# DSA Programming Practice – 2,3,4 – KAVYA U – CSBS( III yr) – 2026 BATCH – 22CB024

## 1. 0-1 Knapsack Problem :

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Examples:

Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}

Output: 3

Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.

Input: N = 3, W = 3, profit[] = {1, 2, 3}, weight[] = {4, 5, 6}

Output: 0

### Code and Output:

```
class knapsack {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static int knapSack(int W, int wt[], int val[], int n)
    {
        if (n == 0 || W == 0)
            return 0;
        if (wt[n - 1] > W)
            return knapSack(W, wt, val, n - 1);
        else
            return Math.max(knapSack(W, wt, val, n - 1),
             val[n - 1] + knapSack(W - wt[n-1], wt, val, n-1));
    }

    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String args[])
    {
        int profit[] = new int[] { 60, 100, 120 };
        int weight[] = new int[] { 10, 20, 30 };
        int W = 50;
        int n = profit.length;
        System.out.println(knapSack(W, weight, profit, n));
    }
}
```

```
OUTPUT  PROBLEMS 33  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

PS D:\SDE JAVA DSA> cd "d:\SDE JAVA DSA\" ; if ($?) { javac knapsack.java } ; i
220
PS D:\SDE JAVA DSA>
```

**Time Complexity:** O(2^N)

## 2. Floor in a Sorted Array :

Given a sorted array arr[] (with unique elements) and an integer k, find the index (0-based) of the largest element in arr[] that is less than / equal to k. This element is called the "floor" of k. If such an element does not exist, return -1.

Examples

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 0

Output: -1

Explanation: No element less than 0 is found. So output is -1.

Input: arr[] = [1, 2, 8, 10, 11, 12, 19], k = 5

Output: 1

Explanation: Largest Number less than 5 is 2 , whose index is 1.

Input: arr[] = [1, 2, 8], k = 1

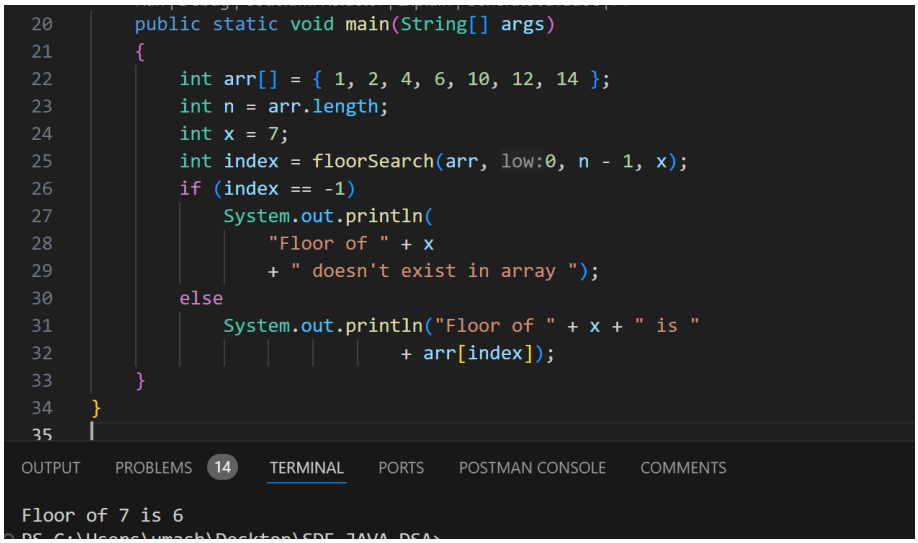
Output: 0

Explanation: Largest Number less than or equal to  1 is 1 , whose index is 0.

**Code and Output:**

```java
import java.io.*;

Codeium: Refactor | Explain
class floorinarray {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static int floorSearch(int arr[], int low, int high,
                           int x)
    {
        if (low > high)
            return -1;
        if (x >= arr[high])
            return high;
        int mid = (low + high) / 2;
        if (arr[mid] == x)
            return mid;
        if (mid > 0 && arr[mid - 1] <= x && x < arr[mid])
            return mid - 1;
        if (x < arr[mid])
            return floorSearch(arr, low, mid - 1, x);
        return floorSearch(arr, mid + 1, high, x);
    }
```



```
    public static void main(String[] args)
    {
        int arr[] = { 1, 2, 4, 6, 10, 12, 14 };
        int n = arr.length;
        int x = 7;
        int index = floorSearch(arr, low:0, n - 1, x);
        if (index == -1)
            System.out.println(
                "Floor of " + x
                + " doesn't exist in array ");
        else
            System.out.println("Floor of " + x + " is "
                               + arr[index]);
    }
}
```

OUTPUT PROBLEMS 14 TERMINAL PORTS POSTMAN CONSOLE COMMENTS

```
Floor of 7 is 6
```

**Time Complexity:** O(logN)

## 3. Check Equal Arrays :

Given two given arrays of equal length, the task is to find if given arrays are equal or not. Two arrays are said to be equal if both of them contain the same set of elements and in the same order.

