1. Evaluate Reverse Polish Notation link

   Approach :

   Using Stacks.

   Complexity:

   Time Complexity: O(N)

   Space Complexity: O(N)

   Code:

```
class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack=new Stack();
        String operator="+-*/";
        for(String s:tokens){
            if(operator.contains(s)&&!stack.isEmpty()){
              int temp1=stack.pop();
              int temp2=stack.pop();
              int ans=helper(temp2,s,temp1);
              stack.push(ans);
            }else{
                stack.push(Integer.parseInt(s));
            }
        }
        return stack.pop();
    }
    public int helper(int temp2,String s,int temp1){
        if(s.equals("+"))return temp2+temp1;
        else if(s.equals("-"))return temp2-temp1;
        else if(s.equals("/"))return temp2/temp1;
        return temp2*temp1;
    }
}
```

2. Combination Sum III link

   Approach :

   BackTracking and Recursion

   Complexity:

   Time Complexity: O(N)

   Space Complexity: O(N)

Code:

```
class Solution {
    public List<List<Integer>> combinationSum3(int k, int sum) {
        List<Integer> temp=new ArrayList();
        List<List<Integer>> res=new ArrayList();
        helper(k, sum, 1, temp, res);
        return res;

    }
public static void helper(int k, int sum, int index, List<Integer> temp,
List<List<Integer>> res)
    {
        if(k==0 && sum==0)
        {
            res.add(new ArrayList(temp));
            return;
        }
        for(int i=index;i<10;i++)
        {
            temp.add(i);
            helper(k-1, sum-i, i+1, temp, res);
            temp.remove(temp.size()-1);
        }
    }
}
```

3. Bulls and Cows [link]

Approach :

Hashing

Complexity:

Time Complexity: O(N)

Code:

```
class Solution {
    public String getHint(String secret, String guess) {
        int bulls = 0, cows = 0;

        Map<Character, Integer> secret1 = new HashMap<>();
        Map<Character, Integer> guess1 = new HashMap<>();
```

```
            for (int i = 0; i < secret.length(); i++) {
                char s = secret.charAt(i);
                char g = guess.charAt(i);
                if (s == g) bulls++;
                else {
                    secret1.put(s, secret1.getOrDefault(s, 0) + 1);
                    guess1.put(g, guess1.getOrDefault(g, 0) + 1);
                }
            }
            for (char c : secret1.keySet()) {
                if (guess1.containsKey(c)) {
                    cows += Math.min(secret1.get(c), guess1.get(c));
                }
            }
            return bulls + "A" + cows + "B";
        }
    }
```

4. Rotate Function [link]

   Approach :

   Mathematics, Arrays

   Complexity:

   Time Complexity: O(N)

   Code:

```
    class Solution {
        public int maxRotateFunction(int[] nums){
          int sum=0;
          int result=0;
          int n=nums.length;
          for(int i=0;i<n;i++){
              sum+=nums[i];
              result+=i*nums[i];
          }
          int finalresult=result;
          for(int i=n-1;i>0;i--){
              finalresult=Math.max(finalresult,result+sum-(n*nums[i]));
              result=result+sum-(n*nums[i]);
          }
          return finalresult;
        }
    }
```

5. Largest Divisible Subset [link]

Approach :

Mathematics, Arrays

Complexity:

Time Complexity: O(N)

Code:

```java
class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums) {
        int[] lis = new int[nums.length];
        int[] hash = new int[nums.length];
        Arrays.fill(hash, -1);
        Arrays.sort(nums);
        int maxIdx = 0;
        for(int i=1; i<nums.length; ++i){
            for(int j=0; j<i; ++j){
                if(nums[i]%nums[j] == 0 && lis[j]+1 > lis[i]){
                    lis[i] = lis[j]+1;
                    hash[i] = j;
                    if(lis[maxIdx] < lis[i])
                        maxIdx = i;
                }
            }
        }

        return buildSeq(nums, hash, maxIdx);
    }

    public List<Integer> buildSeq(int[] nums, int[] lis, int idx){
        List<Integer> seq = new ArrayList<>();
        while(idx >=0){
            seq.add(nums[idx]);
            idx = lis[idx];
        }
        return seq;
    }
}
```

