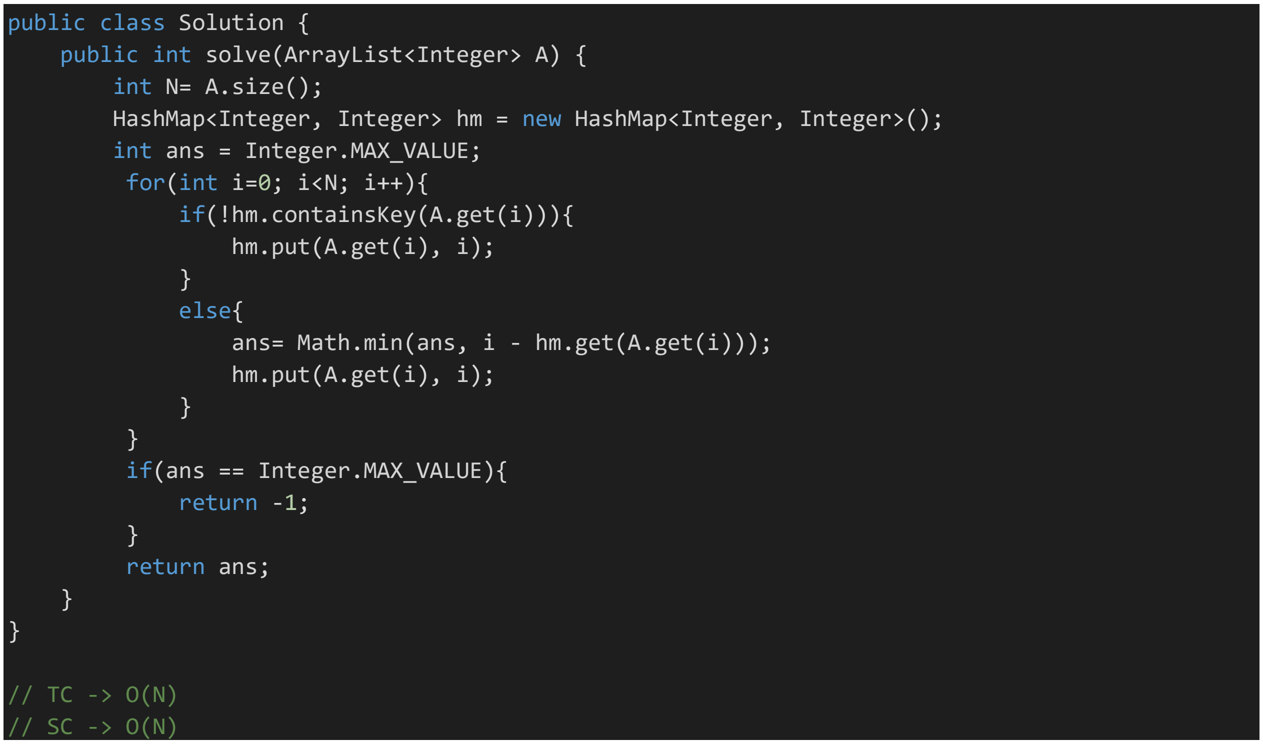
Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

```
public class Solution {
    public int hasPathSum(TreeNode A, int B) {
        if(A == null){
            return 0;
        }
        if(A.left == null && A.right == null){
            if(A.val == B){
                return 1;
            }
        }
        int left= hasPathSum(A.left, B - A.val);
        int right= hasPathSum(A.right, B - A.val);
        if(left == 1 || right == 1){
            return 1;
        }
        return 0;
        // return (hasPathSum(A.left, B - A.val) || hasPathSum(A.right, B - A.val)); this not
work because return type is int
    }
}
// TC -> O(N)
// SC -> O(N)
```

Q2. Next Pointer Binary Tree

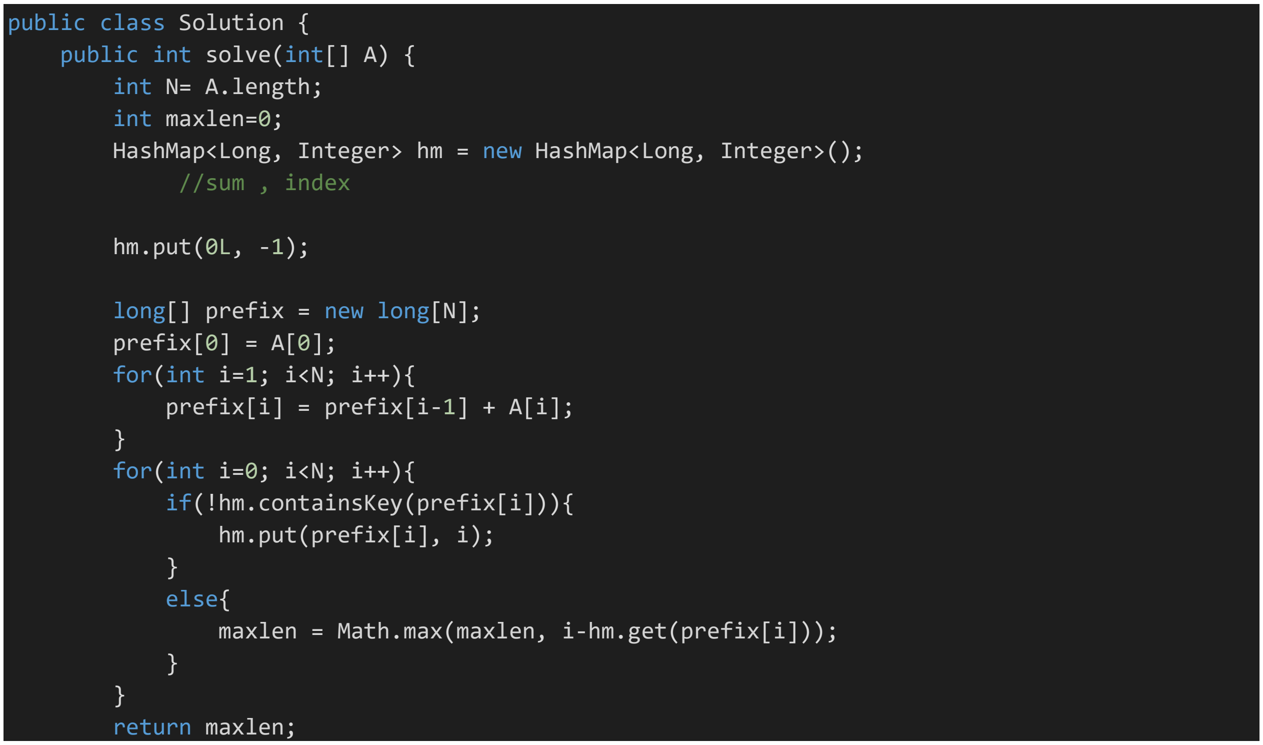
```
//without using space
public class Solution {
    public void connect(TreeLinkNode root) {
        TreeLinkNode curr = root;
        while(curr.left != null){
            TreeLinkNode temp = curr;
            while(temp != null){
                temp.left.next = temp.right;
                if(temp.next != null){
                    temp.right.next = temp.next.left;
                }
                temp= temp.next;
            }
            curr= curr.left;
        }
    }
}

// TC -> O(N)
// Sc -O(1)



//  with queue
// public class Solution {
//     public void connect(TreeLinkNode root) {
//         Queue<TreeLinkNode> q = new LinkedList<>();
//         q.offer(root);
//         while(q.size() != 0){
//             int sz = q.size();
//             for(int i=0; i<sz; i++){
//                 TreeLinkNode temp = q.remove();
//                 if(temp.left != null){
//                     q.offer(temp.left);
//                 }
//                 if(temp.right != null){
//                     q.offer(temp.right);
//                 }
//                 if(i < sz-1){
//                     temp.next = q.peek();
//                 }
//             }
//         }
//     }
// }

// TC -> O(N)
// SC ->O(N)
```

Q1. Shaggy and distances



```java
public class Solution {
    public int solve(ArrayList<Integer> A) {
        int N= A.size();
        HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
        int ans = Integer.MAX_VALUE;
         for(int i=0; i<N; i++){
             if(!hm.containsKey(A.get(i))){
                 hm.put(A.get(i), i);
             }
             else{
                 ans= Math.min(ans, i - hm.get(A.get(i)));
                 hm.put(A.get(i), i);
             }
         }
         if(ans == Integer.MAX_VALUE){
             return -1;
         }
         return ans;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q2. Longest Subarray Zero Sum



```java
public class Solution {
    public int solve(int[] A) {
        int N= A.length;
        int maxlen=0;
        HashMap<Long, Integer> hm = new HashMap<Long, Integer>();
             //sum , index

        hm.put(0L, -1);

        long[] prefix = new long[N];
        prefix[0] = A[0];
        for(int i=1; i<N; i++){
            prefix[i] = prefix[i-1] + A[i];
        }
        for(int i=0; i<N; i++){
            if(!hm.containsKey(prefix[i])){
                hm.put(prefix[i], i);
            }
            else{
                maxlen = Math.max(maxlen, i-hm.get(prefix[i]));
            }
        }
        return maxlen;
    }
}
```

Hashing

Q2. Count Subarray Zero Sum

```
public class Solution {
    public int solve(int[] A) {
        int N=A.length;
        int[] prefix = new int[N];
        HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
        prefix[0] = A[0];
        for(int i=1; i<N; i++){
            prefix[i] = prefix[i-1] + A[i];
        }
        int count =0;

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

        HashSet<Integer> hs=  new HashSet<Integer>();
        for(int i=0; i<N; i++){
            if(! hs.contains(prefix[i])){
                hs.add(prefix[i]);
                int freq= hm.get(prefix[i]);
                if(freq > 1){
                    count = count + (freq *(freq-1))/2;
                }
            }
        }
        return count;
    }
}