Examples:

Input : arr1[] = {1, 2, 5, 4, 0}; arr2[] = {1, 2, 5, 4, 0};

Output : Yes

Input : arr1[] = {1, 2, 5, 4, 0, 2}; arr2[] = {2, 4, 5, 0};

Output : No

Input : arr1[] = {1, 7, 7}; arr2[] = {7, 7, 1};

Output : No

**Code and Output:**

```java
class arrayequal {
    public static boolean areEqual(int arr1[], int arr2[])
    {
        int N = arr1.length;
        int M = arr2.length;
        if (N != M)
            return false;
        Map<Integer, Integer> map
            = new HashMap<Integer, Integer>();
        int count = 0;
        for (int i = 0; i < N; i++) {
            if (map.get(arr1[i]) == null)
                map.put(arr1[i], value:1);
            else {
                count = map.get(arr1[i]);
                count++;
                map.put(arr1[i], count);
            }
        }
        for (int i = 0; i < N; i++) {
            if (!map.containsKey(arr2[i]))
                return false;
            if (map.get(arr2[i]) == 0)
                return false;
            count = map.get(arr2[i]);
            --count;
            map.put(arr2[i], count);
        }
        return true;
```

```java
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc |
    public static void main(String[] args)
    {
        int arr1[] = { 3, 5, 2, 5, 2 };
        int arr2[] = { 2, 3, 5, 5, 2 };
        if (areEqual(arr1, arr2))
            System.out.println(x:"Yes");
        else
            System.out.println(x:"No");
    }
}
```

```
OUTPUT  PROBLEMS 13  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

Yes
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(N)

## 4. Palindrome linked list :

Given a singly linked list. The task is to check if the given linked list is palindrome or not.

Examples:

Input: head: 1->2->1->1->2->1

Output: true

Explanation: The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.

Input: head: 1->2->3->4

Output: false

Explanation: The given linked list is 1->2->3->4, which is not a palindrome and Hence, the output is false.

## Code and Output:

```java
class Node {
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}
Codeium: Refactor | Explain
class palindromeLL {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static boolean isPalindrome(Node head) {
        Node currNode = head;
        Stack<Integer> s = new Stack<>();
        while (currNode != null) {
            s.push(currNode.data);
            currNode = currNode.next;
        }
        while (head != null) {
            int c = s.pop();
            if (head.data != c) {
                return false;
            }
            head = head.next;
        }
        return true;
    }
```

```java
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print(s:"Enter the number of nodes in the linked list: ");
        int n = scanner.nextInt();
        System.out.println(x:"Enter the elements of the linked list:");
        Node head = null, tail = null;
        for (int i = 0; i < n; i++) {
            int value = scanner.nextInt();
            Node newNode = new Node(value);
            if (head == null) {
                head = newNode;
                tail = newNode;
            } else {
                tail.next = newNode;
                tail = newNode;
            }
        }
        boolean result = isPalindrome(head);
        if (result)
            System.out.println(x:"The linked list is a palindrome.");
        else
            System.out.println(x:"The linked list is not a palindrome.");

        scanner.close();
    }
}
```

```
OUTPUT  PROBLEMS 14  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

Enter the elements of the linked list:
1 2 3 2 1
The linked list is a palindrome.
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(N)

## 5. Balanced Tree Check :

Given a binary tree, find if it is height balanced or not.  A tree is height balanced if difference between heights of left and right subtrees is not more than one for all nodes of tree.

