SARAVANAN BS-AI&DS-DSA-PRACTICE-4

1. K-th smallest element

import java.util.PriorityQueue;

class Solution {

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

PriorityQueue<Integer> minHeap = new PriorityQueue<>();

for (int num : arr) {

minHeap.add(num);

}

int ans = -1;

while (k > 0) {

ans = minHeap.poll();

k--;

}

return ans;

}

}

Result:

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3  
Output: 7

Input: arr[] = [2, 3, 1, 20, 15], k = 4   
Output: 15

Time complexity: O(n log(n))

2. Minimize the heights-II

class Solution {  
 int getMinDiff(int[] arr, int k) {  
 // code here  
 int n =arr.length;  
 if( n == 1){  
 return 0;  
 }  
 Arrays.sort(arr);  
   
 int ans = arr[n-1] - arr[0];  
 int min = arr[0] + k;  
 int max = arr[n-1] - k;  
   
 for(int i=0; i<n-1; i++){  
 int min\_ele = Math.min(min, arr[i+1]-k);  
 int max\_ele = Math.max(max, arr[i]+k);  
 if(min\_ele < 0)  
 continue;  
 ans = Math.min(ans, max\_ele - min\_ele);  
 }  
 return ans;  
 }  
}

Result :

Input: k = 2, arr[] = {1, 5, 8, 10}  
Output: 5

Input: k = 3, arr[] = {3, 9, 12, 16, 20}  
Output: 11

Time Complexity: O(n log(n))

3. Parenthesis checker

class Solution {

// Function to check if brackets are balanced or not.

static boolean isParenthesisBalanced(String s) {

StringBuilder sb = new StringBuilder();

for (char c : s.toCharArray()) {

if (c == '(' || c == '{' || c == '[') {

sb.append(c);

}

else if (c == ')' || c == '}' || c == ']') {

if (sb.length() == 0) {

return false;

}

char last = sb.charAt(sb.length() - 1);

if ((c == ')' && last != '(') ||

(c == '}' && last != '{') ||

(c == ']' && last != '[')) {

return false;

}

sb.setLength(sb.length() - 1);

}

}

return sb.length() == 0;

}

}

Result:  
Input: s = "{([])}"  
Output: true

Input: s = "()"  
Output: true

Input: s = "([]"  
Output: false

Time complexity: O(N)

4. Equilibrium point

class Solution {  
 // Function to find equilibrium point in the array.  
 public static int equilibriumPoint(int arr[]) {  
 int rsum = 0;  
 for(int i:arr) rsum += i;  
 int lsum = 0;  
 for(int i=0;i<arr.length;i++){  
 if(lsum == (rsum - lsum-arr[i])) return i+1;  
 lsum += arr[i];  
 }  
 return -1;  
 }  
}

Result:

Input: arr[] = [1, 3, 5, 2, 2]  
Output: 3

Input: arr[] = [1]  
Output: 1

Input: arr[] = [1, 2, 3]  
Output: -1

Time Complexity: O(n)

5.Binary Search

class Solution {

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

// Code Here

int low = 0;

int high = arr.length-1;

while(low <= high){

int mid = (low + high) / 2;

if(arr[mid] == k){

return mid;

}

else if(arr[mid] > k){

high = mid - 1;

}

else{

low = mid + 1;

}

}

return -1;

}

}

Result:

Input: arr[] = [1, 2, 3, 4, 5], k = 4  
Output: 3

Input: arr[] = [11, 22, 33, 44, 55], k = 445  
Output: -1

Time complexity: O(n log(N))

6. Next greater element

class Solution {

// Function to find the next greater element for each element of the array.

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

int n = arr.length;

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

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

boolean flag = true;

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

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

flag = false;

answer.add(arr[j]);

break;

}

}

if(flag){

answer.add(-1);

}

}

answer.add(-1);

return answer;

}

}

Result:

Input: arr[] = [1, 3, 2, 4]  
Output: [3, 4, 4, -1]

Input: arr[] = [6, 8, 0, 1, 3]  
Output: [8, -1, 1, 3, -1]

Input: arr[] = [10, 20, 30, 50]  
Output: [20, 30, 50, -1]

Input: arr[] = [50, 40, 30, 10]  
Output: [-1, -1, -1, -1]

Time complexity: O(n\*n)

7. Union of two arrays with duplicate elements

class Solution {

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

HashMap<Integer, Integer> hash = new HashMap<>();

for(int i: a){

hash.put(i, 0);

}

for(int i: b){

hash.put(i, 0);

}

return hash.size();

}

}

Result:

Input: a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]  
Output: 5

Input: a[] = [85, 25, 1, 32, 54, 6], b[] = [85, 2]   
Output: 7

Input: a[] = [1, 2, 1, 1, 2], b[] = [2, 2, 1, 2, 1]   
Output: 2

Time complexity: O(n)