// TC -> O(N);
// SC -> O(N);
```

Q3. Common Elements

```java
public class Solution {
    public ArrayList<Integer> solve(ArrayList<Integer> A, ArrayList<Integer> B) {
        HashMap<Integer, Integer> hm= new HashMap<Integer, Integer>();
        ArrayList<Integer> ans= new ArrayList<Integer>();
        int N= A.size();
        int M= B.size();
        for(int i=0; i<N; i++){
            if(hm.containsKey(A.get(i))){
                hm.put(A.get(i), hm.get(A.get(i)) + 1);
            }
            else{
                hm.put(A.get(i), 1);
            }
        }
        for(int i=0; i<M; i++){
            if(hm.containsKey(B.get(i)) && hm.get(B.get(i))>0){
                ans.add(B.get(i));
                hm.put(B.get(i), hm.get(B.get(i))-1);
            }
        }
        return ans;
    }
}

// TC -> O(N+M);
// SC -> O(N);
```

Q1. Check Pair Sum

`i!=j`

```java
public class Solution {
    public int solve(int A, int[] B) {
        HashSet<Integer> hs = new HashSet<Integer>();
        int N= B.length;

        for(int i=0; i<N; i++){
            if(hs.contains(A-B[i])){
                return 1;
            }
            hs.add(B[i]);
        }
        return 0;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q2. Count Pair Difference

A[i] - A[j] = B and i ≠ j.

```
public class Solution {
    public int solve(int[] A, int B) {
        int N= A.length;
        HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
        int count=0;
        for(int i=0; i<N; i++){
            int target1= A[i]+ B;
            int target2= A[i]- B;
            if(hm.containsKey(target1)){
                count = count+ hm.get(target1);
            }
            if(hm.containsKey(target2)){
                count = count+ hm.get(target2);
            }
            if(hm.containsKey(A[i])){
                hm.put(A[i], hm.get(A[i])+1);
            }
            else{
                hm.put(A[i], 1);
            }
        }
        return count;
    }
}

// i !< j
// |A[i] - A[j]| = B

// TC -> O(N)
// SC -> O(N)
```

Q3. Subarray Sum Equals K

```
public class Solution {
    public int solve(int[] A, int B) {
        int N= A.length;
        int[] prefix = new int[N];
        prefix[0]= A[0];
        for(int i=1; i<N; i++){
            prefix[i] = prefix[i-1] + A[i];
        }
        HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
        int count=0;
        for(int i=N-1; i>=0; i--){
            if(prefix[i] == B){
                count++;
            }
            int target = B+prefix[i];  //for right to left
            // int target = prefix[i]-B;   //for left to right
            // sum = p[j] - p[i-1];
            // sum - p[j] = -p[i-1];
            // p[j] - sum = p[i-1];
            if(hm.containsKey(target)){
                count = count + hm.get(target);
            }
            if(hm.containsKey(prefix[i])){
                hm.put(prefix[i], hm.get(prefix[i]) + 1);
            }
            else{
                hm.put(prefix[i], 1);
            }
        }
        return count;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q4. Distinct Numbers in Window

```
public class Solution {
    public ArrayList<Integer> dNums(ArrayList<Integer> A, int B) {
        int N=A.size();
        ArrayList<Integer> ans= new ArrayList<Integer>();
        HashMap<Integer, Integer> hm = new HashMap<Integer, Integer>();
        if(N < B){
            ans.add(-1);
            return ans;
        }
        for(int i=0; i<B; i++){
            if(hm.containsKey(A.get(i))){
                hm.put(A.get(i), hm.get(A.get(i))+1);
            }
            else{
                hm.put(A.get(i), 1);
            }
        }
        ans.add(hm.size());
        int s=0;
        for(int e=B; e<N; e++){
             if(A.get(s) != A.get(e)){
                hm.put(A.get(s), hm.get(A.get(s))-1);
                if(hm.get(A.get(s)) == 0){
                    hm.remove(A.get(s));
                }
                if(hm.containsKey(A.get(e))){
                    hm.put(A.get(e), hm.get(A.get(e)) + 1);
                }
                else{
                    hm.put(A.get(e), 1);
                }
            }
            ans.add(hm.size());
            s++;
        }
        return ans;
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q1. Merge Two Sorted Arrays

```
//test
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int[] solve(final int[] A, final int[] B) {
        int N=A.length;
        int M=B.length;
        int i=0;
        int j=0;
        int ind=0;
        int[] ans= new int[N+M];
        while(i<N && j<M){
            if(A[i] < B[j]){
                ans[ind]= A[i];
                ind++;
                i++;
            }
            else{
                ans[ind]=B[j];
                ind++;
                j++;
            }
        }
        while(i<N){
            ans[ind]= A[i];
            ind++;
            i++;
        }
        while(j<M){
            ans[ind]=B[j];
            ind++;
            j++;
        }
        return ans;
    }
}

// TC -> O(N+M)
// SC -> O(N+M)
```

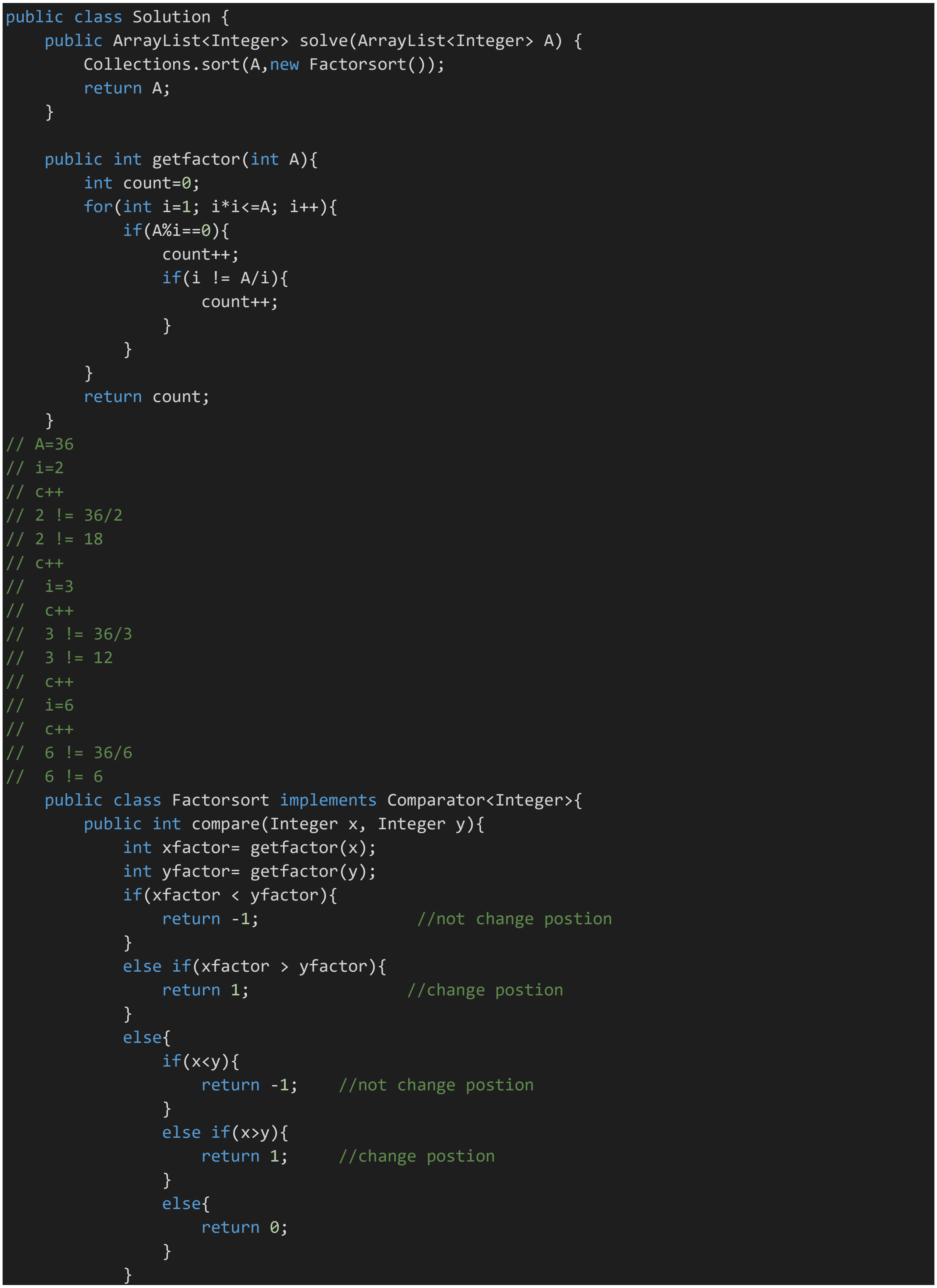
Q2. Inversion count in an array

```java
public class Solution {
    int mod=1000000007;
    int count;
    public void mergesort(int[] A, int s, int e){
        if(s>=e){
            return;
        }
        int mid=(s+e)/2;
        mergesort(A,s,mid);
        mergesort(A,mid+1,e);
        merge(A,s,mid,e);
    }
    public void merge(int[] A,int s,int mid,int e){
        int n1=mid-s+1;
        int n2=e-mid;
        int arr1[]=new int[n1];
        int arr2[]=new int[n2];
        int ind1=0;
        int ind2=0;
        for(int i=s; i<=mid;i++){
            arr1[ind1]=A[i];
            ind1++;
        }
        for(int i=mid+1; i<=e; i++){
            arr2[ind2]=A[i];
            ind2++;
        }

        int ind=s;
        int i=0;
        int j=0;

        while(i<n1 && j<n2){
            if(arr1[i] <= arr2[j]){
                A[ind]=arr1[i];
                ind++;
                i++;
            }
            else{
                A[ind]=arr2[j];
                ind++;
                j++;
                count= (count + n1 - i) % mod;  //distsnce from i to n1(n1 - 1 - i + 1)
            }
        }
        if(i<n1){
            while(i<n1){
                A[ind]=arr1[i];
                ind++;
                i++;
            }
        }
        else{
            while(j<n2){
                A[ind]=arr2[j];
                ind++;
                j++;
            }
        }
    }
    public int solve(int[] A) {
        int s=0;
        int e=A.length-1;
        count=0;
        mergesort(A,s,e);
        return count;
    }
}

// TC - O(NlongN)
// SC - O(N)
```