Examples:

Input:

```
    1
  /
 2
  \
   3
```

Output: 0

Explanation: The max difference in height of left subtree and right subtree is 2, which is greater than 1. Hence unbalanced

Input:

```
     10
   /   \
  20   30
 /  \
40   60
```

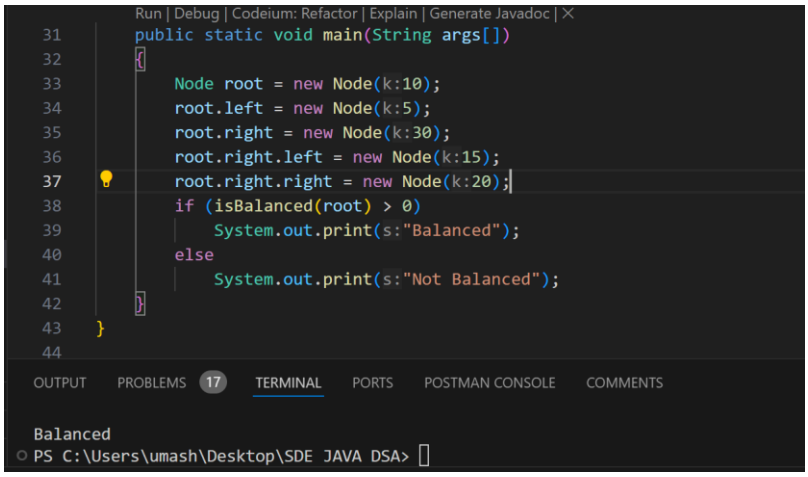
Output: 1

Explanation: The max difference in height of left subtree and right subtree is 1. Hence balanced.

### Code and Output:

```java
class Node {
    int key;
    Node left;
    Node right;
    Node(int k)
    {
        key = k;
        left = right = null;
    }
}
Codeium: Refactor | Explain
class heightbalanceBT {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int isBalanced(Node root)
    {
        if (root == null)
            return 0;
        int lh = isBalanced(root.left);
        if (lh == -1)
            return -1;
        int rh = isBalanced(root.right);
        if (rh == -1)
            return -1;

        if (Math.abs(lh - rh) > 1)
            return -1;
        else
            return Math.max(lh, rh) + 1;
    }
```




**Time Complexity:** O(N)

## 6. Triplet Sum in Array :

Given an array arr[] of size n and an integer sum. Find if there's a triplet in the array which sums up to the given integer sum.

Examples:

Input: arr = {12, 3, 4, 1, 6, 9}, sum = 24;

Output: 12, 3, 9

Explanation: There is a triplet (12, 3 and 9) present

in the array whose sum is 24.

Input: arr = {1, 2, 3, 4, 5}, sum = 9

Output: 5, 3, 1

Explanation: There is a triplet (5, 3 and 1) present

in the array whose sum is 9.

Input: arr = {2, 10, 12, 4, 8}, sum = 9

Output: No Triplet

Explanation: We do not print in this case and return false.

**Code and Output:**

```java
import java.util.Arrays;
Codeium: Refactor | Explain
public class tripletsum {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static boolean find3Numbers(int[] arr, int sum)
    {
        int n = arr.length;
        Arrays.sort(arr);
        for (int i = 0; i < n - 2; i++) {
            int l = i + 1;
            int r = n - 1;
            while (l < r) {
                int curr_sum = arr[i] + arr[l] + arr[r];
                if (curr_sum == sum) {
                    System.out.println(
                        "Triplet is " + arr[i] + ", "
                        + arr[l] + ", " + arr[r]);
                    return true;
                }
                else if (curr_sum < sum) {
                    l++;
                }
                else {
                    r--;
                }
            }
        }
        return false;
    }
    public static void main(String[] args)
    {
        int[] arr = { 1, 4, 45, 6, 10, 8 };
        int sum = 22;
        find3Numbers(arr, sum);
    }
}

```

```
OUTPUT  PROBLEMS 16  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

Triplet is 4, 8, 10
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(N^2)

## 7. Find the row with maximum number of 1s:

Given a binary 2D array, where each row is sorted. Find the row with the maximum number of 1s.

Examples: Input matrix : 0 1 1 1

0 0 1 1

1 1 1 1

0 0 0 0

Output: 2

Explanation: Row = 2 has maximum number of 1s, that is 4.

Input matrix : 0 0 1 1

0 1 1 1

0 0 1 1

0 0 0 0

Output: 1

Explanation: Row = 1 has maximum number of 1s, that is 3.

**Code and Output:**

```java
public class maxonesinmat {
    static final int R = 4;
    static final int C = 4;
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int rowWithMax1s(int[][] mat)
    {
        int maxRow = -1, row = 0, col = C - 1;
        while (row < R && col >= 0) {
            if (mat[row][col] == 0) {
                row++;
            }
            else {
                maxRow = row;
                col--;
            }
        }
        return maxRow;
    }
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args)
    {
        int[][] mat = { { 0, 0, 0, 1 },
                        { 0, 1, 1, 1 },
                        { 1, 1, 1, 1 },
                        { 0, 0, 0, 0 } };

