Experiment-4

Student Name: Kiyansh **UID:** 22BCS13103
Branch: BE-CSE **Section/Group:** 22BCS-IOT-640-B
Semester: 6th **Date of Performance:** 04/02/2025
Subject Name: AP Lab-2 **Subject Code:** 22CSH-359

1. **Aim:** Divide and Conquer

2. **Problem Statements:**

Problem 1.1: A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return the skyline formed by these buildings collectively.

Problem 1.2: Given an integer array nums, return the number of reverse pairs in the array.

A reverse pair is a pair (i, j) where:

- 0 <= i < j < nums.length and
- nums[i] > 2 * nums[j]

Problem 1.3: Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Problem 1.4: A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

3. Implementation/Code:

Problem 1.1

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<List<Integer>> result = new ArrayList<>();
        List<int[]> events = new ArrayList<>();

        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]});
            events.add(new int[]{b[1], b[2]});
        }

        events.sort((a, b) -> {
            if (a[0] != b[0]) return a[0] - b[0];
            return a[1] - b[1];
        });

        PriorityQueue<Integer> maxHeap = new
PriorityQueue<>(Collections.reverseOrder());
        maxHeap.add(0);
        int prevMaxHeight = 0;
        for (int[] event : events) {
            if (event[1] < 0) {
                maxHeap.add(-event[1]);
            } else {
                maxHeap.remove(event[1]);
            }

            int currentMaxHeight = maxHeap.peek();
            if (currentMaxHeight != prevMaxHeight) {
                result.add(Arrays.asList(event[0], currentMaxHeight));
                prevMaxHeight = currentMaxHeight;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[][] buildings1 = {{2,9,10},{3,7,15},{5,12,12},{15,20,10},{19,24,8}};
        System.out.println(solution.getSkyline(buildings1));

        int[][] buildings2 = {{0,2,3},{2,5,3}};
        System.out.println(solution.getSkyline(buildings2));
    }
}
```

Problem 1.2:

```
import java.util.*;

class Solution {
    public int reversePairs(int[] nums) {
        if (nums == null || nums.length == 0) return 0;
        return mergeSort(nums, 0, nums.length - 1);
    }

    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) return 0;

        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

        count += countPairs(nums, left, mid, right);

        merge(nums, left, mid, right);

        return count;
    }
```

```
private int countPairs(int[] nums, int left, int mid, int right) {
    int count = 0, j = mid + 1;
    for (int i = left; i <= mid; i++) {
        while (j <= right && nums[i] > 2L * nums[j]) {
            j++;
        }
        count += (j - (mid + 1));
    }
    return count;
}

private void merge(int[] nums, int left, int mid, int right) {
    int[] temp = new int[right - left + 1];
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        if (nums[i] <= nums[j]) {
            temp[k++] = nums[i++];
        } else {
            temp[k++] = nums[j++];
        }
    }
```

```
        while (i <= mid) temp[k++] = nums[i++];
        while (j <= right) temp[k++] = nums[j++];

        System.arraycopy(temp, 0, nums, left, temp.length);
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        int[] nums1 = {1,3,2,3,1};
        System.out.println(solution.reversePairs(nums1));

        int[] nums2 = {2,4,3,5,1};
        System.out.println(solution.reversePairs(nums2));
    }
}
```

Problem 1.3:

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) return
false;

        int rows = matrix.length;
        int cols = matrix[0].length;

        int row = 0, col = cols - 1; // Start from the top-right corner

        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--; // Move left
            } else {
                row++; // Move down
            }
        }

        return false; // Target not found
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        int[][] matrix1 = {
            {1, 4, 7, 11, 15},
            {2, 5, 8, 12, 19},
            {3, 6, 9, 16, 22},
            {10, 13, 14, 17, 24},
            {18, 21, 23, 26, 30}
        };

        System.out.println(solution.searchMatrix(matrix1, 5));  // Output: true
        System.out.println(solution.searchMatrix(matrix1, 20)); // Output: false
    }
}
```

Problem 1.4:

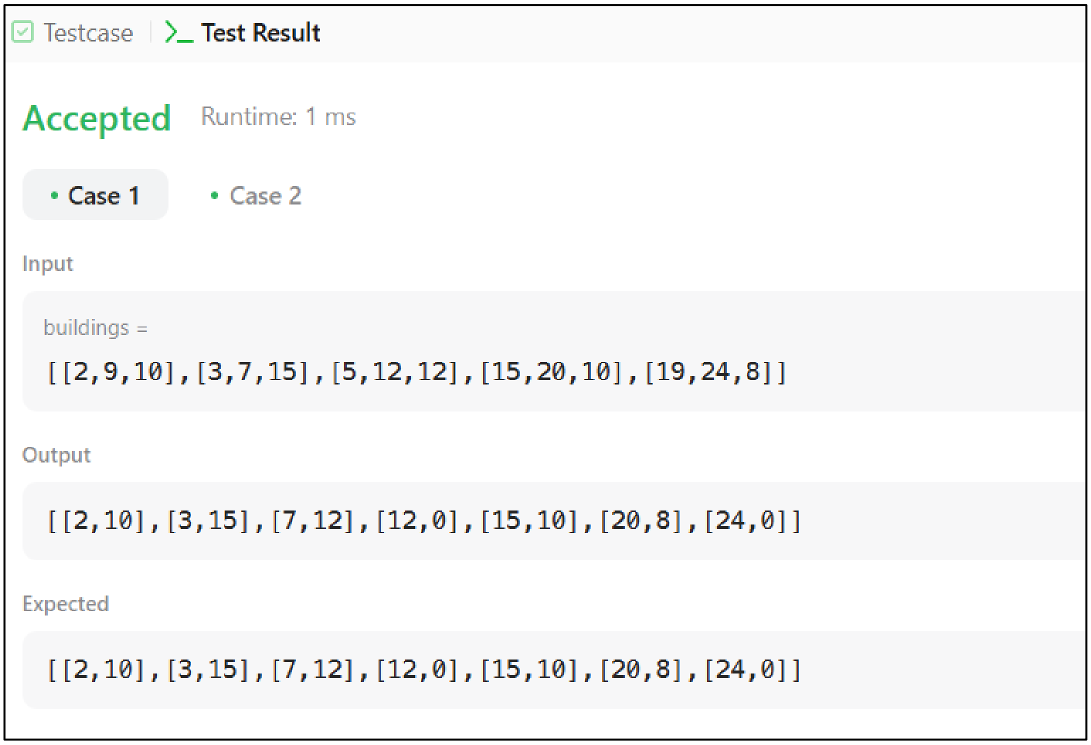
```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";

        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (s.indexOf(Character.toUpperCase(c)) == -1 ||
s.indexOf(Character.toLowerCase(c)) == -1) {
                String left = longestNiceSubstring(s.substring(0, i));
                String right = longestNiceSubstring(s.substring(i + 1));
                return left.length() >= right.length() ? left : right;
            }
        }
        return s;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        System.out.println(solution.longestNiceSubstring("YazaAay"));
        System.out.println(solution.longestNiceSubstring("Bb"));
        System.out.println(solution.longestNiceSubstring("c"));
    }
}
```

4. Output:



Accepted Runtime: 1 ms

Input

buildings =

```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

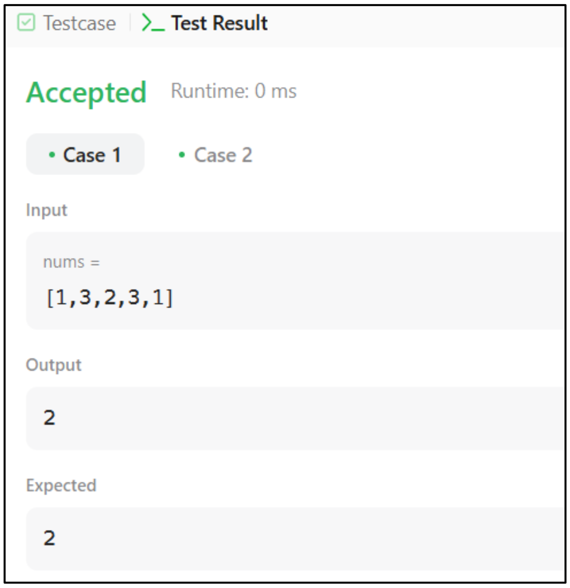
Output

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Expected

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

(Fig. 1- Problem 1.1 Output)



Accepted Runtime: 0 ms

Input

nums =