Q3. Count Sort

```
public class Solution {
    public int[] solve(int[] A) {
        int N= A.length;
        int max=Integer.MIN_VALUE;
        for(int i=0; i<N; i++){
            if(max < A[i]){
                max = A[i];
            }
        }

        int[] freq = new int[max+1];
        for(int i=0; i<N; i++){
            freq[A[i]]++;
        }

        int ind =0;
        for(int i=0; i<max+1; i++){
            for(int j=0; j<freq[i]; j++){
                A[ind]= i;
                ind++;
            }
        }
        return A;
    }
}

// TC -> O(N)
// SC -> O(1) as we using static(fix) space
```

Q1. Factors sort



```
public class Solution {
    public ArrayList<Integer> solve(ArrayList<Integer> A) {
        Collections.sort(A,new Factorsort());
        return A;
    }

    public int getfactor(int A){
        int count=0;
        for(int i=1; i*i<=A; i++){
            if(A%i==0){
                count++;
                if(i != A/i){
                    count++;
                }
            }
        }
        return count;
    }
// A=36
// i=2
// c++
// 2 != 36/2
// 2 != 18
// c++
//  i=3
//  c++
//  3 != 36/3
//  3 != 12
//  c++
//  i=6
//  c++
//  6 != 36/6
//  6 != 6
    public class Factorsort implements Comparator<Integer>{
        public int compare(Integer x, Integer y){
            int xfactor= getfactor(x);
            int yfactor= getfactor(y);
            if(xfactor < yfactor){
                return -1;                //not change postion
            }
            else if(xfactor > yfactor){
                return 1;                //change postion
            }
            else{
                if(x<y){
                    return -1;    //not change postion
                }
                else if(x>y){
                    return 1;     //change postion
                }
                else{
                    return 0;
                }
            }
        }
    }
}

// TC -> O(NlogN * root A[i])
```

Q2. Largest Number

```
public class Solution {
    public String largestNumber(ArrayList<Integer> A) {
        Collections.sort(A, new LargeNumSort());
        int N=A.size();

        StringBuilder sb = new StringBuilder();     //StringBuilder will take less time than
nor sting var(evertime
        for (int i = 0; i < N; i++) {               //it will create new "string var" we add
continusly)
            sb.append(Integer.toString(A.get(i)));
        }
        if(A.get(0) == 0){
            return "0";
        }
        return sb.toString();
    }

    public class LargeNumSort implements Comparator<Integer>{
        public int compare(Integer x, Integer y){
            String a = String.valueOf(x) + String.valueOf(y);
            String b = String.valueOf(y) + String.valueOf(x);
            // int P= Integer.valueOf(a);   //integer out of bound check Constraints (worst
case (2*10^9) * (2*10^9))
            // int Q= Integer.valueOf(b);
            return -a.compareTo(b);   //a.compareTo(b) will give -1 if a<b but we want to
change so used -a.compareTo(b)
            // if(P < Q){          //yx that is Q is greter then replace the position
            //     return 1;
            // }
            // else if(P > Q){     //xy that is P is greter then  x and y postion should be
same
            //     return -1;
            // }
            // else{
            //     return 0;
            // }
        }
    }
}


// N * logN * M
// M= addition of legnth of 2 String
// N= number of sting
// TC -> O(NlogN * M)
// M= addition of legnth of 2 String
// N= number of sting

// SC -> O(N*M)   N*M*2
// N= size of ArrayList
// M= Max length of any string
```

Q3. B Closest Points to Origin

```
public class Solution {
    public int[][] solve(int[][] A, int B) {
    Arrays.sort(A,new Distance());
    int[][] ans=new int[B][2];
     for(int i=0;i<B;i++){
        ans[i]=A[i];
     }
     return ans;
    }

    public class Distance implements Comparator<int[]>{
        public int compare(int[] a, int[] b){
            int da=a[0]*a[0] + a[1]*a[1];
            int db=b[0]*b[0] + b[1]*b[1];
            if(da == db){
                return a[0]-b[0];
            }else{
                return da-db;
            }
        }
    }
}

// TC -> O(NlogN)
// SC -> O(N)
```

Binary Search

Q1. Sorted Insert Position

```
public class Solution {
    public int searchInsert(int[] A, int B) {
        int s=0;
        int e=A.length - 1;
        while(s<=e){
            int mid = s+(e-s)/2;
            if(A[mid] == B){
                return mid;
            }
            else if(A[mid] > B){
                e=mid-1;
                // ind=mid;
            }
            else{      //A[mid] < B
                s=mid+1;
            }
        }
        return s;
    }
}

// TC -> O(log(N)base 2)
```

Q2. Search for a Range

```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int[] searchRange(final int[] A, int B) {
        int N=A.length;
        int s=0;
        int e=N - 1;
        int[] ans = {-1,-1};
        while(s<=e){
            int mid = s + (e-s)/2;
            if(A[mid] == B){
                // if(mid == 0){
                //     ans[0]=mid;
                //     break;
                // }
                if((mid == 0) || (A[mid-1] != A[mid])){
                    ans[0]=mid;
                    break;
                }
                else{
                    e=mid-1;
                }
            }
            else if(A[mid] < B){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }

        s=0;
        e=N - 1;
        while(s<=e){
            int mid = s + (e-s)/2;
            if(A[mid] == B){
                // if(mid == N-1){
                //     ans[1]=mid;
                //     break;
                // }
                if((mid == N-1) || (A[mid+1] != A[mid])){
                    ans[1]=mid;
                    break;
                }
                else{
                    s=mid+1;
                }
            }
            else if(A[mid] < B){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        return ans;
    }
}
// TC -> O(log(N))
// SC -> O(1)
```

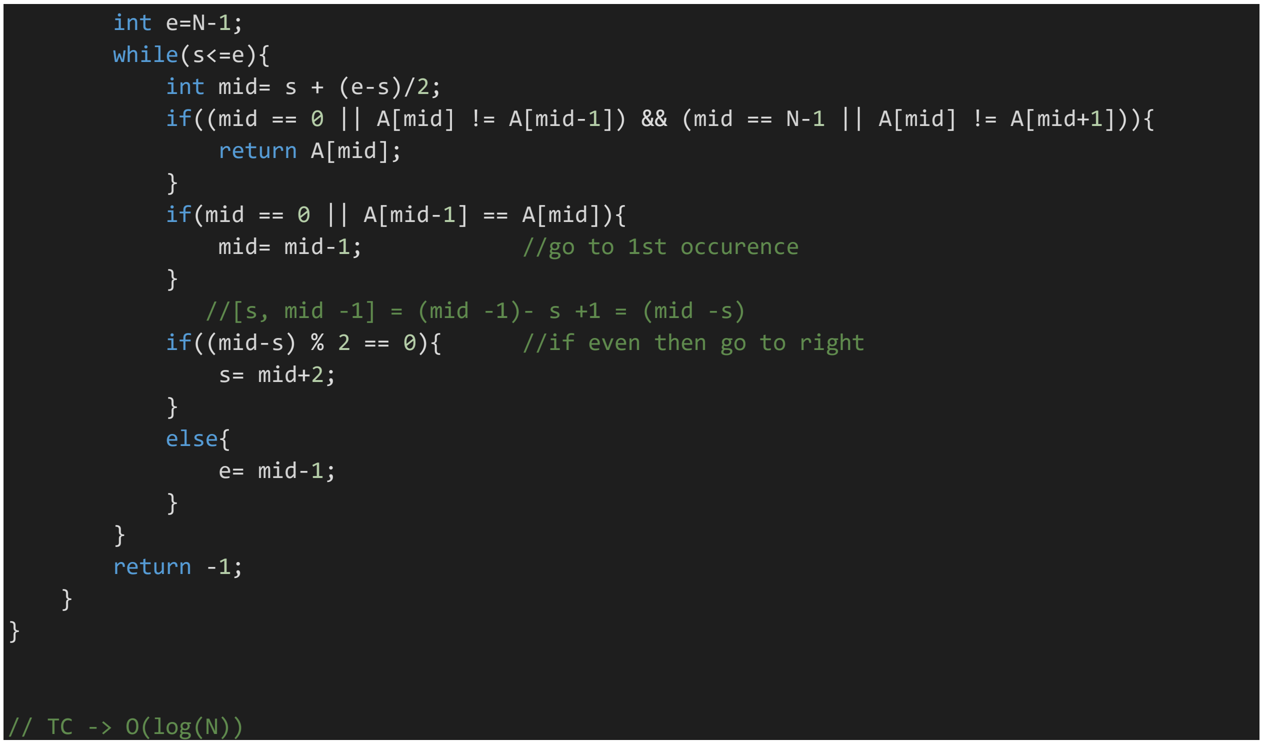
Q3. Find a peak element

```
public class Solution {
    public int solve(int[] A) {
        int N=A.length;
        int s=0;
        int e=N-1;
        while(s<=e){
            int mid= s+(e-s)/2;
            if((mid==0 || A[mid-1]<=A[mid]) && (mid==N-1 || A[mid] >= A[mid+1])){
                return A[mid];
            }
            else if(A[mid] < A[mid+1]){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        return -1;
    }
}
```

