Experiment 4

Student Name: Amar Kumar Mandal
UID: 22BET10379
Branch: IT
Section/Group: 22BET_IOT-702/B
Semester: 6th
Date of Performance: 14/02/25
Subject Name: Advance Programming-II
Subject Code: 22ITP-367

Problem: 1.4.1: Longest Nice Substring

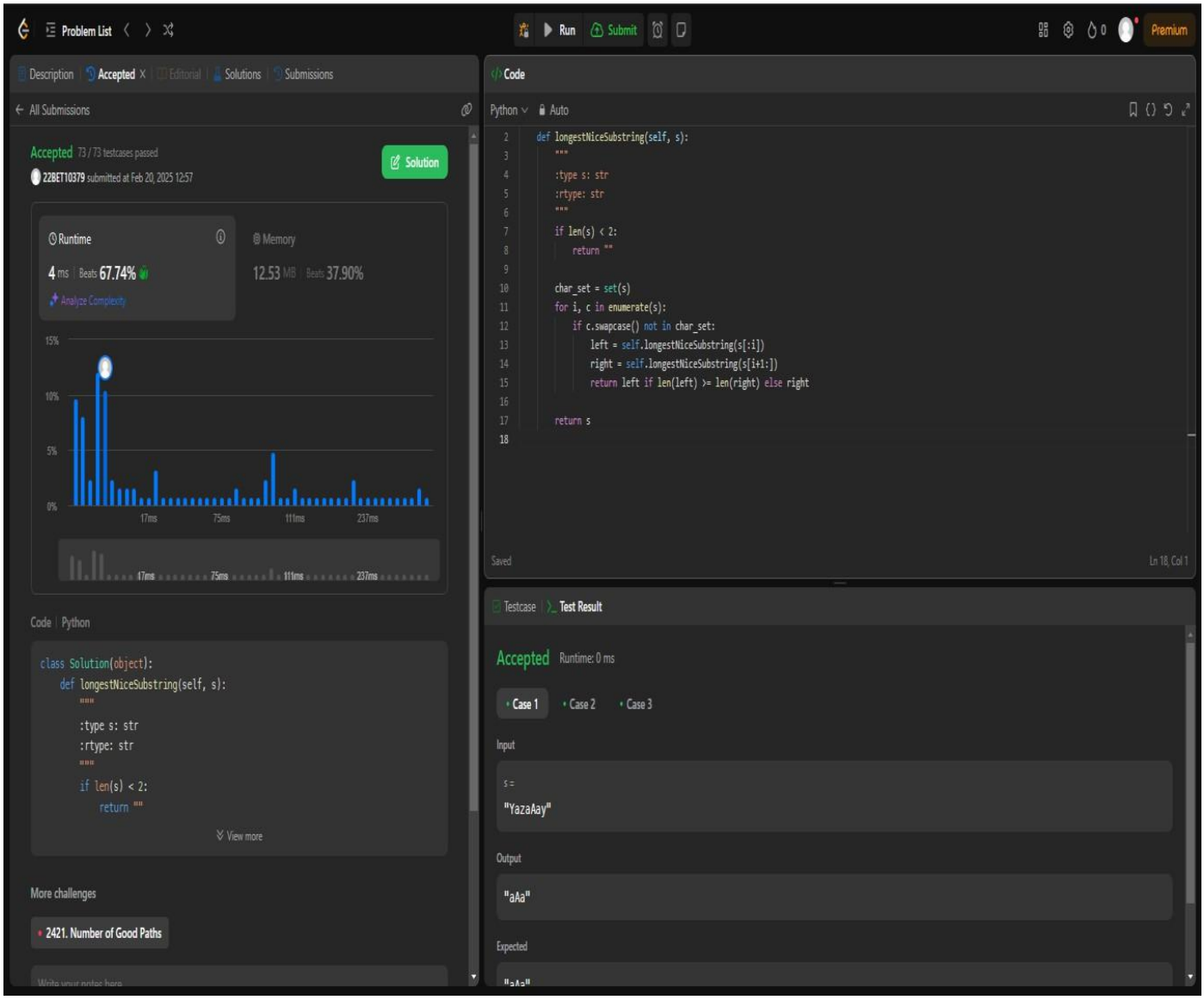
Problem Statement: A string s is considered **nice** if, for every character in the string, the character's uppercase and lowercase forms both exist in the string.

1. **Objective:** Find the longest contiguous substring where every character has both its uppercase and lowercase counterpart present.

