**13-11-2024 DSA PRACTICE PROBLEMS SET 4**

**1. Kth Smallest Element**

**CODE:**

class Solution {

public static int kthSmallest(int[] arr, int k) {

PriorityQueue<Integer> pq = new PriorityQueue<>((a,b)-> b-a);

int n = arr.length;

for(int i=0;i<n;i++){

pq.offer(arr[i]);

if (pq.size()>k){

pq.poll();

}

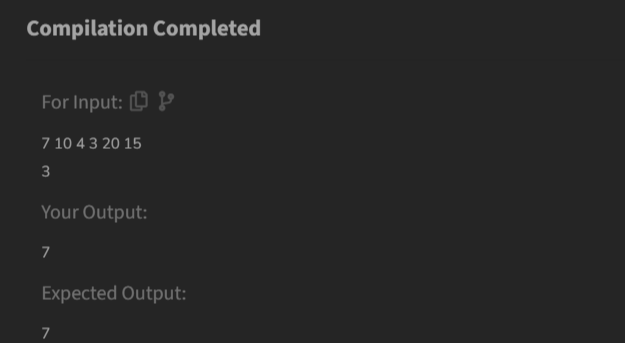
}

return pq.peek();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N \* log(K))**

**2. Minimize The Heights II**

**CODE:**

class Solution {

int getMinDiff(int[] arr, int k) {

Arrays.sort(arr);

int n = arr.length;

int max=0, min=0;

int res = arr[n-1]-arr[0];

int large = arr[n-1]-k, small = arr[0]+k;

for(int i=0;i<n-1;i++){

min = Math.min(small, arr[i+1] - k);

max = Math.max(large, arr[i] + k);

if(min<0) continue;

res = Math.min(res, max-min);

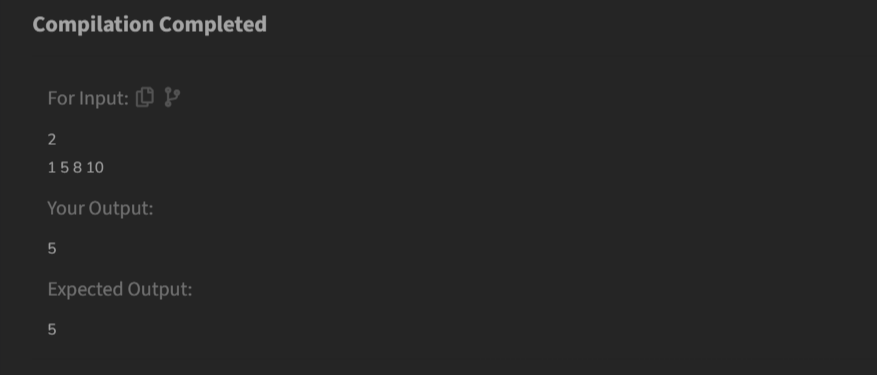
}

return res;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N\*(log(N)))**

**3. Parenthesis Checker**

**CODE:**

class Solution {

static boolean isParenthesisBalanced(String s) {

Stack<Character> st = new Stack<>();

for (int i = 0; i < s.length(); i++) {

if (s.charAt(i) == '(' || s.charAt(i) == '{' || s.charAt(i) == '[') {

st.push(s.charAt(i));

}

else {

if (!st.empty() &&

((st.peek() == '(' && s.charAt(i) == ')') ||

(st.peek() == '{' && s.charAt(i) == '}') ||

(st.peek() == '[' && s.charAt(i) == ']'))) {

st.pop();

}

else {

return false;

}

}

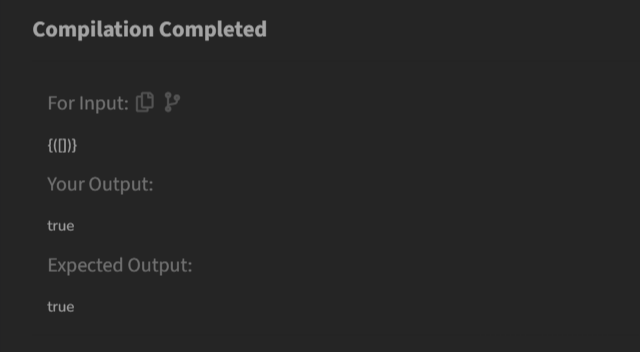
}

return st.empty();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**4. Equilibrium Point**

**CODE:**

class Solution {

public static int equilibriumPoint(int arr[]) {

int n=arr.length;

if (n==1){

return 1;

}

int[] pre = new int[n];

int[] suf = new int[n];

pre[0] = arr[0];

for (int i=1;i<n;i++){

pre[i] = pre[i-1]+arr[i];