Q4. Single Element in Sorted Array

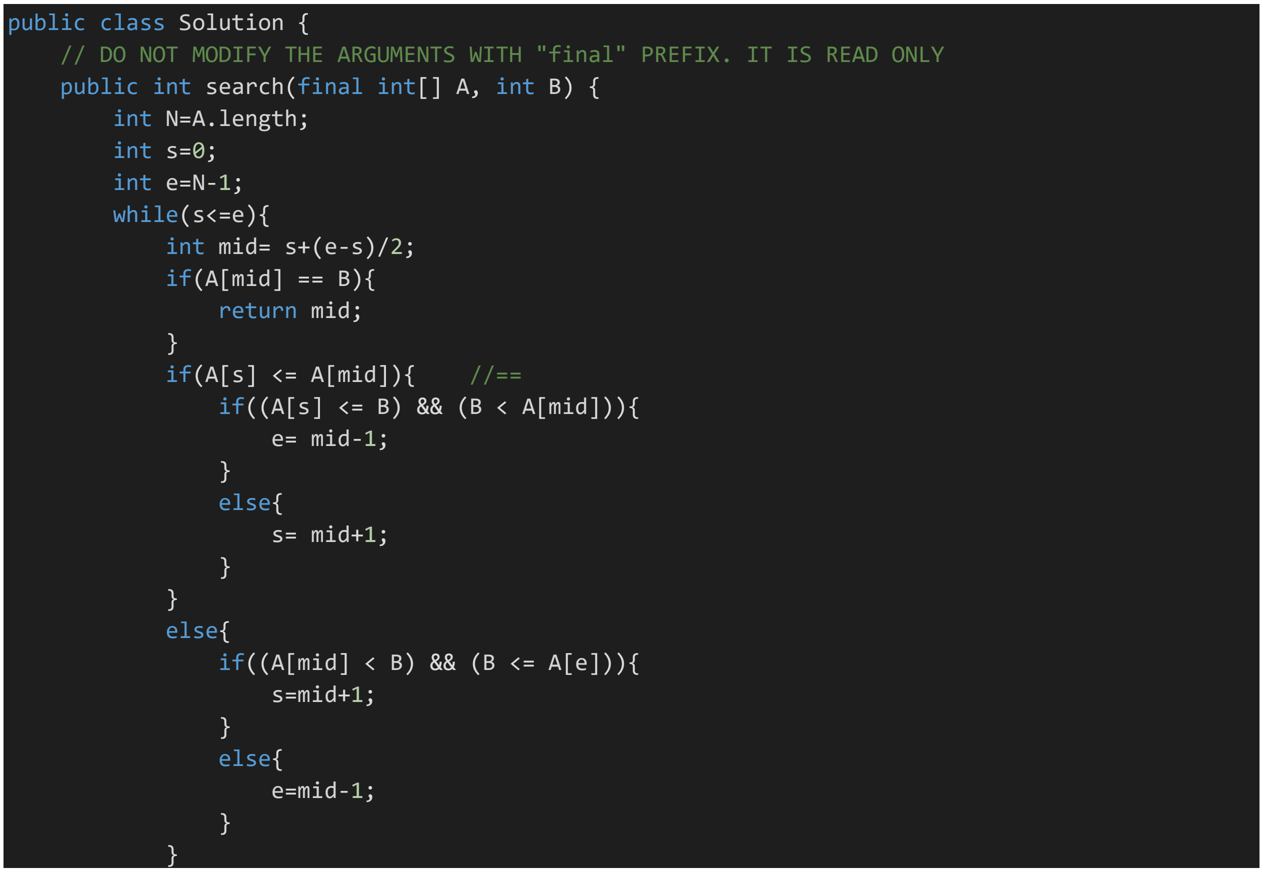
Odd even

```
public class Solution {
    public int solve(int[] A) {
        int N= A.length;
        int s=0;
```

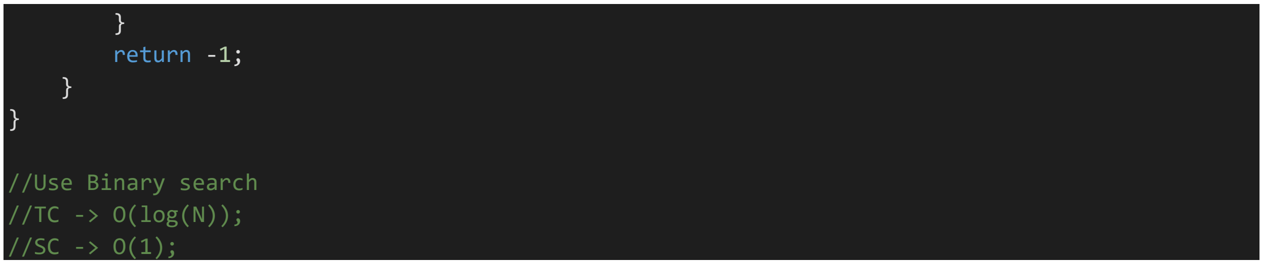


```
        int e=N-1;
        while(s<=e){
            int mid= s + (e-s)/2;
            if((mid == 0 || A[mid] != A[mid-1]) && (mid == N-1 || A[mid] != A[mid+1])){
                return A[mid];
            }
            if(mid == 0 || A[mid-1] == A[mid]){
                mid= mid-1;            //go to 1st occurence
            }
               //[s, mid -1] = (mid -1)- s +1 = (mid -s)
            if((mid-s) % 2 == 0){      //if even then go to right
                s= mid+2;
            }
            else{
                e= mid-1;
            }
        }
        return -1;
    }
}


// TC -> O(log(N))
```

Q1. Rotated Sorted Array Search



```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int search(final int[] A, int B) {
        int N=A.length;
        int s=0;
        int e=N-1;
        while(s<=e){
            int mid= s+(e-s)/2;
            if(A[mid] == B){
                return mid;
            }
            if(A[s] <= A[mid]){    //==
                if((A[s] <= B) && (B < A[mid])){
                    e= mid-1;
                }
                else{
                    s= mid+1;
                }
            }
            else{
                if((A[mid] < B) && (B <= A[e])){
                    s=mid+1;
                }
                else{
                    e=mid-1;
                }
            }
```



```
        }
        return -1;
    }
}

//Use Binary search
//TC -> O(log(N));
//SC -> O(1);
```

Q2. Median of Array

```
public class Solution {
    // DO NOT MODIFY BOTH THE LISTS
    public double findMedianSortedArrays(final List<Integer> a, final List<Integer> b) {
        int m = a.size();
        int n = b.size();
        if(m > n){
            return findMedianSortedArrays(b, a);
        }
        int start = 0;
        int end = m;
        int medianPos = ((m + n) + 1)/2;
        while(start <= end){
            int cut1 = (start + end) /2;
            int cut2 = medianPos - cut1;
            int l1 = (cut1 == 0) ? Integer.MIN_VALUE : a.get(cut1 - 1);
            int l2 = (cut2 == 0) ? Integer.MAX_VALUE : b.get(cut2 - 1);
            int r1 = (cut1 == m) ? Integer.MAX_VALUE : a.get(cut1);
            int r2 = (cut2 == n) ? Integer.MIN_VALUE : b.get(cut2);
            if(l1 <= r2 && l2 <= r1){
                if((m + n) % 2 != 0){
                    return Math.max(l1,l2);
                }
                else{
                    return (Math.max(l1, l2) + Math.min(r1, r2))/2.0;
                }
            }
            else if(l1 > r2){
                end = cut1 - 1;
            }
            else{
                start = cut1 + 1;
            }
        }
        return 0.0;
    }
}
```

Q3. Ath Magical Number

Divisible by B and C

```
//By Sir Method
public class Solution {
    public int solve(int A, int B, int C) {
        long min=Math.min(B,C);
        long s=1;
        long e=min*A;
        long lcm=LCM(B,C);    //Divided_By_Both_Number B nad C
        long mod = 1000000007;
        while(s <= e){
            long mid=s + (e-s)/2;
            long temp=(mid/B) + (mid/C) - (mid/lcm);
            if(temp == A){
                if(mid%B==0 || mid%C==0){
                    return (int)(mid%mod);
                }
                else{
                    e=mid-1;
                }
            }
            else if(temp > A){
                e= mid-1;
            }
            else{
                s= mid+1;
            }
        }
        return -1;
    }
    public int LCM(int b, int c){
        return ((b/GCD(b,c))*c);
    }
    public int GCD(int b, int c){
        if(b==0){
            return c;
        }
        return GCD(c%b , b);
    }
}

// T(min(B,C)*A) + T(log(min(B,C)))

// TC -> O(log(min(B,C)*A))
```

Q4. Square Root of Integer

```
public class Solution {
    public int sqrt(int A) {
        long s=1;
        long e=A;
        if(A==0){
            return 0;
        }
        while(s<=e){
            long mid = s + (e-s)/2;
            if((mid*mid) == A){
                return (int)mid;
            }
            else if((mid*mid) > A){
                e=mid-1;
            }
            else{      //(mid*mid) < A
                if(((mid+1)*(mid+1)) > A){
                    return (int)mid;
                }
                else{
                    s=mid+1;
                }
            }
        }
        return -1;
    }
}

// TC -> O(log(N))
// SC -> O(1)
```

Q1. Aggressive cows

```
//From Discussion
public class Solution {                                 // TC=O(n*log(search
space))     SC=O(1)
    public int with_mid(int[] A, int mid){   // TC=O(n)
        int n = A.length;
        int d = mid; // min_d : dist b/w 2 nearest stalls
        int c=1;    // at A[0]   // c=1 bcz we place 1st cow b4hand at 1st posn/1st stall i..
A[0];
        for(int i=1;i<n;i++ ){
            int prev_pos = A[i-1];
            int curr_pos = A[i];
            if( (prev_pos + d) < curr_pos){
                d = 0;
                c++;
                d = mid;
            }
            else if( (prev_pos + d) == curr_pos ) {
                d = 0;      // only written to show that we have reached next positon/ next
stall
                c++;        // now we will place our cow here
                d = mid;
            }
            else if( (prev_pos + d) > curr_pos) {
                d = prev_pos + d- curr_pos; // or d=d-(curr-prev)
            }
        }
        return c;
    }
    public int solve(int[] A, int B) { // TC=O(log(search-space)) = O(log(Range)) -- Range-
[min_d or 1, max_d]
        Arrays.sort(A);     // Stalls positions given are jumbled-
        int n=A.length;
        int min_d = Integer.MAX_VALUE;
        int max_d = A[n-1]-A[0];  // max_distance can be only when 2 cows are placed at stalls
at 2 ends
        int ans=0;

    // Getting mininum distance (i.e. distance b/w 2 nearest stalls )
         for(int i=0;i<n-1;i++){
            min_d=Math.min(A[i+1]-A[i], min_d);
        }

        int l=min_d;        // min_d= distance b/w 2 nearest stalls (in given posns of stalls)
        int h=max_d;        // max_d= distance b/w 2 farthest stalls (in given posns of
stalls)
        while(l<=h){
            int mid=(l+h)/2;
            int c=with_mid(A, mid);
            if(c==B){
                ans=mid;
                l=mid+1;
            }
            else if(c<B){
                h=mid-1;
            }
            else if(c>B){
                ans=mid;
                l=mid+1;
            }
        }
        return ans;
    }
}
```