6. Perfect Rectangle [link]

Approach :

Mathematics, Arrays, Hashing

Complexity:

Time Complexity: O(N)

Code:

```
class Solution {
    public boolean isRectangleCover(int[][] rectangles) {
        if (rectangles.length == 0 || rectangles[0].length == 0) return
false;
        int x1 = Integer.MAX_VALUE;
        int x2 = Integer.MIN_VALUE;
        int y1 = Integer.MAX_VALUE;
        int y2 = Integer.MIN_VALUE;

        HashSet<String> set = new HashSet<String>();
        int area = 0;

        for (int[] rect : rectangles) {
            x1 = Math.min(rect[0], x1);
            y1 = Math.min(rect[1], y1);
            x2 = Math.max(rect[2], x2);
            y2 = Math.max(rect[3], y2);

            area += (rect[2] - rect[0]) * (rect[3] - rect[1]);

            String s1 = rect[0] + " " + rect[1];
            String s2 = rect[0] + " " + rect[3];
            String s3 = rect[2] + " " + rect[3];
            String s4 = rect[2] + " " + rect[1];

            if (!set.add(s1)) set.remove(s1);
            if (!set.add(s2)) set.remove(s2);
            if (!set.add(s3)) set.remove(s3);
            if (!set.add(s4)) set.remove(s4);
        }
        if (!set.contains(x1 + " " + y1) || !set.contains(x1 + " " + y2)
|| !set.contains(x2 + " " + y1) || !set.contains(x2 + " " + y2) ||
set.size() != 4) return false;

        return area == (x2-x1) * (y2-y1);
    }
}
```

7. Course Schedule link

   Approach :

   Queues

   Complexity:

   Time Complexity: O(N+E)

   Code:

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
    int[][] matrix = new int[numCourses][numCourses]; // i -> j
    int[] indegree = new int[numCourses];

    for (int i=0; i<prerequisites.length; i++) {
        int ready = prerequisites[i][0];
        int pre = prerequisites[i][1];
        if (matrix[pre][ready] == 0)
            indegree[ready]++; //duplicate case
        matrix[pre][ready] = 1;
    }

    int count = 0;
    Queue<Integer> queue = new LinkedList();
    for (int i=0; i<indegree.length; i++) {
        if (indegree[i] == 0) queue.offer(i);
    }
    while (!queue.isEmpty()) {
        int course = queue.poll();
        count++;
        for (int i=0; i<numCourses; i++) {
            if (matrix[course][i] != 0) {
                if (--indegree[i] == 0)
                    queue.offer(i);
            }
        }
    }
    return count == numCourses;
}
}
```

8. Most Profitable Path in a Tree link

   Approach :

Graphs

Complexity:

Time Complexity: O(N+E)

Code:

```
class Solution {
        class Node {
         int nd;
         int amount;
         public Node(int nd, int amount) {
             this.amount = amount;
             this.nd = nd;
         }
    }
    public int mostProfitablePath(int[][] edges, int bob, int[] amount){
         int m = edges.length;

         Map<Integer, List<Integer>> graph = new HashMap<>();
         Set<Integer> visited = new HashSet<>();
         Set<Integer> opened = new HashSet<>();
         Map<Integer, Integer> parent = new HashMap<>();
         parent.put(0, -1);
         for (int i = 0; i < m; i++) {
             int a = edges[i][0];
             int b = edges[i][1];
             graph.putIfAbsent(a, new ArrayList<>());
             graph.putIfAbsent(b, new ArrayList<>());
             graph.get(a).add(b);
             graph.get(b).add(a);
         }
         updateParent(graph, visited, 0, parent);
         visited.clear();
         Queue<Node> queue = new LinkedList<>();

         queue.add(new Node(0, amount[0]));
         int max = Integer.MIN_VALUE;
         while (!queue.isEmpty()) {
             if (bob != -1) {
                 opened.add(bob);
                 bob = parent.get(bob);
             }
             int l = queue.size();
```

