✅ EASY — E1: Palindrome Number (string-safe)

📥 Input

12321

📤 Output

YES

🧠 Logic

Number/string యొక్క reverse తయారుచేసి original తో compare చేయండి.

ఎలాంటి leading zeros వస్తే string-approach safe.

📝 Pseudocode

```
read s
if s == reverse(s): print "YES" else print "NO"
```

☕ Java Code (PalindromeNumber.java)

```
import java.io.*;

public class PalindromeNumber {
   public static void main(String[] args) throws Exception {
       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
       String s = br.readLine().trim();
       String rev = new StringBuilder(s).reverse().toString();
       System.out.println(s.equals(rev) ? "YES" : "NO");
   }
}
```

✅ EASY — E2: Rotate Array Right by K

📥 Input

```
5 2
1 2 3 4 5
```

(first line: N K, second line: N integers)

📤 Output

```
4 5 1 2 3
```

🧠 Logic

Rotate right by k = take last k elements to front.

Normalize k = k % n.

Print arr[n-k..n-1] then arr[0..n-k-1].

📝 Pseudocode

```
read n,k
read array a
k = k % n
print a[n-k ... n-1], then a[0 ... n-k-1]
```

☕ Java Code (RotateArrayRight.java)

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

public class RotateArrayRight {
   public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String[] nk = br.readLine().trim().split(" ");
        int n = Integer.parseInt(nk[0]);
        int k = Integer.parseInt(nk[1]) % n;
        String[] parts = br.readLine().trim().split(" ");
        int[] a = new int[n];
        for (int i = 0; i < n; i++) a[i] = Integer.parseInt(parts[i]);
        StringBuilder sb = new StringBuilder();
        for (int i = n - k; i < n; i++) {
            if (i >= 0) sb.append(a[i]).append(" ");
        }
        for (int i = 0; i < n - k; i++) sb.append(a[i]).append(" ");
        System.out.println(sb.toString().trim());
    }
}
```

🔶 MODERATE — M1: Anagram Check (ignore spaces & case)

📥 Input

Listen
Silent

(two lines: two strings)

📤 Output

YES

🧠 Logic

Remove spaces, convert to same case.

Count frequency of letters (or sort) and compare.

📝 Pseudocode

```
read s1, s2
normalize (remove spaces, toLower)
if sorted(s1) == sorted(s2): print YES else NO
```

☕ Java Code (AnagramCheck.java)

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

public class AnagramCheck {
   public static String normalize(String s) {
      return s.replaceAll("\\s+", "").toLowerCase();
   }
   public static void main(String[] args) throws Exception {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      String s1 = br.readLine();
      String s2 = br.readLine();
      s1 = normalize(s1);
      s2 = normalize(s2);
      if (s1.length() != s2.length()) {
         System.out.println("NO");
         return;
      }
      char[] c1 = s1.toCharArray();
      char[] c2 = s2.toCharArray();
      Arrays.sort(c1);
      Arrays.sort(c2);
      System.out.println(Arrays.equals(c1, c2) ? "YES" : "NO");
   }
}
```

💠 MODERATE — M2: Merge Two Sorted Arrays (output merged sorted unique)

📥 Input

```
5 4
1 2 3 5 7
2 3 4 8
```

(first line N M, second line N sorted ints, third line M sorted ints)

📤 Output

```
1 2 3 4 5 7 8
```

🧠 Logic

Classic two-pointer merge; skip duplicates so output contains each value once.

📝 Pseudocode

```
i=0,j=0
while i<n and j<m:
   if a[i] < b[j]: add a[i] if not same as lastAdded; i++
   else if a[i] > b[j]: add b[j] if not same; j++
   else: add one of them; i++; j++
append remaining with duplicate checks
```

☕ Java Code (MergeSortedUnique.java)

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

public class MergeSortedUnique {
   public static void main(String[] args) throws Exception {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      String[] nm = br.readLine().trim().split(" ");
```