Q2. Painter's Partition Problem

```
public class Solution {
    public int paint(int A, int B, int[] C) {
        int N= C.length;
        long s= Integer.MIN_VALUE; //s denotes min length of answer space which is max value
in C array
        long e=0;                //e denotes max length of answer space which we wil find by
adding every element
        for(int i=0; i<N; i++){
            if(C[i] > s){
                s=C[i];
            }
            e += C[i];
        }
        s=s*B;
        e=e*B;
        long mod= 10000003;
        long ans= -1;
        while(s<=e){
            long mid= s+(e-s)/2;

            // System.out.println(check(mid-1, A, B, C));
            if(check(mid, A, B, C)){
                ans =mid;             //we have remove 1 check funnction to obtimize
                e=mid-1;
            }
            else{
                s=mid+1;
            }
        }
        return (int)(ans%mod);
    }

    public boolean check(long t, int a, int b, int[] C){
        long curr_time=t;
        long count=1;
        int N=C.length;

        for(int i=0; i<N; i++){
            if((long)C[i]*b <= curr_time){
                curr_time -= (long)C[i]*1L*b;
            }
            else{
                count++;
                curr_time= t - (long)(C[i]*1L*b);
            }
            if(count > a){
                return false;
            }
        }
        return true;
    }
}
```

Backtracking

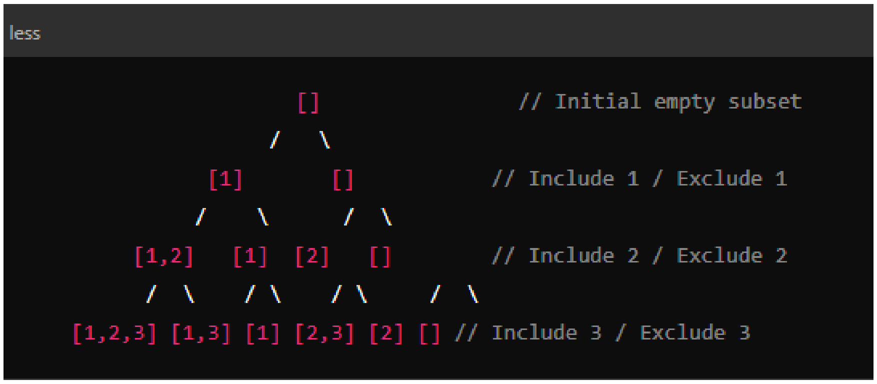
Q1. Generate all Parentheses II

```
public class Solution {

    ArrayList<String> result = new ArrayList<>();

    public ArrayList<String> generateParenthesis(int A) {
        findValidParentheses(A, 0, 0, result, "");
        return result;
    }

    public void findValidParentheses(int limit, int o, int c, ArrayList<String> result, String str) {

        if(o > limit || c > limit || c > o) { // also return if closing is greater than opening return
            return;
        }

        if(o==c && c==limit) {
            result.add(str);
        }

        findValidParentheses(limit, o+1, c, result, str+"(");
        findValidParentheses(limit, o, c+1, result, str+")");
    }

}
```

Q2. Permutations

```
public class Solution {
    public ArrayList<ArrayList<Integer>> permute(ArrayList<Integer> A) {
        int N=A.size();
        ArrayList<Integer> temp = new ArrayList<Integer>();
        int idx = 0;
        for(int i=0;i<N;i++)
        {
            temp.add(i,-1);
        }
        int[] visited = new int[N];  //we can also make arraylist and delare all element with "false" but array look more easy
        ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();
        Permutation(A,idx,temp,visited,ans);
        return ans;
    }

    public static void Permutation(ArrayList<Integer> arr, int idx, ArrayList<Integer> temp, int[] visited, ArrayList<ArrayList<Integer>> ans){
        if(idx == arr.size()){
            ans.add(new ArrayList(temp));
            return;
        }
        //All posibility
        for(int i=0; i<arr.size(); i++){
            if(visited[i] == 0){
                visited[i] = 1;
                temp.set(idx, arr.get(i));
                Permutation(arr, idx+1, temp, visited, ans);
                visited[i] = 0;
            }
        }
    }
}

// T.C. -> >=N! (greater than = to N!)
// S.C. -> O(N)
```

Q3. Subset



less

```
                      []                // Initial empty subset
                    /   \
               [1]        []           // Include 1 / Exclude 1
              /    \      /  \
          [1,2]   [1]  [2]   []        // Include 2 / Exclude 2
           /  \    / \    / \    /  \
     [1,2,3] [1,3] [1] [2,3] [2] [] // Include 3 / Exclude 3
```

```
public class Solution {

    ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

    public ArrayList<ArrayList<Integer>> subsets(ArrayList<Integer> A) {
        Collections.sort(A);                  //  non-descending order.
        int idx=0;
        ArrayList<Integer> temp = new ArrayList<Integer>();
        res.add(new ArrayList<>());
        Helper(A, idx, temp);
        return res;
    }

    public void Helper(ArrayList<Integer> arr, int idx, ArrayList<Integer> temp){
        if(idx == arr.size()){
            return;
        }

        temp.add(arr.get(idx));
        res.add(new ArrayList<>(temp));
        Helper(arr, idx+1, temp);
        temp.remove(temp.size()-1);
        Helper(arr, idx+1, temp);

    }
}


// TC -> >=O(2^N)
// SC -> (N) - depends on hight of tree
```

2 pointer

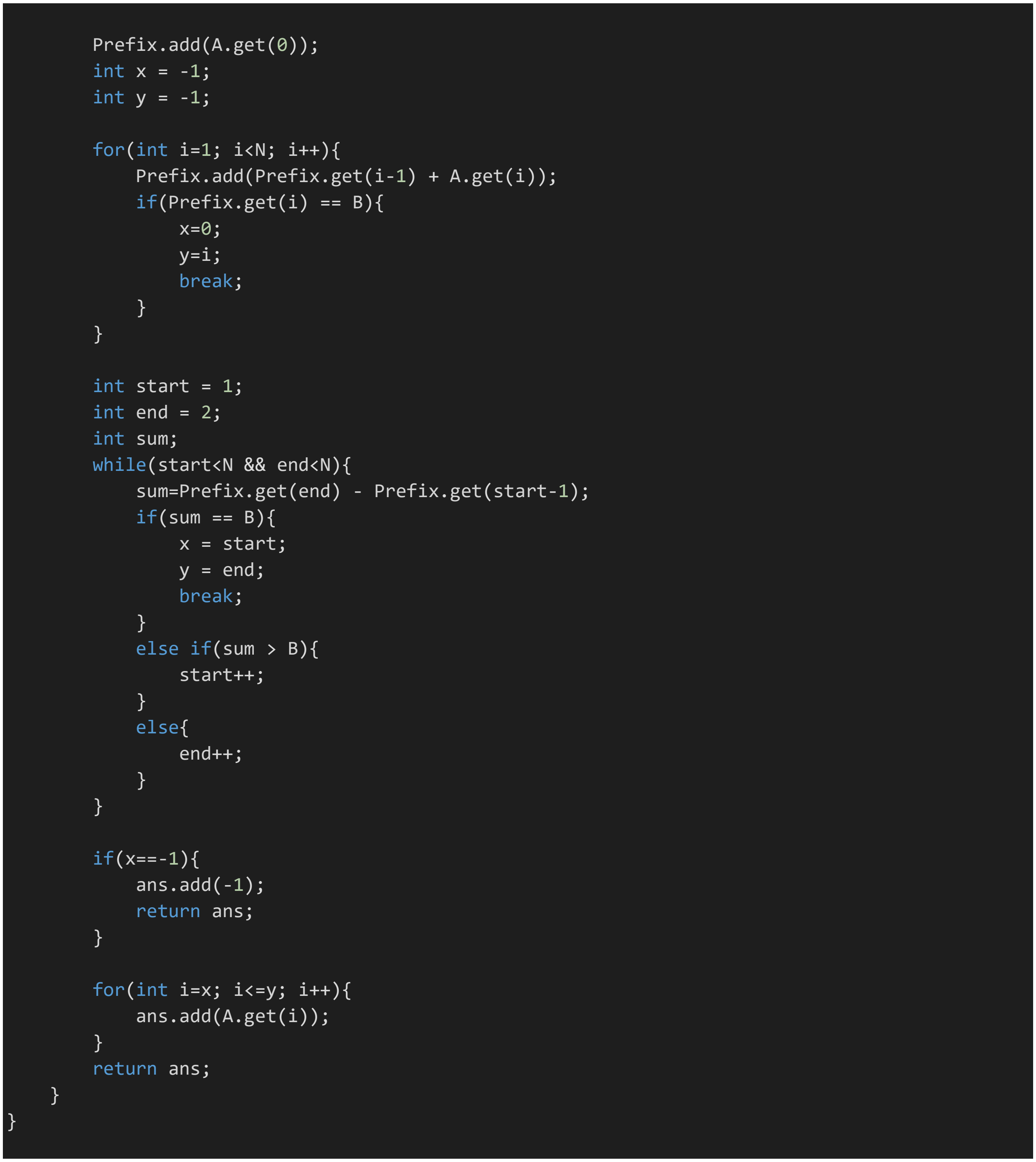
Q1. Container With Most Water

```
public class Solution {
    public int maxArea(int[] A) {
        int N=A.length;
        int i=0;
        int j=N-1;
        int max= Integer.MIN_VALUE;
        int sum=0;
        if(N == 1){
            return 0;
        }
        while(i<j){
            sum=(j-i) * Math.min(A[i],A[j]);
            max=Math.max(max,sum);
            if(A[i] < A[j]){
                i++;
            }
            else if(A[j] < A[i]){
                j--;
            }
            else{
                i++;
                j--;
            }
        }
        return max;
    }
}

// TC -> O(N)
// SC -> O(1)
```

