**SANJAY\_M – CSE – DSA – PRACTICE – 2**

**Q1**. **Kth smallest element**

Given an array **arr[]**of **N** distinct elements and a number **K,** where **K** is smaller than the size of the array. Find the **K’th** smallest element in the given array.

**Input**: arr[] = {7, 10, 4, 3, 20, 15}, K = 3

**Output**: 7

**Input**: arr[] = {7, 10, 4, 3, 20, 15}, K = 4

**Output**: 10

CODE:

import java.util.Arrays;

import java.util.Collections;

class kthsmallest{

public static int kthSmallest(Integer[] arr, int K)

{

Arrays.sort(arr);

return arr[K - 1];

}

public static void main(String[] args)

{

Integer arr[] = new Integer[] { 7, 10, 4, 3, 20, 15};

int K = 3;

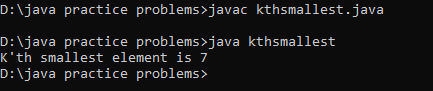
System.out.print("K'th smallest element is "

+ kthSmallest(arr, K));

}

}

OUTPUT:



Time Complexity: O(n log n)

**Q2. Minimize the heights 2**

Given an array arr[] denoting heights of N towers and a positive integer K.

For each tower, you must perform exactly one of the following operations exactly once.

* Increase the height of the tower by K
* Decrease the height of the tower by K

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.   
Note: It is compulsory to increase or decrease the height by K for each tower. After the operation, the resultant array should not contain any negative integers.

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

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

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

Output: 11

Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

CODE:

import java.util.Arrays;

class mindiff {

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

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;

}

public static void main(String[] args) {

mindiff md = new mindiff();

int[] arr = {1, 5, 8, 10};

int k = 2;

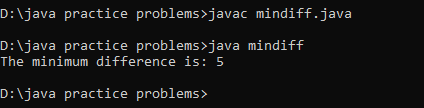
int result = md.getMinDiff(arr, k);

System.out.println("The minimum difference is: " + result);

}

}

OUTPUT:



Time Complexity: O(n log n)

**Q3. Paranthesis checker**

You are given a string **s** representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

**Input**: s = "{([])}"

**Output**: true

**Explanation**:   
- In this expression, every opening bracket has a corresponding closing bracket.  
- The first bracket { is closed by }, the second opening bracket ( is closed by ), and the third opening bracket [ is closed by ].  
- As all brackets are properly paired and closed in the correct order, the expression is considered balanced.

**Input**: s = "()"

**Output**: true

**Explanation**:   
- This expression contains only one type of bracket, the parentheses ( and ).  
- The opening bracket ( is matched with its corresponding closing bracket ).  
- Since they form a complete pair, the expression is balanced.

CODE:

import java.util.\*;

class paranthesischecker {

static boolean isParenthesisBalanced(String s) {

Map<Character, Character> matching = new HashMap<>();

matching.put('(', ')');

matching.put('[', ']');

matching.put('{', '}');

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

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

if (matching.containsKey(c)) {

stack.push(c);

} else {

if (stack.empty()) {

return false;

}

char previousOpening = stack.pop();

if (matching.get(previousOpening) != c) {

return false;

}

}

}

return stack.empty();

}

public static void main(String[] args) {

String s = "{[()]}";

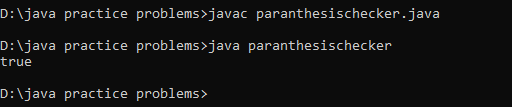
boolean result = isParenthesisBalanced(s);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(n)

**Q4. Equilibrium point**

Given an array**arr**of non-negative numbers. The task is to find the first **equilibrium point** in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements beforethat index is the same as the sumof elements afterit.

**Note:** Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

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

**Output:** 3

**Explanation:** The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

**Input:** arr[] = [1]

**Output:** 1

**Explanation:** Since there's only one element hence it's only the equilibrium point.

CODE:

class equilibriumpoint {

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;

}

public static void main(String[] args) {

int[] arr = {1, 3, 5, 2, 2};

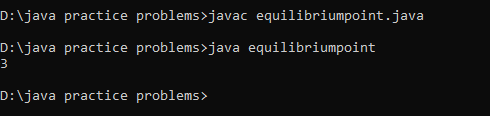
int result = equilibriumPoint(arr);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(n)

**Q5. Binary Search**

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Explanation: 9 exists in nums and its index is 4

Input: nums = [-1,0,3,5,9,12], target = 2

Output: -1

Explanation: 2 does not exist in nums so return -1

CODE:

class binarysearch {

public int search(int[] nums, int target) {

int start = 0;

int end = nums.length - 1;

while (start <= end) {

int mid = (start + end) / 2;

if (nums[mid] == target) {

return mid;

} else if (nums[mid] > target) {

end = mid - 1;

} else {

start = mid + 1;

}

}

return -1;

}

public static void main(String[] args) {

binarysearch bs = new binarysearch();

int[] nums = {-1,0,3,5,9,12};

int target = 9;

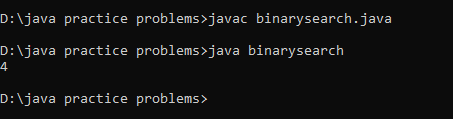
int result = bs.search(nums, target);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(log n)

**Q6. Next greater element**

Given an array arr[ ] of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

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

Output: [3, 4, 4, -1]

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

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

Output: [8, -1, 1, 3, -1]

Explanation: The next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1, for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

CODE:

import java.util.ArrayList;

class nextlarger {

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

int n = arr.length;

ArrayList<Integer> list = new ArrayList<>(n);

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

list.add(-1);

}

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

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

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

list.set(i, arr[j]);

break;

}

}

}

return list;

}

public static void main(String[] args) {

nextlarger nl = new nextlarger();

int[] arr = {1, 3, 2, 4};

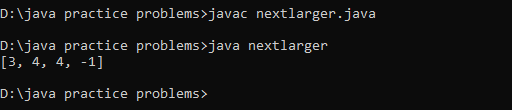
ArrayList<Integer> result = nl.nextLargerElement(arr);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(n^2)

**Q7. Union of 2 arrays with duplicate elements**

Given two arrays a[] and b[], the task is to find the number of elements in the union between these two arrays.The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.

Note: Elements are not necessarily distinct.

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

Output: 5

Explanation: 1, 2, 3, 4 and 5 are the elements which comes in the union setof both arrays. So count is 5.

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

Output: 7

Explanation: 85, 25, 1, 32, 54, 6, and 2 are the elements which comes in the union set of both arrays. So count is 7.

CODE:

class Union {

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

Set<Integer> set = new HashSet<>();

for (int i = 0; i < a.length; i++) set.add(a[i]);

for (int i = 0; i < b.length; i++) set.add(b[i]);

return set.size();

}

public static void main(String[] args) {

int[] a = {1, 2, 3, 4, 5};

int[] b = {1, 2, 3};

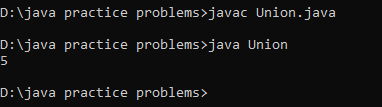
int result = findUnion(a, b);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(n+m)