        System.out.println(
            "Index of row with maximum 1s is "
            + rowWithMax1s(mat));
    }
```

```
OUTPUT  PROBLEMS 27  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

Index of row with maximum 1s is 2
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(M+N)

## 8. Longest Consecutive Subsequence:

Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Examples:

Input: arr[] = {1, 9, 3, 10, 4, 20, 2}

Output: 4

Explanation: The subsequence 1, 3, 4, 2 is the longest subsequence of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}

Output: 5

Explanation: The subsequence 36, 35, 33, 34, 32 is the longest subsequence of consecutive elements.

**Code and Output:**

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

Codeium: Refactor | Explain
class longestconsecutivesubsequence {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static int findLongestConseqSubseq(int arr[], int n)
    {
        HashSet<Integer> S = new HashSet<Integer>();
        int ans = 0;
        for (int i = 0; i < n; ++i)
            S.add(arr[i]);
        for (int i = 0; i < n; ++i) {
            if (!S.contains(arr[i] - 1)) {
                int j = arr[i];
                while (S.contains(j))
                    j++;
                if (ans < j - arr[i])
                    ans = j - arr[i];
            }
        }
        return ans;
    }
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String args[])
    {
        int arr[] = { 1, 9, 3, 10, 4, 20, 2 };
        int n = arr.length;
        System.out.println(
            "Length of the Longest consecutive subsequence is "
            + findLongestConseqSubseq(arr, n));
    }
}
```

```
OUTPUT  PROBLEMS 18  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

Length of the Longest consecutive subsequence is 4
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(N)

## 9. Rat in a Maze Problem :

Consider a rat placed at (0, 0) in a square matrix of order N * N. It has to reach the destination at (N – 1, N – 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are ‘U'(up), ‘D'(down), ‘L’ (left), ‘R’ (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it. Return the list of paths in lexicographically increasing order.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell.

Examples:

Input:

maze = [

[1, 0, 0, 0],

[1, 1, 0, 1],

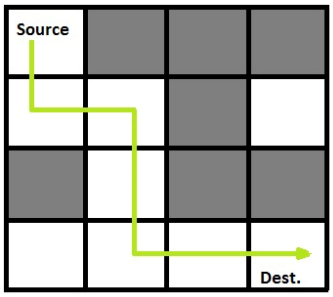
[1, 1, 0, 0],

[0, 1, 1, 1]

]

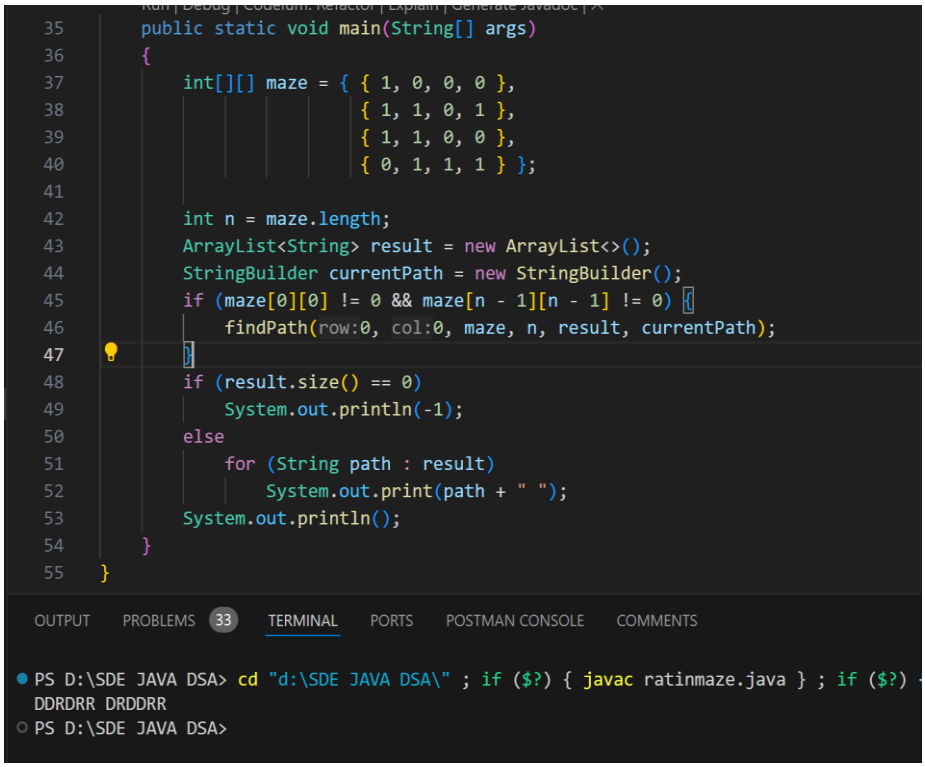
Output: DRDDRR

Explanation:




## Code and Output:

```java
public class ratinmaze {
    static String direction = "DLRU";
    static int[] dr = { 1, 0, 0, -1 };
    static int[] dc = { 0, -1, 1, 0 };
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static boolean isValid(int row, int col, int n,
                           int[][] maze)
    {
        return row >= 0 && col >= 0 && row < n && col < n
            && maze[row][col] == 1;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    static void findPath(int row, int col, int[][] maze,
                         int n, ArrayList<String> ans,
                         StringBuilder currentPath)
    {
        if (row == n - 1 && col == n - 1) {
            ans.add(currentPath.toString());
            return;
        }
        maze[row][col] = 0;
        for (int i = 0; i < 4; i++) {
            int nextrow = row + dr[i];
            int nextcol = col + dc[i];
            if (isValid(nextrow, nextcol, n, maze)) {
                currentPath.append(direction.charAt(i));
                findPath(nextrow, nextcol, maze, n, ans,
                         currentPath);
                currentPath.deleteCharAt(
                    currentPath.length() - 1);
            }
        }
        maze[row][col] = 1;
    }
```



```
    public static void main(String[] args)
    {
        int[][] maze = { { 1, 0, 0, 0 },
                         { 1, 1, 0, 1 },
                         { 1, 1, 0, 0 },
                         { 0, 1, 1, 1 } };

        int n = maze.length;
        ArrayList<String> result = new ArrayList<>();
        StringBuilder currentPath = new StringBuilder();
        if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {
            findPath(row:0, col:0, maze, n, result, currentPath);
        }
        if (result.size() == 0)
            System.out.println(-1);
        else
            for (String path : result)
                System.out.print(path + " ");
        System.out.println();
    }
}
```

```
OUTPUT  PROBLEMS 33  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

PS D:\SDE JAVA DSA> cd "d:\SDE JAVA DSA\" ; if ($?) { javac ratinmaze.java } ; if ($?)
DDRDRR DRDDRR
PS D:\SDE JAVA DSA>
```

**Time Complexity:** O(3^(m*n))

## 10. K'th Smallest Element in Unsorted Array:

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.

Examples:

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 and Output:**

```java
import java.util.Arrays;

Codeium: Refactor | Explain
public class kthsmallest {
    Codeium: Refactor | Explain | Generate Javadoc | X
    static int kthSmallest(int[] arr, int n, int k)
    {
        int max_element = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > max_element) {
                max_element = arr[i];
            }
        }
        int[] freq = new int[max_element + 1];
        Arrays.fill(freq, val:0);
        for (int i = 0; i < n; i++) {
            freq[arr[i]]++;
        }
        int count = 0;
        for (int i = 0; i <= max_element; i++) {
            if (freq[i] != 0) {
                count += freq[i];
                if (count >= k) {
                    return i;
                }
            }
        }
        return -1;
    }
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc | X
    public static void main(String[] args)
    {
        int[] arr = { 12, 3, 5, 7, 19 };
        int n = arr.length;
        int k = 2;
        System.out.println("The " + k + "th smallest element is " + kthSmallest(arr, n, k));
    }
}
```

```
OUTPUT  PROBLEMS 26  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS

The 2th smallest element is 5
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(N + max_element)

## 11. Minimize the maximum difference between the heights :

Given the heights of n towers and a positive integer k, increase or decrease the height of all towers by k (only once). After modifications, the task is to find the minimum difference between the heights of the tallest and the shortest tower.

Examples:

Input: arr[] = {12, 6, 4, 15, 17, 10}, k = 6

Output: 8

Explanation: Update arr[] as {12 – 6, 6 + 6, 4 + 6, 15 – 6, 17 – 6, 10 – 6} = {6, 12, 10, 9, 11, 4}. Now, the minimum difference is 12 – 4 = 8.

Input: arr[] = {12, 6, 4, 15, 17, 10}, k = 3

Output: 8

Explanation: Update arr[] as {1 + 3, 5 + 3, 10 – 3, 15 – 3} = {4, 8, 7, 12}. Now, the minimum difference is 8.

**Code and Output :**