}

suf[n-1]=arr[n-1];

for (int i=n-2;i>=0;i--){

suf[i] = suf[i+1]+arr[i];

}

for (int i=0; i<n; i++){

if (pre[i]==suf[i]){

return i+1;

}

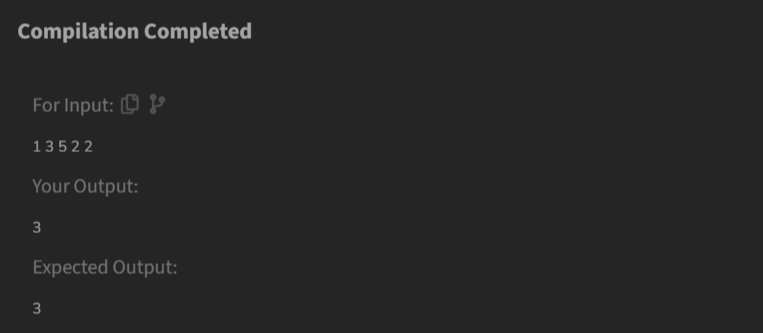
}

return -1;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**5. Binary Search**

**CODE:**

class Solution {

public int binarysearch(int[] arr, int k) {

int n = arr.length;

int low=0;

int high=n-1;

while(low<=high){

int mid = (low+high)/2;

if (arr[mid]==k){

return mid;

}

if (arr[mid]<k){

low=mid+1;

}

else{

high=mid-1;

}

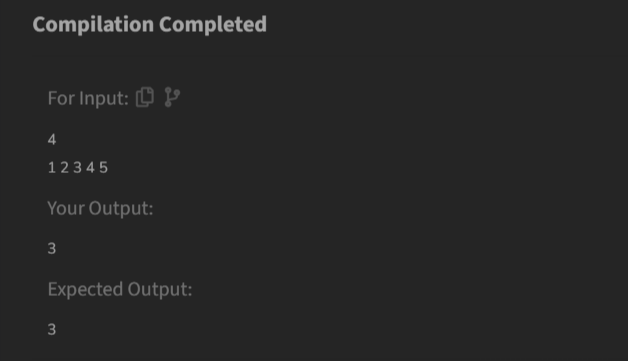
}

return -1;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(log(N))**

**6. Next Larger Element**

**CODE:**

class Solution {

public ArrayList<Integer> nextLargerElement(int[] arr) {

int next, i, j;

int n = arr.length;

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

for (i = 0; i < n; i++) {

next = -1;

for (j = i + 1; j < n; j++) {

if (arr[i] < arr[j]) {

next = arr[j];

break;

}

}

ans.add(next);

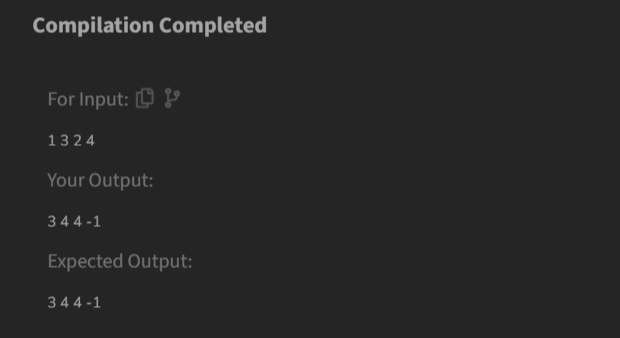
}

return ans;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N^2)**

**7. Union Of Two Arrays with Duplicate Elements**

**CODE:**

class Solution {

public static int findUnion(int a[], int b[]) {

Stack<Integer> st = new Stack<>();

for (int i : a){

if(!st.contains(i)){

st.push(i);

}

}

for (int j : b){

if(!st.contains(j)){

st.push(j);

}

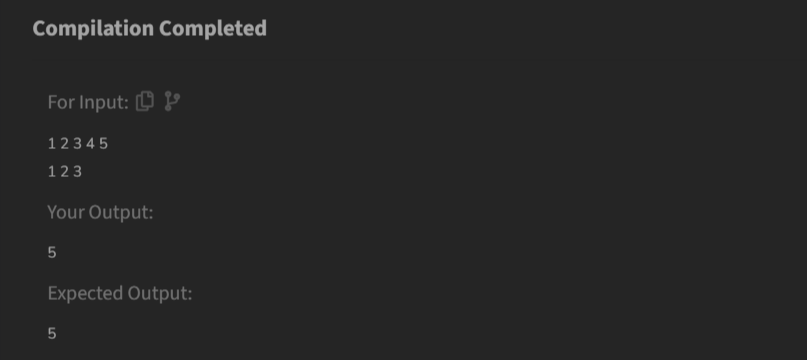
}

return st.size();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**