JAVA DSA NOTES

1.Chips Factory

//All the zeros to the end

Code:

```
public class Solution {
    public int[] solve(int[] A) {
        int j=0;
        for(int i=0;i<A.length;i++)
        {
            if(A[i]==0)
            {
                j=i;
                break;
            }
        }
        for(int i=j+1;i<A.length;i++)
        {
            if(A[i]!=0)
            {
```

```
                int temp=A[j];
                A[j]=A[i];
                A[i]=temp;
                j++;
            }
        }
        return A;
    }
}
```

2.Greater than all

Code:

```
public class Solution {
    public int solve(int[] A) {
        int count=1;
         int max=A[0];

         for(int j=1;j<A.length;j++)
         {
            if(A[j]>max)
            {
               max=A[j];
               count++;
            }
         }

        return count;
    }
}
```

Example Input

Input 1:

```
A = [1, 2, 3, 4]
```

Input 2:

```
A = [1, 1, 2, 2]
```

Example Output

Output 1:

```
4
```

Output 2:

```
2
```

3.Pythagoras Triplets:

Code:

```
public class Solution {
    public int solve(int A) {
         int[]arr=new int[A];
        int a=1;
        for(int i=0;i<A;i++)
        {
            arr[i]=a*a;
            a++;
        }
        int count=0;
        for(int i=0;i<A;i++)
        {
            for(int j=0;j<A-1;j++)
            {
                for(int k=j+1;k<A;k++)
                {
```

```
                    if(arr[i]==arr[j]+arr[k])
                    {
                        count++;
                    }
                }
            }
        }
        return count;

    }
}
```

Example Input

Input 1:

```
A = 5
```

Input 2:

```
A = 13
```

Example Output

Output 1:

```
1
```

Output 2:

```
3
```

Example Explanation

Explanation 1:

```
Then only triplet is {3, 4, 5}
```

Explanation 2:

```
The triplets are {3, 4, 5}, {6, 8, 10}, {5, 12, 13}.
```

4.Diagonal Flip:

Code:

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

        for(int i=0;i<n;i++)
        {
            for(int j=0;j<i;j++)
            {
                int temp=A[i][j];
                A[i][j]=A[j][i];
                A[j][i]=temp;
            }
        }
        return A;
    }
}
```

Example Input

Input 1:

```
A = 4
B = [[1, 0],
     [0, 1]]
```

Input 2:

```
A = [[1, 0],
```

```
      [1, 0]]
```

Example Output

Output 1:

```
[[1, 0],
 [0, 1]]
```

Output 2:

```
[[1, 1],
 [0, 0]]
```

5.Positive And Negative:

Code:

```
public class Solution {
    public ArrayList<Integer> solve(ArrayList<Integer> A) {
          ArrayList<Integer>arr=new ArrayList<>();
        int count=0;
        int count1=0;
        for(int i=0;i<A.size();i++)
        {
            if(A.get(i)>0)
            {
                count++;
            }
            else if(A.get(i)<0)
            {
                count1++;
            }
            else{
                continue;
            }
        }
```

```
        arr.add(count);
        arr.add(count1);
        return arr;
    }
}
```

6.N/3 Repeat Number:

Code

```
public class Solution {
    // DO NOT MODIFY THE LIST
    public int repeatedNumber(final List<Integer> a) {
         HashMap<Integer,Integer>map=new HashMap<>();

    for(int i=0;i<a.size();i++)
    {
        if(map.containsKey(a.get(i)))
        {
            map.put(a.get(i),map.get(a.get(i))+1);
        }
        else{
        map.put(a.get(i),1);
        }
    }

       int maxCount = 0;
        for (Map.Entry<Integer, Integer> entry : map.entrySet())
            int count = entry.getValue();
            if (count > maxCount) {
                maxCount = count;
            }
        }
         if((a.size()/3)<maxCount)
         {
            for(Map.Entry<Integer,Integer>entry:map.entrySet())
```

```
            {
                if(entry.getValue().equals(maxCount))
                {
                    return entry.getKey();
                }
            }
        }
        else{
            return -1;
        }
        return 0;
    }
}
```

Example Input

`[1 2 3 1 1]`

Example Output

`1`

Example Explanation

`1 occurs 3 times which is more than 5/3 times.`

7.Occurence of Each Number:

Code:

```
import java.util.*;
public class Solution {
    public ArrayList<Integer> findOccurences(ArrayList<Integer>


      TreeMap<Integer,Integer>map=new TreeMap<>();

      for(int i=0;i<A.size();i++)
      {
        if(map.containsKey(A.get(i)))
        {
```

```
          map.put(A.get(i),map.get(A.get(i))+1);
        }
        else{
          map.put(A.get(i),1);
        }
      }
    return new ArrayList<>(map.values());
    }
}
```

8.Greater of Lesser:

Code:

```
public class Solution {
    public int solve(int[] A, int[] B, int C) {
        int count=0;
        int count1=0;

        for(int i=0;i<A.length;i++)
        {
            if(A[i]>C)
            count++;
        }
         for(int i=0;i<B.length;i++)
        {
            if(B[i]<C)
            count1++;
        }
        int max=Math.max(count,count1);
        if(max>0)
        {
            return max;
        }
        return 0;
```

```
    }
}
```

Example Input

Input 1:

```
A = [1, 2, 3, 4]
B = [5, 6, 7, 8]
C = 4
```

Input 2:

```
A = [1, 10, 100]
B = [9, 9, 9]
C = 50
```

Example Output

Output 1:

```
0
```

Output 2:

```
3
```

Example Explanation

Explanation 1:

```
There are no integers greater than C in A.
There are no integers less than C in B.
```

Explanation 2:

```
Integers greater than C in A are [100].
```

```
Integers less than C in A are [9, 9,
```

9.Spiral Matrix:

Code:

```
public class Solution {
    public int[][] solve(int[] A, int B, int C) {
         int[][] spiralMatrix = new int[B][C];

        int top = 0, bottom = B - 1, left = 0, right = C - 1;

        int index = 0;

        while (top <= bottom && left <= right && index < A.lengt

                // Traverse left to right
                for (int i = left; i <= right; i++) {
                    spiralMatrix[top][i] = A[index++];
                }
                top++;

                // Traverse top to bottom
                for (int i = top; i <= bottom; i++) {
                    spiralMatrix[i][right] = A[index++];
                }
                right--;

                // Traverse right to left
                if(top<=bottom)
                {
                for (int i = right; i >= left; i--) {
                    spiralMatrix[bottom][i] = A[index++];
                }
                }
                bottom--;
```

```
                // Traverse bottom to top
                if(left<=right)
                {
                for (int i = bottom; i >= top; i--) {
                    spiralMatrix[i][left] = A[index++];
                }
                }
                left++;


    }
     return spiralMatrix;
}
}
```

10.Product of all:

Code:

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

        int[]left=new int[A.length];
        int[]right=new int[A.length];

  //left
       left[0]=1;
       for(int i=1;i<A.length;i++)
       {
           left[i]=(int)(((long)left[i - 1] * A[i - 1]) %1000000
       }
    //right
       right[A.length-1]=1;

       for(int i=A.length-2;i>=0;i--)
```

```
        {
            right[i]=(int)(((long)right[i+1] * A[i + 1]) % 10000
        }

        //product of left and right
         int []ans=new int[A.length];
        for(int i=0;i<A.length;i++)
        {
            ans[i]=(int)(((long)left[i]*right[i])%1000000007);
        }

        return ans;
    }
}
```

Example Input

Input 1:

```
A = [1, 2, 3, 4]
```

Input 2:

```
A = [9, 9, 9]
```

Example Output

Output 1:

```
[24, 12, 8, 6]
```

Output 2:

```
[81, 81, 81]
```

Example Explanation

Explanation 1:

```
[2×3×4, 1×3×4, 1×2×4, 1×2×3]
```

Explanation 2:

```
[9×9, 9×9, 9×9]
```

11.Set Matrix Zero:

Code:

```
public class Solution {
    public void setZeroes(ArrayList<ArrayList<Integer>> a) {

        int row[]=new int[a.size()];
        int col[]=new int[a.get(0).size()];

        for(int i=0;i<a.size();i++)
        {
            for(int j=0;j<a.get(0).size();j++)
            {
                if(a.get(i).get(j)==0)
                {
                    row[i]=1;
                    col[j]=1;
                }
            }
        }

         for(int i=0;i<a.size();i++)
        {
            for(int j=0;j<a.get(0).size();j++)
            {
                if(row[i]==1||col[j]==1)
                {
                     a.get(i).set(j, 0);
                }
```

```
            }
        }

    }
}
```

12.First Missing Number:

Code:

```
public class Solution {
    public int firstMissingPositive(ArrayList<Integer> A) {
        int n = A.size();


        for (int i = 0; i < n; i++) {
            if (A.get(i) <= 0) {
                A.set(i, n + 1);
            }
        }


        for (int i = 0; i < n; i++) {
            int num = Math.abs(A.get(i));
            if (num <= n) {
                A.set(num - 1, -Math.abs(A.get(num - 1)));
            }
        }

        for (int i = 0; i < n; i++) {
            if (A.get(i) > 0) {
                return i + 1;
            }
        }

        return n + 1;
```

```
    }
}
```

13.Maximum sum square submatrix:

Code:

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

       int n=A.length;

       //Prefix Sum table:

       for(int i=0;i<n;i++)
       {
           for(int j=0;j<n;j++)
           {
               if(i>0)
               {
               A[i][j]+=A[i-1][j];
               }
               if(j>0)
               {
               A[i][j]+=A[i][j-1];
               }
               if(i>0&&j>0)
               {
                   A[i][j]-=A[i-1][j-1];
               }
           }
       }

       int ans=Integer.MIN_VALUE;
       for(int i=B-1;i<n;i++)
```

```
        {
            for(int j=B-1;j<n;j++)
            {
                int local=A[i][j];//19
                if(i-B>=0)
                local-=A[i-B][j];//19-7
                 if(j-B>=0)
                local-=A[i][j-B];//19-7-4
                 if(i-B>=0&&j-B>=0)
                local+=A[i-B][j-B];//19-7-4+1

                ans=Math.max(local,ans);
            }
        }
         return ans;
    }
}
```

Example Input

Input 1:

```
A = [
       [1, 1, 1, 1, 1]
       [2, 2, 2, 2, 2]
       [3, 8, 6, 7, 3]
       [4, 4, 4, 4, 4]
       [5, 5, 5, 5, 5]
    ]
B = 3
```

Input 2:

```
A = [
       [2, 2]
       [2, 2]
```

```
    ]
B = 2
```

Example Output

Output 1:

```
48
```

Output 2:

```
8
```

14.Repeating and Missing Number:

Code:

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public ArrayList<Integer> repeatedNumber(final List<Integer>

    long n=A.size();

    long s=(n*(n+1))/2;
    long s2=((n*(n+1)*(2*n+1))/6);

    long sn=0;
    long sn2=0;

    for(int i=0;i<A.size();i++)
    {
        long num=A.get(i);
        sn+=num;
        sn2+=(long)num*(long)num;
    }
    long val1=s-sn;
    long val2=s2-sn2;
```

```
    val2=val2/val1;
    long x=(val1+val2)/2;
    long y=x-val1;

   ArrayList<Integer> result = new ArrayList<>();
        result.add((int) y);
        result.add((int) x);
        return result;
    }
}
```

15.Wave Array:

Code:

```
public class Solution {
    public int[] wave(int[] A) {
         Arrays.sort(A);
        for(int i=0;i<A.length-1;i+=2)
        {
            int temp=A[i];
            A[i]=A[i+1];
            A[i+1]=temp;
        }
       return A;
    }
}
```

16.Hotels booking possible:

```
public class Solution {
    public boolean hotel(ArrayList<Integer> arrive, ArrayList<I

        Collections.sort(arrive);
        Collections.sort(depart);
```

```
        int i=1;
        int j=0;
        boolean rev=true;
        int count=1;

        while(i<arrive.size()&&j<depart.size())
        {
            if(arrive.get(i)<=depart.get(j))
            {
                count++;
                if(count>K)
                {
                    rev=false;
                    break;
                }
                 i++;
           }

           else{
               count--;
               if(count<0)
               count=0;
                j++;
           }

        }
        return rev;


    }



}
```