Q2. Subarray with given sum

Given an array of positive integers **A** and an integer **B**, find and return first continuous subarray which adds to **B**.
If the answer does not exist **return an array** with a single integer "**-1**".
First sub-array means the sub-array for which starting index in minimum.

```
// this 2 pointer approach works only for +ve elemnet
// if array contains -ve as well then we need to use hashing approach check A[i] constraints
// TC -> O(N)
// SC -> O(N)

public class Solution {
    public ArrayList<Integer> solve(ArrayList<Integer> A, int B) {
        int N=A.size();
        ArrayList<Integer> ans = new ArrayList<Integer>();
        ArrayList<Integer> Prefix = new ArrayList<Integer>();
```



```
        Prefix.add(A.get(0));
        int x = -1;
        int y = -1;

        for(int i=1; i<N; i++){
            Prefix.add(Prefix.get(i-1) + A.get(i));
            if(Prefix.get(i) == B){
                x=0;
                y=i;
                break;
            }
        }

        int start = 1;
        int end = 2;
        int sum;
        while(start<N && end<N){
            sum=Prefix.get(end) - Prefix.get(start-1);
            if(sum == B){
                x = start;
                y = end;
                break;
            }
            else if(sum > B){
                start++;
            }
            else{
                end++;
            }
        }

        if(x==-1){
            ans.add(-1);
            return ans;
        }

        for(int i=x; i<=y; i++){
            ans.add(A.get(i));
        }
        return ans;
    }
}
```

Q3. Pairs with Given Difference

Given an one-dimensional integer array **A** of size **N** and an integer **B**.
Count all distinct pairs with difference equal to B.
Here a pair is defined as an integer pair (x, y), where x and y are both numbers in the array and their absolute difference is **B**.

```
//Discussion
public class Solution {
    public int solve(int[] A, int B) {
        Arrays.sort(A);//sort  the array to use two pointers approach
        int p1=0;
        int p2=1;
        int ans=0;
        while(p2<A.length){
            if(p1>0 && A[p1]==A[p1-1] && A[p2]==A[p2-1]){//edge case -> A[]=[1, 1, 1, 2, 2]
B=0 it will skip the duplicate elements
                p1++;
                p2++;
                continue;//continue->skip the current iteration
            }
            if(A[p2]-A[p1]==B){//if the B is found increase both p1 and p2 to get next pair
                ans++;// count it & do both p1 and p2 increment
                p1++;
                p2++;
            }else if(A[p2]-A[p1]>B){//if p1-p2 is greater than B then increase the p1
                p1++;
                if(p1==p2){//in both case p1 and p2 is equal increase the p2
                    p2++;
                }
            }else{//if p1-p2 is less than B increase the p2
                p2++;
            }
        }
        return ans;
    }
}
```

Q4. Pairs with given sum II

Given a sorted array of integers (not necessarily distinct) **A** and an integer **B**, find and return how many pair of integers (A[i], A[j]) such that i != j have sum equal to B. Since the number of such pairs can be very large, return number of such pairs modulo (10^9 + 7).

```
//Discussion
public class Solution {
    public int solve(int[] A, int B) {
        int n=A.length;
        int  i=0;
        int j=n-1;
        long ans=0;
        long mod=1000000007;
        while(i<j){
           int sum=A[i]+A[j];
           if(sum>B){
               j--;
           }
           else if(sum<B){
               i++;
           }
           else{
               int x=i;
               int y=j;

               if(A[i]==A[j]){
                   long count=(j-i+1);
                   ans=ans+count*(count-1)/2;
                   break;
               }
               else{
                   int count1=0;
                   while(A[x]==A[i]){
                       count1++;
                       x++;
                   }
                    // int count1=x-i;
                   i=x;
                  int count2=0;
                   while(A[j]==A[y]){
                       count2++;
                       y--;
                   }
                    //  int count2=j-y;
                   j=y;

                   ans=ans+count1*count2;
               }
           }
        }
        ans=ans%mod;
     return (int)(ans);

    }
}
```

Arrays

Q1. Max Sum Contiguous Subarray

Kadane’s Algo

```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int maxSubArray(final int[] A) {
        int N= A.length;
        // int ans=0;
        int cursum=0;
        int maxsum=Integer.MIN_VALUE;
        // or int maxsum=A[0];

        for(int i=0; i<N; i++){
            cursum= cursum + A[i];
            maxsum=Math.max(maxsum, cursum);
            if(cursum<0){
                cursum=0;
            }
        }
        return maxsum;
    }
}

//O(N) , O(1)
```

Q2. Continuous Sum Query

```
public class Solution {
    public int[] solve(int A, int[][] B) {
        int arr[]= new int[A];
        for(int i=0; i<B.length; i++){
            int l=B[i][0] -1;
            int r=B[i][1] -1;
            int p=B[i][2];

            arr[l]=arr[l] +p;
            if(r+1 < A){
                arr[r+1]=arr[r+1] -p;
            }
        }
        int sum=arr[0];
        for(int i=1; i<A; i++){
            arr[i] = arr[i-1] +arr[i];
        }
        return arr;
    }
}

// TC -> O(A)
// SC -> O(A)
```

Q3. Rain Water Trapped

```
//2 pointer approach
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int trap(final int[] A) {
        int N=A.length;
        int i=0;
        int j=N-1;
        int ans = 0;
        int water = 0;
        int lmax = A[0];
        int rmax = A[N-1];
        while(i<j){
            if(lmax < rmax){
                i++;
                water =lmax - A[i];
                lmax= Math.max(lmax, A[i]);
            }
            else{
                j--;
                water =rmax - A[j];
                rmax= Math.max(rmax, A[j]);
            }
            if(water > 0){
                ans = ans+ water;
            }
        }
        return ans;
    }
}
```

Q1. Search in a row wise and column wise sorted matrix

```
public class Solution {
    public int solve(int[][] A, int B) {
        int N=A.length;
        int M=A[0].length;
        int i=0;
        int j=M-1;
        int min=Integer.MAX_VALUE;

        while(i<N && j>=0){
            if(A[i][j] == B){
                min= Math.min(min, ((i+1) * 1009 + (j+1)));
                j--;
            }
            else if(A[i][j] < B){
                i++;
            }
            else if(A[i][j] > B){
                j--;
            }
        }
        if(min < Integer.MAX_VALUE){
            return min;
        }
        else{
            return -1;
        }
    }
}

// TC -> O(N+M)
// SC -> O(1)
```

Q2. Spiral Order Matrix II

```
public class Solution {
    public int[][] generateMatrix(int A) {
        int[][] ans= new int[A][A];
        int count =1;
        int i=0;
        int j=0;
        int round= A;

        while(round >= 2){
            // int i=x;
            // int j=y;
            for(int a=0; a<round-1; a++){
                ans[i][j++] = count++;
            }
            for(int a=0; a<round-1; a++){
                ans[i++][j] = count++;
            }
            for(int a=0; a<round-1; a++){
                ans[i][j--] = count++;
            }
            for(int a=0; a<round-1; a++){
                ans[i--][j] = count++;
            }
            round = round-2;
            i++;
            j++;
        }
        if(round == 1){
            ans[i][j]= count;
        }
        return ans;
    }
}

// TC -> O(4N) -> O(N)
// SC -> O(1)
```

Q3. Row to Column Zero

```
public class Solution {
    public int[][] solve(int[][] A) {
        int N=A.length;
        int M=A[0].length;
        for(int i=0; i<N; i++){             //for row wise check and update
            int flag =0;
            for(int j=0; j<M; j++){
                if(A[i][j] == 0){
                    flag =1;
                }
            }
            if(flag == 1){
                for(int j=0; j<M; j++){
                    if(A[i][j] != 0){
                        A[i][j]=-1;
                    }
                }
            }
        }

        for(int j=0; j<M; j++){                      //for coloum wise check and update
            int flag=0;
            for(int i=0; i<N; i++){
                if(A[i][j] == 0){
                    flag=1;
                }
            }
            if(flag == 1){
                for(int i=0; i<N; i++){
                    if(A[i][j] != 0){
                        A[i][j] = -1;
                    }
                }
            }
        }

        for(int i=0; i<N; i++){             //check -1 and replace with 0
            for(int j=0; j<M; j++){
                if(A[i][j] == -1){
                    A[i][j]=0;
                }
            }
        }
        return A;
    }
}

// TC -> O(N^2)
// SC -> O(1)
```

Q1. Merge Intervals

```
/**
 * Definition for an interval.
 * public class Interval {
```

```
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval newInterval) {
        ArrayList<Interval> result= new ArrayList<Interval>();
        for(int i=0; i<intervals.size(); i++){
            Interval curr= intervals.get(i);

            if(newInterval.start > curr.end){          //before overlapping we will print
                result.add(curr);
            }
            else if(newInterval.end < curr.start){            ////after overlapping we will
print overlap inerval and
                result.add(newInterval);                        // start printing other
remaining intervals and end the loop
                for(int j=i; j<intervals.size(); j++){
                    result.add(intervals.get(j));
                }
                return result;
                // newInterval = curr;
            }
            else{                                             //if we find out overlapped we
will mearge with new interval
                newInterval.start = Math.min(curr.start, newInterval.start);
                newInterval.end = Math.max(curr.end, newInterval.end);
                // result.add(newInterval);
            }
        }
        result.add(newInterval);
        return result;
    }
}
```

