Assignment 4

Submitted by:

Sahil(22BCS15237)

608B

1. Longest Nice Substring:

```java
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;
    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted Mar 06, 2025	Java	1 ms	41.7 MB	

2. Reverse Bits:

```java
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int result = 0;
        for (int i = 0; i < 32; i++) {
            result <<= 1; // Shift result left by 1 bit
            result |= (n & 1); // Extract the last bit of n and add it to result
            n >>= 1; // Shift n right by 1 bit
        }
        return result;
    }

    }

```

	Status	Language	Runtime	Memory	Notes
1	Accepted 15 minutes ago	Java	0 ms	42.1 MB	

3. Number of 1 Bits:

```java
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1); // Check if the last bit is set
            n >>= 1; // Shift n right by 1
        }
        return count;

    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted 15 minutes ago	Java	0 ms	40.6 MB	

4. Maximum Subarray:

```java
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0]; // Initialize maxSum to the first element
        int currentSum = nums[0]; // Initialize currentSum to the first element

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]); // Decide whether to start a new subarray or
            maxSum = Math.max(maxSum, currentSum); // Update maxSum if currentSum is greater
        }

        return maxSum;
    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted 15 minutes ago	Java	1 ms	56.8 MB	

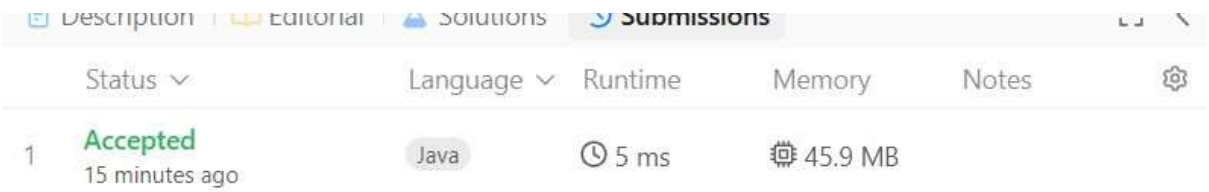
5. Search a 2D Matrix II:

```java
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; // Target found
            } else if (matrix[row][col] > target) {
                col--; // Move left
            } else {
                row++; // Move down
            }
        }

        return false;
    }
}
```



6. Super Pow:

```java
import java.util.*;

class Solution {
    private static final int MOD = 1337;
    public int superPow(int a, int[] b) {
        a %= MOD;
        int result = 1;
        for (int digit : b) {
            result = (powerMod(result, 10) * powerMod(a, digit)) % MOD;
        }
        return result;
    }

    public static int powerMod(int a, int b) {
        int result = 1;
        a %= MOD;
        for (int i = 0; i < b; i++) {
            result = (result * a) % MOD;
        }
        return result;
    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted 13 minutes ago	Java	4 ms	44.6 MB	

7. Beautiful Array:

```java
class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1);

        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();

            // Generate odd numbers
            for (int num : result) {
                int odd = num * 2 - 1;
                if (odd <= n) temp.add(odd);
            }

            // Generate even numbers
            for (int num : result) {
                int even = num * 2;
                if (even <= n) temp.add(even);
            }

            result = temp;
        }

        // Convert List to array
        int[] resArray = new int[n];
        for (int i = 0; i < n; i++) {
            resArray[i] = result.get(i);
        }
        return resArray;
```

	Status	Language	Runtime	Memory	Notes
1	Accepted 15 minutes ago	Java	4 ms	42.4 MB	

8.

The Skyline Problem:

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

        // Step 1: Convert buildings into events
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2], b[1]}); // Start event (-height for priority order)
            events.add(new int[]{b[1], b[2], 0}); // End event
        }

        // Step 2: Sort events
        Collections.sort(events, (a, b) -> {
            if (a[0] != b[0]) return Integer.compare(a[0], b[0]); // Sort by x-coordinate
            if (a[1] != b[1]) return Integer.compare(a[1], b[1]); // Sort by height (start before end)
            return Integer.compare(a[2], b[2]); // Sort by right coordinate
        });

        // Step 3: Sweep Line Algorithm with TreeMap
        List<List<Integer>> result = new ArrayList<>();
        TreeMap<Integer, Integer> heightMap = new TreeMap<>(Collections.reverseOrder());
        heightMap.put(0, 1); // Ground level
        int prevHeight = 0;

        for (int[] event : events) {
            int x = event[0], height = event[1];

            if (height < 0) { // Start of a building
                heightMap.put(-height, heightMap.getOrDefault(-height, 0) + 1);
            } else { // End of a building
                if (heightMap.get(height) == 1) {
                    heightMap.remove(height);
                } else {
                    heightMap.put(height, heightMap.get(height) - 1);
                }
            }

            int currHeight = heightMap.firstKey();
            if (currHeight != prevHeight) { // Height changed, add key point
                result.add(Arrays.asList(x, currHeight));
                prevHeight = currHeight;
            }
        }

        return result;
    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted 13 minutes ago	Java	36 ms	50.8 MB	

9.

Reverse Pairs:

10.

```java
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 reverse pairs across the two halves
        count += countReversePairs(nums, left, mid, right);

        // Merge the two halves
        merge(nums, left, mid, right);

        return count;
    }

    private int countReversePairs(int[] nums, int left, int mid, int right) {
        int count = 0;
        int j = mid + 1;

        for (int i = left; i <= mid; i++) {
            while (j <= right && (long) 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);
    }
}
```

11.

Longest Increasing Subsequence II:

```java
class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        TreeMap<Integer, Integer> map = new TreeMap<>();
        int maxLength = 0;

        for (int num : nums) {
            // Find the max dp[j] where nums[j] is in range [num-k, num-1]
            Integer floorKey = map.floorKey(num - 1);
            int bestPrev = 0;

            while (floorKey != null && floorKey >= num - k) {
                bestPrev = Math.max(bestPrev, map.get(floorKey));
                floorKey = map.lowerKey(floorKey);
            }

            // Compute dp[i] for current number
            int currLength = bestPrev + 1;
            map.put(num, currLength);

            // Maintain max length
            maxLength = Math.max(maxLength, currLength);
        }

        return maxLength;
    }
}
```