```java
import java.util.Arrays;
class minmaxdiff {
    static int getMinDiff(int[] arr, int k) {
        int n = arr.length;
        Arrays.sort(arr);
        int res = arr[n - 1] - arr[0];
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] - k < 0)
                continue;
            int minH = Math.min(arr[0] + k, arr[i] - k);
            int maxH = Math.max(arr[i - 1] + k, arr[n - 1] - k);
            res = Math.min(res, maxH - minH);
        }
        return res;
    }
    public static void main(String[] args) {
        int k = 6;
        int[] arr = {12, 6, 4, 15, 17, 10};

        int ans = getMinDiff(arr, k);
        System.out.println(ans);
    }
}
```

```
OUTPUT  PROBLEMS 26  TERMINAL  PORTS  POSTMAN CONSOLE  COMMENTS
8
PS C:\Users\umash\Desktop\SDE JAVA DSA>
```

**Time Complexity:** O(logN)

## 12. Equilibrium index of an array :

Given an array arr[] of size n, return an equilibrium index (if any) or -1 if no equilibrium index exists. The equilibrium index of an array is an index such that the sum of elements at lower indexes equals the sum of elements at higher indexes.

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

Examples:

Input: arr[] = {-7, 1, 5, 2, -4, 3, 0}

Output: 4

Explanation: In 1-based indexing, 4 is an equilibrium index, because: arr[1] + arr[2] + arr[3] = arr[5] + arr[6] + arr[7]

Input: arr[] = {1, 2, 3}

Output: -1

Explanation: There is no equilibrium index in the array.

**Code and Output:**


**Time Complexity:** O(N)

## 13. Union of two arrays with duplicate elements:

We are given two sorted arrays a[] and b[] and the task is to return union of both the arrays in sorted order. Union of two arrays is an array having all distinct element

s that are present in either array. The input arrays may contain duplicates.

Examples:

Input: a[] = {1, 1, 2, 2, 2, 4}, b[] = {2, 2, 4, 4}

Output: {1, 2, 4}

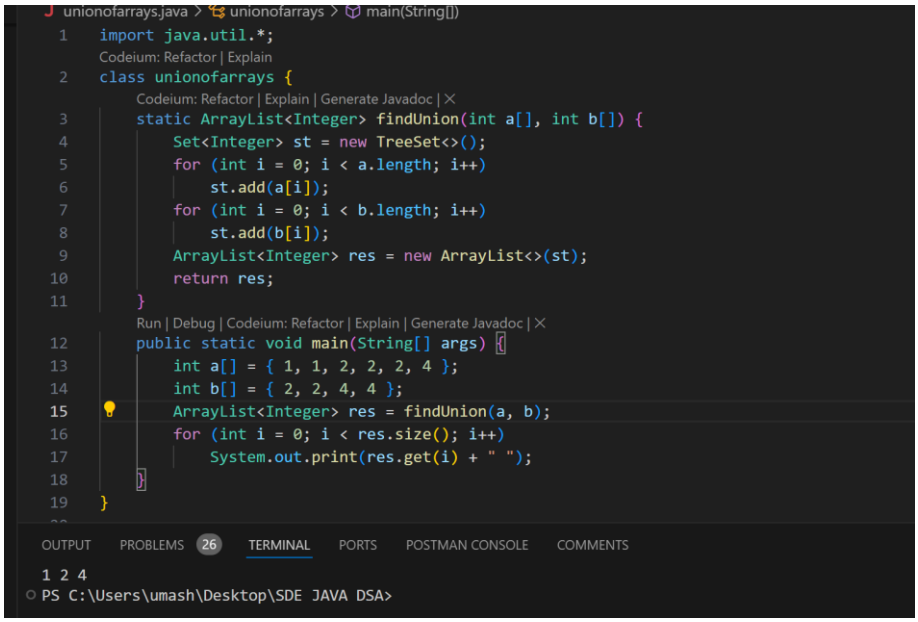
Explanation: 1, 2 and 4 are the distinct elements present in either array.

Input: a[] = {3, 5, 10, 10, 10, 15, 15, 20}, b[] = {5, 10, 10, 15, 30}

Output: {3, 5, 10, 15, 20, 30}

Explanation: 3, 5, 10, 15, 20 and 30 are the distinct elements present in either array.

**Code and Output:**




**Time Complexity:** O((n + m) * (log (n + m)))

Todo questions: buy&sell stock III&IV, egg drop, gas station