✅ EASY – Problem E1: Sum of Digits

📥 Input

ఒక పెద్ద integer ని string రూపంలో ఇస్తారు.

ఉదా:

12345

📤 Output

Digits మొత్తం.

15

🤔 Logic Explanation

సంఖ్యను string గా చదవాలి.

ప్రతి character digit అయితే దాన్ని integer గా మార్చి మొత్తం చేయాలి.

📝 Pseudocode

```
read N as string
sum = 0
for each character c in N:
   if c is digit:
      sum += (c - '0')
print sum
```

☕ Java Code

```
import java.io.*;
public class SumOfDigits {
   public static void main(String[] args) throws Exception {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
        String n = br.readLine().trim();
        long sum = 0;
        for (char c : n.toCharArray()) {
            if (Character.isDigit(c)) sum += c - '0';
        }
        System.out.println(sum);
    }
}
```

✅ Problem E2: Second Largest Distinct

📥 Input

5

2 8 8 6 3

📤 Output

6

🤔 Logic Explanation

Distinct values తీసుకోవాలి → set

Sort అయిన set లో పెద్దది తీసేసి → రెండవ పెద్దది ప్రింట్ చేయాలి.

📝 Pseudocode

read N

read N numbers

insert all numbers into a TreeSet (auto sorted, distinct)

if size == 1:

print only element

else:

remove last element (largest)

print new last element

Java Code

```
import java.io.*;
import java.util.*;
public class SecondLargestDistinct {
   public static void main(String[] args) throws Exception {
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      int n=Integer.parseInt(br.readLine());
      String[] parts=br.readLine().split(" ");
      TreeSet<Integer> set=new TreeSet<>();
      for(String p:parts) set.add(Integer.parseInt(p));
      if(set.size()==1){
         System.out.println(set.last());
         return;
      }
      set.pollLast();
      System.out.println(set.last());
   }
}
```

✅ MODERATE – Problem M1: First Non-Repeating Character

📥 Input

swiss

📤 Output

w

🤔 Logic Explanation

మొదట freq array ఉపయోగించి ప్రతి character occurrences లెక్కించాలి.

తరాత్ string ని left → right సాగ్ చేసి frequency = 1 ఉన్నమొదటి character ని ప్రింట్ చేయాలి.

📝 Pseudocode

```
read string S
freq[256] = {0}
for each c in S:
   freq[c]++

for each c in S:
   if freq[c] == 1:
      print c
      exit
print -1
```

☕ Java Code

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

```
public class FirstNonRepeatingChar {
   public static void main(String[] args) throws Exception {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String s = br.readLine();
      int[] freq = new int[256];
      for(char c : s.toCharArray()) freq[c]++;
      for(char c : s.toCharArray()) {
         if(freq[c] == 1) {
            System.out.println(c);
            return;
         }
      }
      System.out.println(-1);
   }
}
```

✅ Problem M2: Count Pairs with Given Sum

📥 Input

5 7

1 2 3 4 5

📤 Output

2

🧠 Logic Explanation

Pairs:

(2,5) → 7

(3,4) → 7

Frequency map దాన్తా x మరియు k-x pair లెకిథ్చాలి.

రెండు count multiply చేస్తే total pair count వస్తుంది.

📝 Pseudocode

```
read N, K
read array
freq map = empty

for each x in array:
   freq[x]++

answer = 0

for each key x in map:
   y = K - x
   if x < y: answer += freq[x] * freq[y]
   if x == y: answer += freq[x] * (freq[x] - 1) / 2

print answer
```

☕ Java Code

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

```
import java.util.*;
public class CountPairsWithSum {
   public static void main(String[] args)throws Exception {
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      String[] nk=br.readLine().split(" ");
      int n=Integer.parseInt(nk[0]);
      long k=Long.parseLong(nk[1]);
      String[] parts=br.readLine().split(" ");
      HashMap<Long,Long> map=new HashMap<>();
      for(String p:parts){
         long x=Long.parseLong(p);
         map.put(x,map.getOrDefault(x,0L)+1);
      }
      long ans=0;
      for(long x:map.keySet()){
         long y=k-x;
         if(!map.containsKey(y)) continue;
         if(x<y) ans+=map.get(x)*map.get(y);
         else if(x==y) ans+=map.get(x)*(map.get(x)-1)/2;
      }
      System.out.println(ans);
   }
}
```

✅ HARD – Problem H1: Longest Substring Without Repeating Characters

📥 Input

abcabcbb

📤 Output

3

🤔 Logic Explanation

Sliding Window Technique:

last[] array లో character చివరి position save చేయాలి

duplicate character window లో కనిపిస్తే start pointer move చేయాలి

ప్రతి step లో max length update చేయాలి

📝 Pseudocode

```
read S
last[256] = -1
start = 0
best = 0

for i from 0 to len(S)-1:
   c = S[i]
   if last[c] >= start:
```

```
    start = last[c] + 1
  last[c] = i
  best = max(best, i - start + 1)

print best
```

☕ Java Code

```
import java.io.*;
import java.util.*;
public class LongestUniqueSubstring {
  public static void main(String[] args)throws Exception {
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String s=br.readLine();
    int[] last=new int[256];
    Arrays.fill(last,-1);
    int start=0,best=0;
    for(int i=0;i<s.length();i++){
      char c=s.charAt(i);
      if(last[c]>=start) start=last[c]+1;
      last[c]=i;
      best=Math.max(best,i-start+1);
    }
    System.out.println(best);
  }
}
```

✅ HARD – Problem H2: Maximum Subarray Sum (Kadane's Algorithm)

📥 Input

9

-2 1 -3 4 -1 2 1 -5 4

📤 Output

6

🧠 Logic Explanation

Kadane:

maxEnding = max(current_number, maxEnding + current_number)

maxSoFar = max(maxSoFar, maxEnding)

Negative numbers ఉన్నాఫనిచేస్తుంది.

📝 Pseudocode

read N

read array

maxEnding = arr[0]

maxSoFar = arr[0]

for i = 1 to N-1:

  maxEnding = max(arr[i], maxEnding + arr[i])

```
    maxSoFar = max(maxSoFar, maxEnding)


print maxSoFar
```

Java Code

```
import java.io.*;
import java.util.*;
public class KadaneMaxSubarray {
    public static void main(String[] args)throws Exception {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        int n=Integer.parseInt(br.readLine());
        String[] parts=br.readLine().split(" ");
        long[] arr=new long[n];
        for(int i=0;i<n;i++) arr[i]=Long.parseLong(parts[i]);
        long maxEnding=arr[0];
        long maxSoFar=arr[0];
        for(int i=1;i<n;i++){
            maxEnding=Math.max(arr[i],maxEnding+arr[i]);
            maxSoFar=Math.max(maxSoFar,maxEnding);
        }
        System.out.println(maxSoFar);
    }
}
```