Example Input

Input 1:

```
A = [1, 3, 5]
B = [2, 6, 8]
C = 1
```

Input 2:

```
A = [1, 2, 3]
B = [2, 3, 4]
C = 2
```

Example Output

Output 1:

```
0
```

Output 2:

```
1
```

Example Explanation

Explanation 1:

```
 At day = 5, there are 2 guests in the hotel. But I have only
one room.
```

Explanation 2:

```
At day = 1, there is 1 guest in the hotel.
At day = 2, there are 2 guests in the hotel.
At day = 3, there are 2 guests in the hotel.
At day = 4, there is 1 guest in the hotel.
```

```
 We have two rooms available, which satisfy the demand.
```

17.Max Distance:

Code:

18. Maximum Unsored subarray:

code:

```
public class Solution {
    public int[] subUnsort(int[] A) {

        int n=A.length;
        if(A.length<=1)
        return new int[]{-1};

        int l=-1;
        int r=-1;

        for(int i=0;i<n-1;i++)
        {
            if(A[i]>A[i+1])
            {
                l=i;
                break;
            }
        }
        if(l==-1)
        return new int[]{-1};

         for(int i=n-1;i>0;i--)
        {
            if(A[i]<A[i-1])
```

```
            {
                r=i;
                break;
            }
        }

        int min=Integer.MAX_VALUE;
        int max=Integer.MIN_VALUE;

        for(int i=l;i<=r;i++)
        {
            min=Math.min(min,A[i]);
            max=Math.max(max,A[i]);
        }

        while(l>0 && A[l-1]>min)
        {
            l--;
        }
        while(r<n-1 && A[r+1]<max)
        {
            r++;
        }


        return new int[]{l,r};

    }
}
```

Example Input

Input 1:

```
A = [1, 3, 2, 4, 5]
```

Input 2:

```
A = [1, 2, 3, 4, 5]
```

Example Output

Output 1:

```
[1, 2]
```

Output 2:

```
[-1]
```

19.Rotate Matrix:

Code:

```
public class Solution {
    public void rotate(ArrayList<ArrayList<Integer>> a) {


        int r=a.size();
        int c=a.get(0).size();

        for(int i=0;i<r;i++)
        {
            for(int j=0;j<i;j++)
            {
                int temp=a.get(i).get(j);
                a.get(i).set(j,a.get(j).get(i));
                a.get(j).set(i,temp);
            }
        }

        for(int i=0;i<r;i++)
        {
```

```
            for(int j=0;j<c/2;j++)
            {
                int temp=a.get(i).get(j);
                a.get(i).set(j,a.get(i).get(r-j-1));
                a.get(i).set(r-j-1,temp);
            }
        }
    }
}
```

20.Next Permutation:

Code:

```
public class Solution {
    public ArrayList<Integer> nextPermutation(ArrayList<Integer>

      int n = A.size(); // size of the array.

        // Step 1: Find the break point:
        int ind = -1; // break point
        for (int i = n - 2; i >= 0; i--) {
            if (A.get(i) < A.get(i + 1)) {
                // index i is the break point
                ind = i;
                break;
            }
        }

        // If break point does not exist:
        if (ind == -1) {
            // reverse the whole array:
            Collections.reverse(A);
            return A;
        }
```

```
        // Step 2: Find the next greater element
        //         and swap it with arr[ind]:

        for (int i = n - 1; i > ind; i--) {
            if (A.get(i) > A.get(ind)) {
                int tmp = A.get(i);
                A.set(i, A.get(ind));
                A.set(ind, tmp);
                break;
            }
        }

        // Step 3: reverse the right half:
        List<Integer> sublist = A.subList(ind + 1, n);
        Collections.reverse(sublist);

        return A;
}
}
```

21.Find Permutation:

Code:

```
public class Solution {
    // DO NOT MODIFY THE LIST. IT IS READ ONLY
    public ArrayList<Integer> findPerm(final String A, int B) {

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

        int start = 1;
        int end = B;

        for (char c : A.toCharArray()) {
            if (c == 'I') {
                result.add(start);
```

```
                start++;
            } else if (c == 'D') {
                result.add(end);
                end--;
            }
        }

        // Add the remaining value
        result.add(start); // or result.add(end);

        return result;
    }
}
```

```
Input 1:

n = 3

s = ID

Return: [1, 3, 2]
```

22.Noble Integer

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

        Arrays.sort(A);
        int n=A.length;

        for(int i=0;i<n;i++)
        {
            if(i<n-1&&A[i]==A[i+1])
            {
                continue;
```

```
            }
            if(A[i]==n-1-i)
            {
                return 1;
            }
        }
        return -1;
    }
}
```

Example Input

Input 1:

```
A = [3, 2, 1, 3]
```

Input 2:

```
A = [1, 1, 3, 3]
```

Example Output

Output 1:

```
1
```

Output 2:

```
-1
```

Example Explanation

Explanation 1:

```
 For integer 2, there are 2 greater elements in the array. S
o, return 1.
```

Explanation 2:

```
 There is no such integer exists.
```

23.Rearrange Array

```
public class Solution {
    public void arrange(ArrayList<Integer> a) {

        int n=a.size();
        ArrayList<Integer>b=new ArrayList<>(a);
        for(int i=0;i<n;i++)
        {
            a.set(i,b.get(b.get(i)));
        }

}}
Given an array A of size N. Rearrange the given array so that A
```

24.Grid Unique path

```
public class Solution {
    public int uniquePaths(int A, int B) {

        int[][] arr=new int[A][B];

        for(int i=0;i<B;i++)
        {
            arr[0][i]=1;
        }
        for(int i=0;i<A;i++)
        {
            arr[i][0]=1;
        }

        while(arr[A-1][B-1]==0)
        {
```

```
            for(int i=1;i<A;i++)
            {
                for(int j=1;j<B;j++)
                {
                    if(arr[i-1][j]!=0&&arr[i][j-1]!=0)
                    {
                        arr[i][j]=arr[i-1][j]+arr[i][j-1];
                    }
                }
            }
        }
        return arr[A-1][B-1];
    }
}
```

25.Armstrong Number

```
public class Solution {
    public int solve(int A) {

        int str=String.valueOf(A).length();
        int temp=A;
        int sum=0;
        while(A>0)
        {
            int rem=A%10;
            sum+=(Math.pow(rem,str));
            A/=10;
        }
        if(temp==sum)
        return 1;

        return 0;
    }
}
```

26.Power of Two Integers

```
public class Solution {
    public int isPower(int A) {

         if (A == 1) // Special case for 1
            return 1;

        for(int base=2;base<=Math.sqrt(A);base++)
        {
            int expo=(int)(Math.log(A)/Math.log(base));

            if(Math.pow(base,expo)==A)
            {
                return 1;
            }
        }
        return 0;
    }
}
```

27.Trailing Zeros inFactorial

```
public class Solution {
    public int trailingZeroes(int A) {
         int count=0;

        while(A>=5)
        {
          count+=A/5;
          A/=5;
        }
        return count;
    }
}
//5-120
```

```
//10-end with 00
//15 end with 000
```

28.Largest Coprime Divisor

```
public class Solution {
      public static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public int cpFact(int A, int B) {
        // Reduce A to its GCD with B
        while (gcd(A, B) != 1) {
            A = A / gcd(A, B);
        }
        return A;
    }
}
```

29.Sum of 7's multiple

code:

```
public class Solution {
    public long solve(int A, int B) {
       long firstTerm = A + (7 - A % 7) % 7; // Find the first t
        long lastTerm = B - B % 7; // Find the last term divisib
        long n = ((lastTerm - firstTerm) / 7) + 1; // Calculate
        long sum = (n * (firstTerm + lastTerm)) / 2; // Calculat
        return sum;
```

```
    }
}
```

Input 1:

```
A = 1
B = 7
```

Input 2:

```
A = 99
B = 115
```

Example Output

Output 1:

```
7
```

Output 2:

```
217
```