

Assignment 4

Submitted by:

Muskan(22BCS16260)

608B

1. Longest Nice Substring:

```
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:

```

1 public class Solution {
2     // you need treat n as an unsigned value
3     public int reverseBits(int n) {
4         int result = 0;
5         for (int i = 0; i < 32; i++) {
6             result <<= 1; // Shift result left by 1 bit
7             result |= (n & 1); // Extract the last bit of n and add it to result
8             n >>= 1; // Shift n right by 1 bit
9         }
10        return result;
11    }
12 }
13 }
14

```

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

3. Number of 1 Bits:

```

1 class Solution {
2     public int hammingWeight(int n) {
3         int count = 0;
4         while (n != 0) {
5             count += (n & 1); // Check if the last bit is set
6             n >>= 1; // Shift n right by 1
7         }
8         return count;
9     }
10 }
11 }

```

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

4. Maximum Subarray:

```

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:

```

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;
    }
}

```

Description		Editorial	Solutions	Submissions			
Status	Language		Runtime	Memory	Notes		
1	Accepted	15 minutes ago	Java	5 ms	45.9 MB		

6. Super Pow:

```

1  import java.util.*;
2
3  class Solution {
4      private static final int MOD = 1337;
5      public int superPow(int a, int[] b) {
6          a %= MOD;
7          int result = 1;
8          for (int digit : b) {
9              result = (powerMod(result, 10) * powerMod(a, digit)) % MOD;
10         }
11         return result;
12     }
13
14     public static int powerMod(int a, int b) {
15         int result = 1;
16         a %= MOD;
17         for (int i = 0; i < b; i++) {
18             result = (result * a) % MOD;
19         }
20         return result;
21     }
22 }

```

	Status ▾	Language ▾	Runtime	Memory	Notes	⚙
1	Accepted 13 minutes ago	Java	🕒 4 ms	💾 44.6 MB		

7. Beautiful Array:

```

1  class Solution {
2      public int[] beautifulArray(int n) {
3          List<Integer> result = new ArrayList<>();
4          result.add(1);
5
6          while (result.size() < n) {
7              List<Integer> temp = new ArrayList<>();
8
9              // Generate odd numbers
10             for (int num : result) {
11                 int odd = num * 2 - 1;
12                 if (odd <= n) temp.add(odd);
13             }
14
15             // Generate even numbers
16             for (int num : result) {
17                 int even = num * 2;
18                 if (even <= n) temp.add(even);
19             }
20
21             result = temp;
22         }
23
24         // Convert List to array
25         int[] resArray = new int[n];
26         for (int i = 0; i < n; i++) {
27             resArray[i] = result.get(i);
28         }
29         return resArray;

```

Status ▾

Language ▾

Runtime

Memory

Notes



1 **Accepted**
15 minutes ago

Java

🕒 4 ms

🧠 42.4 MB

8. The Skyline Problem:

```
1 class Solution {
2     public List<List<Integer>> getSkyline(int[][] buildings) {
3         List<int[]> events = new ArrayList<>();
4
5         // Step 1: Convert buildings into events
6         for (int[] b : buildings) {
7             events.add(new int[]{b[0], -b[2], b[1]}); // Start event (-height for priority order)
8             events.add(new int[]{b[1], b[2], 0}); // End event
9         }
10
11        // Step 2: Sort events
12        Collections.sort(events, (a, b) -> {
13            if (a[0] != b[0]) return Integer.compare(a[0], b[0]); // Sort by x-coordinate
14            if (a[1] != b[1]) return Integer.compare(a[1], b[1]); // Sort by height (start before end)
15            return Integer.compare(a[2], b[2]); // Sort by right coordinate
16        });
17
18        // Step 3: Sweep Line Algorithm with TreeMap
19        List<List<Integer>> result = new ArrayList<>();
20        TreeMap<Integer, Integer> heightMap = new TreeMap<>(Collections.reverseOrder());
21        heightMap.put(0, 1); // Ground level
22        int prevHeight = 0;
23
24        for (int[] event : events) {
25            int x = event[0], height = event[1];
26
27            if (height < 0) { // Start of a building
28                heightMap.put(-height, heightMap.getOrDefault(-height, 0) + 1);
29            } else { // End of a building
30                if (heightMap.get(height) == 1) {
31                    heightMap.remove(height);
32                } else {
33                    heightMap.put(height, heightMap.get(height) - 1);
34                }
35            }
36
37            int currHeight = heightMap.firstKey();
38            if (currHeight != prevHeight) { // Height changed, add key point
39                result.add(Arrays.asList(x, currHeight));
40                prevHeight = currHeight;
41            }
42        }
43
44        return result;
45    }
46 }
```

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

9. Reverse Pairs:

```
1  class Solution {
2      public int reversePairs(int[] nums) {
3          if (nums == null || nums.length == 0) return 0;
4          return mergeSort(nums, 0, nums.length - 1);
5      }
6
7      private int mergeSort(int[] nums, int left, int right) {
8          if (left >= right) return 0;
9
10         int mid = left + (right - left) / 2;
11         int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
12
13         // Count reverse pairs across the two halves
14         count += countReversePairs(nums, left, mid, right);
15
16         // Merge the two halves
17         merge(nums, left, mid, right);
18
19         return count;
20     }
21
22     private int countReversePairs(int[] nums, int left, int mid, int right) {
23         int count = 0;
24         int j = mid + 1;
25
26         for (int i = left; i <= mid; i++) {
27             while (j <= right && (long) nums[i] > 2L * nums[j]) {
28                 j++;
29             }
30             count += (j - (mid + 1));
31         }
32
33         return count;
34     }
35
36     private void merge(int[] nums, int left, int mid, int right) {
37         int[] temp = new int[right - left + 1];
38         int i = left, j = mid + 1, k = 0;
39
40         while (i <= mid && j <= right) {
41             if (nums[i] <= nums[j]) {
42                 temp[k++] = nums[i++];
43             } else {
44                 temp[k++] = nums[j++];
45             }
46         }
47
48         while (i <= mid) temp[k++] = nums[i++];
49         while (j <= right) temp[k++] = nums[j++];
50
51         System.arraycopy(temp, 0, nums, left, temp.length);
52     }
53 }
```


10. Longest Increasing Subsequence II:

```
1 class Solution {
2     public int lengthOfLIS(int[] nums, int k) {
3         TreeMap<Integer, Integer> map = new TreeMap<>();
4         int maxLength = 0;
5
6         for (int num : nums) {
7             // Find the max dp[j] where nums[j] is in range [num-k, num-1]
8             Integer floorKey = map.floorKey(num - 1);
9             int bestPrev = 0;
10
11             while (floorKey != null && floorKey >= num - k) {
12                 bestPrev = Math.max(bestPrev, map.get(floorKey));
13                 floorKey = map.lowerKey(floorKey);
14             }
15
16             // Compute dp[i] for current number
17             int currLength = bestPrev + 1;
18             map.put(num, currLength);
19
20             // Maintain max length
21             maxLength = Math.max(maxLength, currLength);
22         }
23
24         return maxLength;
25     }
26 }
```