```
                while (l > 0) {
                    Node node = queue.poll();
                    visited.add(node.nd);
                    int val1 = node.amount;
                    boolean isLeaf = true;
                    for (Integer nd : graph.getOrDefault(node.nd,new
ArrayList<>())) {
                        if (!visited.contains(nd)) {
                            isLeaf = false;
                            int val = val1;
                            if (!opened.contains(nd)) {
                                if (bob == nd) {
                                    val += amount[nd] / 2;
                                } else {
                                    val += amount[nd];
                                }
                            }
                            queue.add(new Node(nd, val));
                        }
                    }
                    if (isLeaf) {
                        max = Math.max(max, node.amount);
                    }
                    l--;
                }
            }
            return max;
        }
        private void updateParent(Map<Integer, List<Integer>> graph,
Set<Integer> visited,int node, Map<Integer, Integer> parent) {
            if (visited.contains(node)) return;
            visited.add(node);
            for (Integer nd : graph.get(node)) {

                if(!visited.contains(nd)){
                    parent.put(nd, node);
                    updateParent(graph, visited, nd, parent);
                }
            }
        }
    }
```

9. Number of Pairs Satisfying Inequality [link]

Code : `class Solution {`

```
    long ans=0;
    int diff;
    public long numberOfPairs(int[] nums1, int[] nums2, int diff) {
        this.diff=diff;
        int n=nums1.length;
        int[] nums=new int[n];
        for(int i=0;i<n;i++){
            nums[i]=nums1[i]-nums2[i];
        }
        mergeSort(nums,0,n-1);
        return ans;
    }
    public void mergeSort(int[] nums,int i,int j){
        if(i>=j) return;
        int mid=i+(j-i)/2;
        mergeSort(nums,i,mid);
        mergeSort(nums,mid+1,j);
        calculateInverse(nums,i,mid,j);
        merge(nums,i,mid,j);
    }
    public void calculateInverse(int[] nums,int i,int mid,int j){
        int[] temp=new int[j-i+1];
        int ptr1=i;
        int ptr2=mid+1;
        int ptr=0;
        while(ptr1<=mid && ptr2<=j){
            if(nums[ptr1]>nums[ptr2]+diff){
                temp[ptr]=nums[ptr2]+diff;
                ans=ans+(long)ptr1-i;
                ptr2++;
                ptr++;
            }
            else{
                temp[ptr]=nums[ptr1];
                ptr++;
                ptr1++;
            }
        }
        if(ptr2<=j){
            ans=ans+(long)(ptr1-i)*(j-ptr2+1);
```

```
        }
    }
    public void merge(int[] nums,int i,int mid,int j){
        int[] temp=new int[j-i+1];
        int ptr1=i;
        int ptr2=mid+1;
        int ptr=0;
        while(ptr1<=mid && ptr2<=j){
            if(nums[ptr1]>nums[ptr2]){
                temp[ptr]=nums[ptr2];
                ptr2++;
                ptr++;
            }
            else{
                temp[ptr]=nums[ptr1];
                ptr++;
                ptr1++;
            }
        }
        while(ptr1<=mid){
            temp[ptr]=nums[ptr1];
            ptr1++;ptr++;
        }
        while(ptr2<=j){
            temp[ptr]=nums[ptr2];
            ptr2++;ptr++;
        }
        for(int k=0;k<temp.length;k++){
            nums[i]=temp[k];
            i++;
        }
    }
}
```

10. Shortest Unsorted Continuous Subarray [link]

Code: `class Solution {`

```
    public int findUnsortedSubarray(int[] nums) {
        int n = nums.length;
        int end = -1, max = nums[0];
        for (int i = 1; i < n; i++){
```

```
            if (nums[i] < max)
                end = i;
            else
                max = nums[i];
        }
        int start = 0, min = nums[n - 1];
        for (int i = n - 2; i >= 0; i--){
            if (nums[i] > min)
                start = i;
            else
                min = nums[i];
        }
        return end - start + 1;
    }
}
```

11. Number of Ways to Arrive at Destination [link]

Code:

```
class Solution {
    private final long inf = (1L<<62);
    private final int MOD = 1000000007;

    public int countPaths(int n, int[][] roads) {
        Map<Integer, List<Integer>> edges = new HashMap<>();
        Map<Integer, List<Integer>> costs = new HashMap<>();

        for(int i=0;i<n;i++){
            edges.putIfAbsent(i, new ArrayList<Integer>());
            costs.putIfAbsent(i, new ArrayList<Integer>());
        }

        for(int[] road : roads){
            edges.get(road[0]).add(road[1]);
            edges.get(road[1]).add(road[0]);
            costs.get(road[0]).add(road[2]);
            costs.get(road[1]).add(road[2]);
        }

        long dis[] = new long[n];
        Arrays.fill(dis, inf);
```

```
        dis[0]=0;

        int paths[] = new int[n];
        paths[0]=1;

        PriorityQueue<Integer> pq = new PriorityQueue<>((a,b)->{
            return (int)(dis[a]-dis[b]);
        });
        pq.add(0);

        while(!pq.isEmpty()){
            int u = pq.poll();
            // System.out.println(u);
            for(int vi=0;vi<edges.get(u).size();vi++){
                int v = edges.get(u).get(vi);
                int c = costs.get(u).get(vi);

                if(dis[u]+c<dis[v]){
                    dis[v]=dis[u]+c;
                    paths[v]=paths[u];
                    pq.remove(v); // I was getting WA because of not
removing the older one
                    pq.add(v);
                }
                else if(dis[u]+c==dis[v]){
                    paths[v]=(paths[v]+paths[u])%MOD;
                }
            }
        }
        return paths[n-1];
    }
}
```

12. Longest Happy Prefix [link]

Code: `class Solution {`

```
    public String longestPrefix(String s) {
        int n=s.length();
        int[] arr=new int[n];
        arr[0]=0;
        int len=0;
        int i=1;
```

```
        while(i<n){
            if(s.charAt(i)==s.charAt(len)){
                arr[i++]=++len;
            }
            else{
                if(len==0)
                    arr[i++]=0;
                else
                    len=arr[len-1];
            }
        }
        return s.substring(0,arr[n-1]);
    }
}
```

13. Airplane Seat Assignment Probability link

Code:

```
class Solution {
    public double nthPersonGetsNthSeat(int n) {
        if(n==1)
            return n;
        return (double)1/2;
    }
}
```

14. Minimum Deletions to Make Array Divisible link

Code: class Solution {

```
    public int minOperations(int[] nums, int[] numsDivide) {
        boolean flag=false;
        int ans=0;
        int minnum=0;
        Arrays.sort(nums);
        int count1=0;
        Map<Integer,Integer>map=new TreeMap<>();
        for(int i=0; i<nums.length; i++)
        {
             map.put(nums[i],map.getOrDefault(nums[i],0)+1);
        }
        for(Map.Entry<Integer,Integer>e:map.entrySet())
        {
```

```
            int cnum=e.getKey();
            for(int i=0; i<numsDivide.length; i++){
                if(numsDivide[i]%cnum==0){
                    continue;
                }else{
                            flag=true;
                            break;
                    }
            }
            if(!flag){
                minnum=e.getKey();
                break;
            }
            else{
                ans=ans+e.getValue();
            }
            flag=false;
        }
        if(minnum==0){
            return -1;
        }
        return ans;
    }
}
```

15. Number of Substrings Containing All Three Characters [link]

Code: `class Solution {`

```
    public int numberOfSubstrings(String s){
        Map<Character,Integer> freqMap = new
HashMap<Character,Integer>();
        int leftPointer =0, result =0;
        for(int rightPointer=0; rightPointer<s.length(); rightPointer++)
{

            char rc = s.charAt(rightPointer);
            freqMap.put(rc, freqMap.getOrDefault(rc, 0) + 1);

            while (freqMap.getOrDefault('a', 0) > 0 &&
                   freqMap.getOrDefault('b', 0) > 0 &&
                   freqMap.getOrDefault('c', 0) > 0) {
                char leftCharacter = s.charAt(leftPointer);
```

```
                freqMap.put(leftCharacter, freqMap.get(leftCharacter) -
1);
                leftPointer++;
            }

            result = result + leftPointer;
        }
        return result;
    }
}
```