Q2. Merge Overlapping Intervals

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public ArrayList<Interval> merge(ArrayList<Interval> intervals) {
        Collections.sort(intervals, new intervalsort());
        ArrayList<Interval> ans = new ArrayList<Interval>();
        int N= intervals.size();
        Interval temp;
        Interval curr = intervals.get(0);
        for(int i=1; i<N; i++){
            Interval Arr = intervals.get(i);
            //non overlap
            if(curr.start > Arr.end || curr.end < Arr.start){  //curr.start > Arr.end cane use
without this also
                temp= new Interval(curr.start, curr.end);      //as intervals are sorted
                ans.add(temp);
                curr.start=  Arr.start;
                curr.end= Arr.end;
            }
            //overlap
            else{
                curr.start = Math.min(curr.start, Arr.start);
                curr.end = Math.max(curr.end, Arr.end);
            }
        }
        temp = new Interval(curr.start, curr.end);
        ans.add(temp);
        return ans;
    }

    public class intervalsort implements Comparator<Interval>{
        public int compare(Interval A, Interval B){
            if(A.start < B.start){
                return -1;
            }
            else{
                return 1;
            }
        }
    }
}

// TC -> O(NlogN + N)
// SC -> O(1)  // needss to check for Comparator
```

Q3. First Missing Integer

```
public class Solution {
    public int firstMissingPositive(int[] A) {
        int N= A.length;
        int temp;

        for(int i=0; i<N; i++){
            // swap numbers
            while(A[i] > 0  &&  A[i] <= N  &&  A[A[i]-1] != A[i]){
                temp = A[A[i]-1];
                A[A[i]-1] = A[i];  //doing swap(A[A[i]-1] , A[i])
                A[i] = temp;
            }
        }
        for(int i=0; i<N; i++){
            if(A[i] != i+1){
                return i+1;
            }
        }

        return N+1;
    }
}

// Tc- O(N)
// SC -> O(1)
```

Bit Manipulation

Q1. Set Bit

```
public class Solution {
    public int solve(int A, int B) {
        int ans=0;
        ans= (ans | (1 << A));
        ans= (ans | (1 << B));
        return ans;
        // return (0 | (1 << A) | (1 << B));
    }
}
```

Q2. Unset i-th bit

```
public class Solution {
    public int solve(int A, int B) {
        if((A & (1 << B)) > 0 ){
            A= A ^ (1 << B);
        }
        return A;
    }
}
```

Q3. Check bit

```
public class Solution {
    public int solve(int A, int B) {
        if((A & (1 << B)) >0){
            return 1;
        }
        else{
            return 0;
        }
    }
}
```

Q4. Number of 1 Bits

Write a function that takes an integer and returns the number of 1 bits present in its binary representation.

```java
public class Solution {
    public int numSetBits(int A) {
        int count=0;
        for(int i=0; i<32 ; i++){
            if((A & (1 << i)) >0 ){
                count++;
            }
        }
        return count;
    }
}
```

Q5. Help From Sam

```java
public class Solution {
    public int solve(int A) {
        int count=0;
        for(int i=0; i<32; i++){
            if((A & (1 << i)) >0){
                count++;
            }
        }
        return count;
    }
}

// Time Complexity : O( log(A) )
```

Q1. Single Number

```java
// public class Solution {
//     // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
//     public int singleNumber(final int[] A) {
//         int ans = 0;
//         int N=A.length;
//         for(int i=0; i<N; i++){
//             ans= ans ^ A[i];
//         }
//         return ans;
//     }
// }


public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int singleNumber(final int[] A) {
        int N=A.length;
        int ans=0;
        for(int i=0; i<32; i++){
            int count=0;
            for(int j=0; j<N; j++){
                if((A[j] & (1<<i)) >0 ){
                    count++;
                }
            }
            count= count %2;
            if(count == 1){
                // ans=ans + (int)Math.pow(2,i);
                ans = ans | (1<<i);
            }
        }
        return ans;
    }
}
```

Q2. Single Number II

Common code (every element appears thrice except for one, which occurs once.)

```
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY
    public int singleNumber(final int[] A) {
        int N=A.length;
        int ans=0;
        for(int i=0; i<32; i++){
            int count=0;
            for(int j=0; j<N; j++){
                if((A[j] & (1<<i)) >0 ){
                    count++;
                }
            }
            count= count %3;
            if(count == 1){
                // ans=ans + (int)Math.pow(2,i);
                ans = ans | (1<<i);
            }
        }
        return ans;
    }
}
```

Q3. Single Number III

Given an array of positive integers **A**, two integers appear only once, and all the other integers appear twice.
Find the **two integers that appear only once**.
Note: Return the two numbers in ascending order.

```
public class Solution {
    public int[] solve(int[] A) {
        int N= A.length;
        int exor=0;
        int a=0;
        int b=0;
        int[] arr= new int[2];
        int temp=0;
        for(int i=0; i<N; i++){
            exor= exor ^ A[i];
        }
        int index=0;
        for(int i=0; i<32; i++){
            if((exor & (1<<i)) >0){
                break;   //finding set bit to distingbus
            }
            else{
                index++;
            }
        }

        for(int i=0; i<N ;i++){
            if((A[i] & (1 << index)) > 0){  //set group part
                a= a ^ A[i];
            }
            else{
                b= b ^ A[i];   //unset group part
            }
        }

        arr[0]= Math.min(a,b);
        arr[1]= Math.max(a,b);

        return arr;
    }
}
```

Q4. Find Two Missing Numbers

```
public class Solution {
    public int[] solve(int[] A) {
        int xor = 0;
        for(int i = 0; i < A.length; i++){
            xor = xor^A[i];
        }
        for(int i = 1; i <= A.length+2; i++){ //similar to Q3. Single Number III but here they
give rage so we
            xor = xor^i;                      //just doing exor of that also to get distinct
missing integers
        }
        int idx = 0;
        for(int i =31; i>=0; i--){    //get set MSB so we don't need to use sorting in last
            if((xor & (1<<i)) != 0){
                idx = i;
                break;
            }
        }
        int ans1 = 0; int ans2 = 0;
        for(int i = 0; i < A.length; i++){
            if((A[i] & (1<<idx)) == 0){
                ans1 = ans1^A[i];
            }
            else{
                ans2 = ans2^A[i];
            }
        }
        for(int i = 1; i <= A.length+2; i++){
            if((i & (1<<idx)) == 0){
                ans1 = ans1^i;
            }
            else{
                ans2 = ans2^i;
            }
        }
        int[] ans = new int[2];
        ans[0] = ans1;
        ans[1] = ans2;
        // Arrays.sort(ans);
        return ans;
    }
}

// A = [5, 1, 3, 6]
// N = 4
// all the elements [1, 6] -> [1, 2, 3, 4, 5, 6]
// 2, 4
// new array = [5, 1, 3, 6, 1, 2, 3, 4, 5, 6, 7, 8, 8]
```

Q5. Maximum AND Pair

```
public class Solution {
    public int solve(int[] A) {
        int N=A.length;
        int[] arr= new int[2];

        for(int i=31; i>=0; i--){
            int count=0;
                for(int j=0; j<N; j++){
                    if((A[j] & (1 << i)) > 0){
                        count++;       //countting how bits are set
                    }
                }
                if(count >= 2){
                    for(int k=0; k<N; k++){
                        if((A[k] & (1 << i)) == 0){
                            A[k]=0;    //elimation part
                        }
                    }
                }

        }
        int index=0;
        for(int i=0; i<N; i++){
            if(A[i]>0){
                arr[index++]=A[i];
                if(index == 2){
                    break;
                }
            }
        }

        return (arr[0] & arr[1]);
    }
}

// TC -> O(N)
// SC -> O(1)

// if they ask to count how much pairs will give Maximum AND Pair answer
// then use combination formula on all remaining nonzero elements
// nC2= (n*(n-1))/2
```

Recursion

Q1. Find Fibonacci – II

```
//test
public class Solution {
    public int findAthFibonacci(int A) {
        if(A == 0 || A == 1){
            return A;
        }
        return findAthFibonacci(A-1) + findAthFibonacci(A-2);
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q2. Find Factorial!

```
public class Solution {
    public int solve(int A) {
        if(A == 1){
            return 1;
        }
        return A * solve(A-1);
    }
}


// TC -> O(N)
// SC -> O(N)
```

Q3. Check Palindrome using Recursion

```
public class Solution {
    public int solve(String A) {
        int i=0;
        int j=A.length()-1;
        return pali(i,j,A);
    }
    public int pali(int i, int j, String A){
        if(i >= j){
            return 1;
        }
        if(A.charAt(i) != A.charAt(j)){
            return 0;
        }
        return pali( i+1, j-1, A);
    }
}

//TC -> O(N)
//SC -> O(N)


// public class Solution {
//     public int solve(String A) {
//         int N=A.length();
//         for(int i=0; i<N/2; i++){
//             if(A.charAt(i) != A.charAt(N-1-i)){
//                 return 0;
//             }
//         }
//         return 1;
//     }
// }
```

Q4. Print 1 to A function

```
public class Solution {
    public void solve(int A) {
        seq(A);
        System.out.println();
    }
    public void seq(int A){
        if(A == 0){
            return;
        }
        seq(A-1);
        System.out.print(A + " ");
    }
}

// TC -> O(N)
// SC -> O(N)
```

Q1. Kth Symbol – Hard

```
public class Solution {
    public int find(int n, long k){
         if(k==0){
            return 0;
         }
        int val = find(n- 1 , k / 2);
        if(k % 2 == 0){
            return val;
        }
        return 1 - val;
    }
    public int solve(int A, long B) {
        return find(A, B);
    }
}

// TC-> O(N)
// SC -> O(N)
// //  Row 1: 0 k = 0 -> val=0
//  Row 2: 01 k = 1 -> val=1
//  Row 3: 0110 k = 2 -> val=1
//  Row 4: 0110 1001 k = 5 -> val=0
//  Row 5: 01 10 10 01 100 1 0110 k = 11 -> val=1
//                         x


//     prev row = 0 1 0 1
//     next row = 01 10 01 10

//     next row, k=5th pos


//     get to prev and check the value at pos=5/2=2
//     value = 0 -> if k is even -> return value else if k is odd -> return 1 - value
//     (n, k)
//     (5, 10) -> 1

//     f(n, k) {

//         if(n == 0) return 0;

//         int val = f(n-1, k/2); prev row

//         if(k % 2 == 0) {
//             return val;
//         } else {
//             return 1-val;
//         }



//     }


//   0 -> [0, 1]
//   1 -> [2, 3]
//   2 -> [4, 5]
//   3 -> [6, 7]
//   4 -> [8, 9]
//   5 -> [10, 11]

//   k/2
```