```
        int n = Integer.parseInt(nm[0]), m = Integer.parseInt(nm[1]);
        int[] a = Arrays.stream(br.readLine().trim().split("
")).mapToInt(Integer::parseInt).toArray();
        int[] b = Arrays.stream(br.readLine().trim().split("
")).mapToInt(Integer::parseInt).toArray();
        StringBuilder sb = new StringBuilder();
        int i = 0, j = 0;
        Integer last = null;
        while (i < n && j < m) {
            int val;
            if (a[i] < b[j]) { val = a[i++]; }
            else if (a[i] > b[j]) { val = b[j++]; }
            else { val = a[i]; i++; j++; }
            if (last == null || last != val) {
                sb.append(val).append(" ");
                last = val;
            }
        }
        while (i < n) {
            int val = a[i++];
            if (last == null || last != val) { sb.append(val).append(" "); last = val; }
        }
        while (j < m) {
            int val = b[j++];
            if (last == null || last != val) { sb.append(val).append(" "); last = val; }
        }
        System.out.println(sb.toString().trim());
    }
}
```

🔥 HARD — H1: Count Inversions (using Merge Sort)

(Count pairs i < j with a[i] > a[j])

📥 Input

```
5
2 4 1 3 5
```

📤 Output

```
3
```

(Here inversions: (2,1),(4,1),(4,3))

🧠 Logic

Use modified merge sort that counts cross inversions during merge step. O(n log n).

📝 Pseudocode

```
function mergeSortCount(arr, l, r):
    if l >= r: return 0
    mid = (l+r)/2
    cnt = mergeSortCount(arr,l,mid) + mergeSortCount(arr,mid+1,r)
    cnt += mergeAndCount(arr,l,mid,r)
    return cnt
```

☕ Java Code (CountInversions.java)

```java
import java.io.*;
public class CountInversions {
    private static long mergeSortCount(long[] a, long[] temp, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        long cnt = 0;
        cnt += mergeSortCount(a, temp, left, mid);
        cnt += mergeSortCount(a, temp, mid + 1, right);
        cnt += merge(a, temp, left, mid, right);
        return cnt;
    }

    private static long merge(long[] a, long[] temp, int left, int mid, int right) {
        int i = left, j = mid + 1, k = left;
        long cnt = 0;
        while (i <= mid && j <= right) {
            if (a[i] <= a[j]) {
                temp[k++] = a[i++];
            } else {
                // a[i] > a[j] => all remaining a[i..mid] > a[j]
                temp[k++] = a[j++];
                cnt += (mid - i + 1);
            }
        }
        while (i <= mid) temp[k++] = a[i++];
        while (j <= right) temp[k++] = a[j++];
        for (int idx = left; idx <= right; idx++) a[idx] = temp[idx];
        return cnt;
    }

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        int n = Integer.parseInt(br.readLine().trim());
        String[] parts = br.readLine().trim().split(" ");
        long[] a = new long[n];
        for (int i = 0; i < n; i++) a[i] = Long.parseLong(parts[i]);
        long[] temp = new long[n];
        long inversions = mergeSortCount(a, temp, 0, n - 1);
        System.out.println(inversions);
    }
}
```

🔥 HARD — H2: Longest Palindromic Substring Length (expand-around-center)

📥 Input

babad

📤 Output

3

(longest palindrome "bab" or "aba")

🧠 Logic

For each center (i or between i and i+1), expand left/right while chars equal. Keep max length found. O(n^2) but easy to implement.

📝 Pseudocode

```
for i in 0..n-1:
   len1 = expandAround(i,i)
   len2 = expandAround(i,i+1)
   best = max(best, max(len1,len2))
print best
```

☕ Java Code (LongestPalindromicSubstringLength.java)

```java
import java.io.*;

public class LongestPalindromicSubstringLength {
   private static int expand(String s, int left, int right) {
      while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
         left--; right++;
      }
      return right - left - 1; // length
   }

   public static void main(String[] args) throws Exception {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String s = br.readLine();
      if (s == null || s.length() == 0) { System.out.println(0); return; }
```

```
        int best = 1;
        for (int i = 0; i < s.length(); i++) {
            int len1 = expand(s, i, i);
            int len2 = expand(s, i, i + 1);
            best = Math.max(best, Math.max(len1, len2));
        }
        System.out.println(best);
    }
}
```