```
[1,3,2,3,1]
```

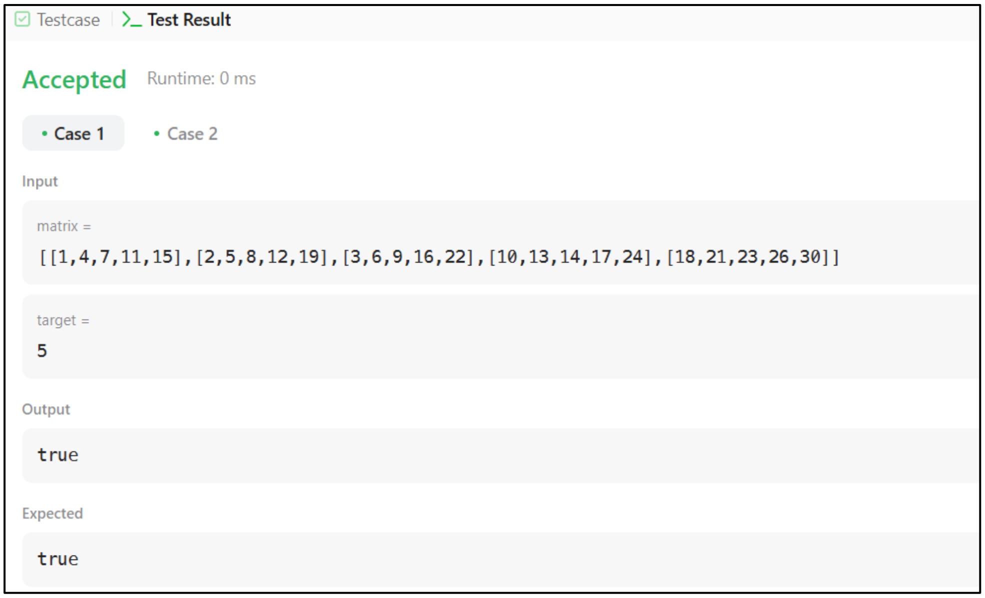
Output

```
2
```

Expected

```
2
```

(Fig. 2- Problem 1.2 Output)



Accepted Runtime: 0 ms

Input

```
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =
5
```

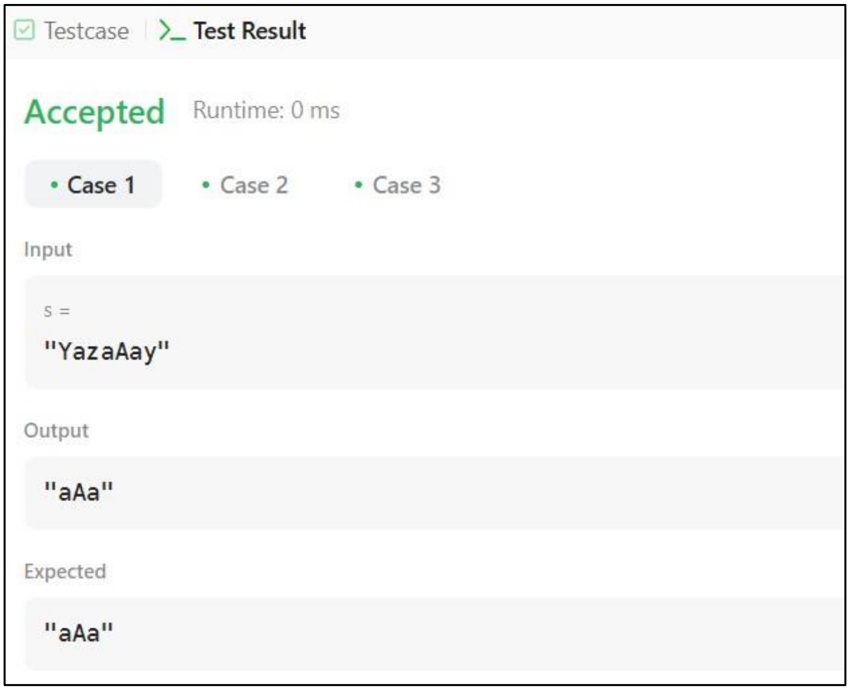
Output

```
true
```

Expected

```
true
```

(Fig. 3- Problem 1.3 Output)



Accepted Runtime: 0 ms

Input

```
s =
"YazaAay"
```

Output

```
"aAa"
```

Expected

```
"aAa"
```

(Fig. 4- Problem 1.4 Output)

5. Learning Outcome:

1. Learn how to use priority queues (heaps) to efficiently track dynamic height changes in skyline problems.
2. Learn how merge sort can be extended to solve problems beyond sorting, such as counting important reverse pairs.
3. Learn how to efficiently traverse a sorted 2D matrix using a strategic approach from the top-right corner.
4. Learn how to divide and conquer a string problem by recursively identifying substrings that satisfy given conditions.