Q2. Tower of Hanoi

```
public class Solution {
    public int[][] towerOfHanoi(int A) {
        int step=(int)Math.pow(2,A)-1;
        int[][] arr= new int[A.length][3];
        int s=1;
        int i=2;
        int d=3;

        TOH(A,s,i,d,arr);
        return arr;
    }
    public void TOH(int disk, int s, int i, int d, int[][] arr){
        TOH(disk-1, s, d, i);
        int[] temp = new int[3];
        temp[0]=disk;
        temp[1]=s;
        temp[2]=d;

    }
}
```

Maths

Q1. Prime Modulo Inverse

```
public class Solution {
    public int solve(int A, int B) {
        long ans=power(A,B-2,B);
        return (int)ans;
    }

    public long power(int A, int P, int B){
        if(P==0){
            return 1;
        }
        long temp= power(A, P/2, B);

        if(P%2 ==0 ){
            return (temp % B * temp % B) % B;
        }
        else{
            return (temp % B * temp % B *  A % B) % B;
        }
    }
}

// fermat's little theorem
// A(inversion) mod B = (A^(B-2)) %B only when gcd(A, B) = 1.

// A^B -> A^B/2 -> A^B/4 ->......-> A^0  = log(B)
// TC -> log(B)
// SC -> log(B)
```

Q2. Pair Sum divisible by M

```
public class Solution {
    public int solve(int[] A, int B) {
        int mod = 1000000007;
        int[] freq = new int[B+1];
        int N= A.length;
        for(int i=0; i<N; i++){
            A[i] = A[i] % B;
        }
        int x= 0;
        int sum=0;
        for(int i=0; i<N; i++){
            x= B - A[i];
            sum = sum + freq[x];
            freq[A[i]] += 1;
        }
        sum += (freq[0]*(freq[0]-1))/2;
        return sum % mod;
    }
}

// TC -> O(N)
// SC -> O(M)
```

Q3. Trailing Zeros in Factorial

```
public class Solution {
    public int trailingZeroes(int A) {
        int temp=5;
        int output=0;
        while(temp <= A){   // A/5 + A/25 + A/125 ...
            output = output + (A / temp);
            temp = temp *5;
        }
        return output;
    }
}

// TC -> O(log(A)to base 5)
// SC -> O(1)
```

Q4. Very Large Power

```
public class Solution {
    public int solve(int A, int B) {
        int factorial= (int)fact(B);
        int power=(int)power(A,factorial);
        // int ans= power %1000000007;
        return power;
    }
// (A^n) % m = ((A%m)^n) %m
    public long fact(long B){
        if(B==0){
            return 1;
        }
        return B * fact(B-1) %1000000006;  //Fernat's littel theorem(when taking mmod in
exponentiial part take m-1)
    }


    public long power(long x,long y){
        if(y==0){
            return 1;
        }

        long p= power(x,y/2);
        if(y%2 == 0){
            return (p %1000000007 * p %1000000007) %1000000007;
        }
        else{
            return (p %1000000007 * p %1000000007 * x %1000000007) %1000000007;
        }
    }
}

// TC -> B + log(N)
// O(B)
// SC -> O(1)
```

Q5. Greatest Common Divisor GCD

```
public class Solution {
    public int gcd(int A, int B) {
        if(B == 0){
            return A;
        }
        return gcd(B, A%B);
    }
}
```

Q6. Delete one GCD

Given an integer array **A** of size **N**. You have to delete **one** element such that the GCD**(Greatest common divisor)** of the remaining array is maximum.

Find the **maximum** value of GCD.

```
public class Solution {
    public int solve(int[] A) {
        int N= A.length;
        int[] prefix = new int[N];
        int[] suffix = new int[N];
        prefix[0]=A[0];
        for(int i=1; i<N; i++){
            prefix[i]= gcd(prefix[i-1], A[i]);
        }

        suffix[N-1]=A[N-1];
        for(int i=N-2; i>=0; i--){
            suffix[i]= gcd(suffix[i+1],A[i]);
        }

        int max=0;
        int ans=0;
        for(int i=0; i<N; i++){
            if(i==0){
                ans=suffix[i+1];
            }
            else if(i==N-1){
                ans=prefix[N-2];
            }
            else{
                ans=gcd(prefix[i-1], suffix[i+1]);
            }
            if(ans>max){
                max=ans;
            }
        }
        return max;
    }

    public int gcd(int A, int B){
        if(B==0){
            return A;
        }
        return gcd(B,A%B);
    }
}

// TC -> O(N * log(A[i])) in which we are iterating over A[]
// SC -> O(1)
```

Q1. Compute nCr % m

```
public class Solution {
    public int solve(int A, int B, int C) {
        int[][] combi= new int[A+1][B+1];
        for(int i=0; i<=A; i++){
            for(int j=0; j<=B; j++){
                if(j==0 || i==j){
                    combi[i][j]=1;
                }
                else if(i<j){
                    break;
                }
                else{
                    combi[i][j]=combi[i-1][j-1] % C+ combi[i-1][j] % C;
                }
            }
        }
        return combi[A][B] % C;
    }
}

// Approach -> Simple Pascal Triangle Approach
// T.C -> "O(A*B)"
// S.C -> "O(A*B)"
```

Q2. Sorted Permutation Rank

```
public class Solution {
    private int mod = 1000003;
    public int fact(int n) {
        if(n == 0 || n == 1)
            return 1;
        else
            return (n * fact(n - 1)) % mod;
    }
    public int findRank(String A) {
        int ans = 0;
        int n = A.length();
        for(int i = 0; i < n - 1; i++) {
            int count = 0;  // count of characters less than A[i]
            for(int j = i + 1; j < n; j++)
                if(A.charAt(j) < A.charAt(i))
                    count++;
            ans += (count * fact(n - i - 1)) % mod;  //fact including dist from ith to n-1(n-
i)
        }                                            //and removing 1 for excluding current
ith position
        return (ans + 1) % mod;
        //consider 5 string smaller than my current string so rank is 6 which is (+1)
    }
}

// a d c b
//     i j

// count 0 2 1
// ans   0 4 5

// ans + 1 = 6
```

Q3. Excel Column Title

```
//Discussion
```

```
public class Solution {
    public String convertToTitle(int A) {
        char[] arr =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','
X','Y','Z'};
        int num=A;
        StringBuilder st = new StringBuilder();
        while(num>0){
            num--;
            int rem=num%26;
            st.append(arr[rem]);
            num=(num/26);
        }
        StringBuilder stans= st.reverse();
        return stans.toString();
    }
}

// TC -> O(logN to base 26)
// SC -> O(logN to base 26)
```

Q4. Pascal Triangle

```
public class Solution {
    public int[][] solve(int A) {
        int[][] pascal= new int[A][A];
        for(int i=0; i<A; i++){
            for(int j=0; j<A; j++){
                if(i==j || j==0){
                    pascal[i][j]=1;
                }
                else if(i<j){
                    break;
                }
                else{
                    pascal[i][j]= pascal[i-1][j-1] + pascal[i-1][j];
                }
            }
        }
        return pascal;
    }
}

// Approach -> Simple Pascal Triangle Approach
```

Prime

Q1. Find All Primes

```
public class Solution {
    public int[] solve(int A) {
        Boolean[] Primearr= new Boolean[A+1];
        int N= Primearr.length;  //A+1
        // Arrays.fill(myArray, true);
        Arrays.fill(Primearr,true);
        Primearr[0] =false;
        Primearr[1] =false;
        for(int i=2; i * i <= A ; i++){
            if(Primearr[i] == true){
                for(int j=i*i; j<=A; j=j+i){
                    if(Primearr[j]== true){
                        Primearr[j]=false;
                    }
                }
            }
        }

        ArrayList<Integer> temp = new ArrayList<Integer>();
        for(int i=0; i<=A; i++){
            if(Primearr[i] == true){
                temp.add(i);
            }
        }

        int[] ans = new int[temp.size()];
        for(int i=0; i<temp.size(); i++){
            ans[i]=temp.get(i);
        }
        return ans;
    }
}
```

Q2. Count of divisors

```
public class Solution {
    public int[] solve(int[] A) {
        int N = A.length;
        int max= A[0];
        for(int i=1; i<N; i++){
            if(max < A[i]){
                max=A[i];
            }
        }
        int[] divisorarr = new int[max+1];

        for(int i=1; i<=max; i++){
            for(int j=i; j<=max; j=j+i){
                divisorarr[j]=divisorarr[j]+1;
            }
        }

        int[] ans = new int[N];
        for(int i=0 ; i<N; i++){
            ans[i] = divisorarr[A[i]];
        }
        return ans;
    }
}

// max/2 max/3 max/4  ..... 1(max/max)
// TC - max * log(max);

// arr[]  0 1 2 2 3 2 4 2 3 3 3
// i      0 1 2 3 4 5 6 7 8 9 A
```

Q3. Sorted Permutation Rank

```
public class Solution {
    private int mod = 1000003;
    public int fact(int n) {
        if(n == 0 || n == 1)
            return 1;
        else
            return (n * fact(n - 1)) % mod;
    }
    public int findRank(String A) {
        int ans = 0;
        int n = A.length();
        for(int i = 0; i < n - 1; i++) {
            int count = 0;  // count of characters less than A[i]
            for(int j = i + 1; j < n; j++)
                if(A.charAt(j) < A.charAt(i))
                    count++;
            ans += (count * fact(n - i - 1)) % mod;
        }
        return (ans + 1) % mod;
    }
}


// a d c b
//     i j

// count 0 2 1
// ans   0 4 5

// ans + 1 = 6
```