2. **Code:**

```
class Solution(object):
    def longestNiceSubstring(self, s):
        """
        :type s: str
        :rtype: str
        """
        if len(s) < 2:
            return ""

        char_set = set(s)
        for i, c in enumerate(s):
            if c.swapcase() not in char_set:
                left = self.longestNiceSubstring(s[:i])
                right = self.longestNiceSubstring(s[i+1:])
                return left if len(left) >= len(right) else right

        return s
```

3. Result:


Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 73 / 73 testcases passed
22BET10379 submitted at Feb 20, 2025 12:57
Solution
Runtime
4 ms | Beats 67.74%
Analyze Complexity
Memory
12.53 MB | Beats 37.90%
15%
10%
5%
0%
17ms
75ms
111ms
237ms
Code | Python
class Solution(object):
def longestNiceSubstring(self, s):
"""
:type s: str
:rtype: str
"""
if len(s) < 2:
return ""
View more
More challenges
2421. Number of Good Paths
Code
Python
Auto
def longestNiceSubstring(self, s):
"""
:type s: str
:rtype: str
"""
if len(s) < 2:
return ""
char_set = set(s)
for i, c in enumerate(s):
if c.swapcase() not in char_set:
left = self.longestNiceSubstring(s[:i])
right = self.longestNiceSubstring(s[i+1:])
return left if len(left) >= len(right) else right
return s
Saved
Ln 18, Col 1
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
s =
"YazaAay"
Output
"aAa"
Expected
"aAa"

Problem 1.4.2: Reverse Bits

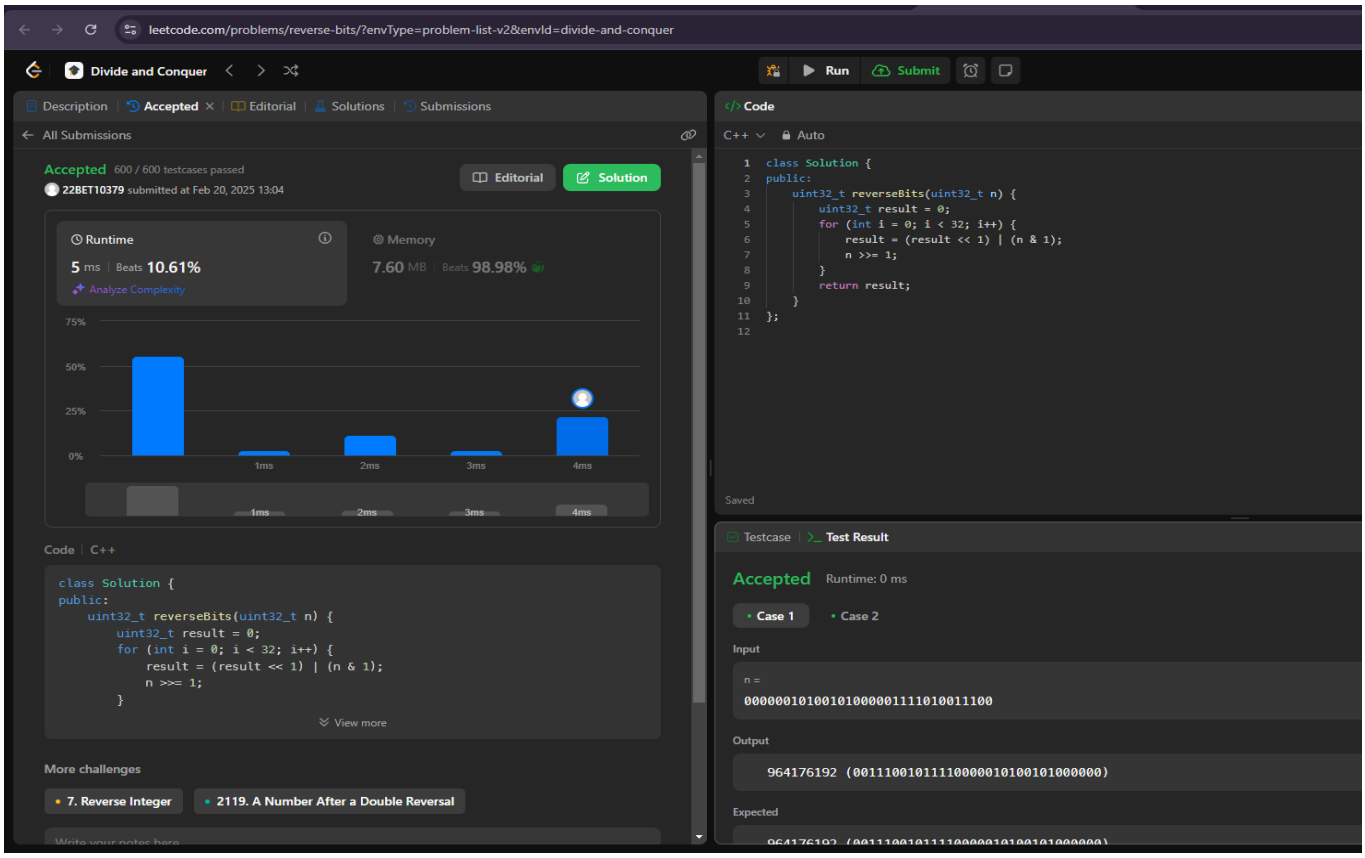
Problem Statement: You are given a 32-bit unsigned integer n. Your task is to reverse the bits of n and return the result as an unsigned integer.

1. Objective:
Reverse the order of bits in the given 32-bit unsigned integer.

2. Code:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```

3. Result:



Problem 1.4.3: Number of 1 bits

Problem Statement: You are given a 32-bit unsigned integer n. Your task is to return the number of '1' bits it has, also known as the **Hamming Weight**.

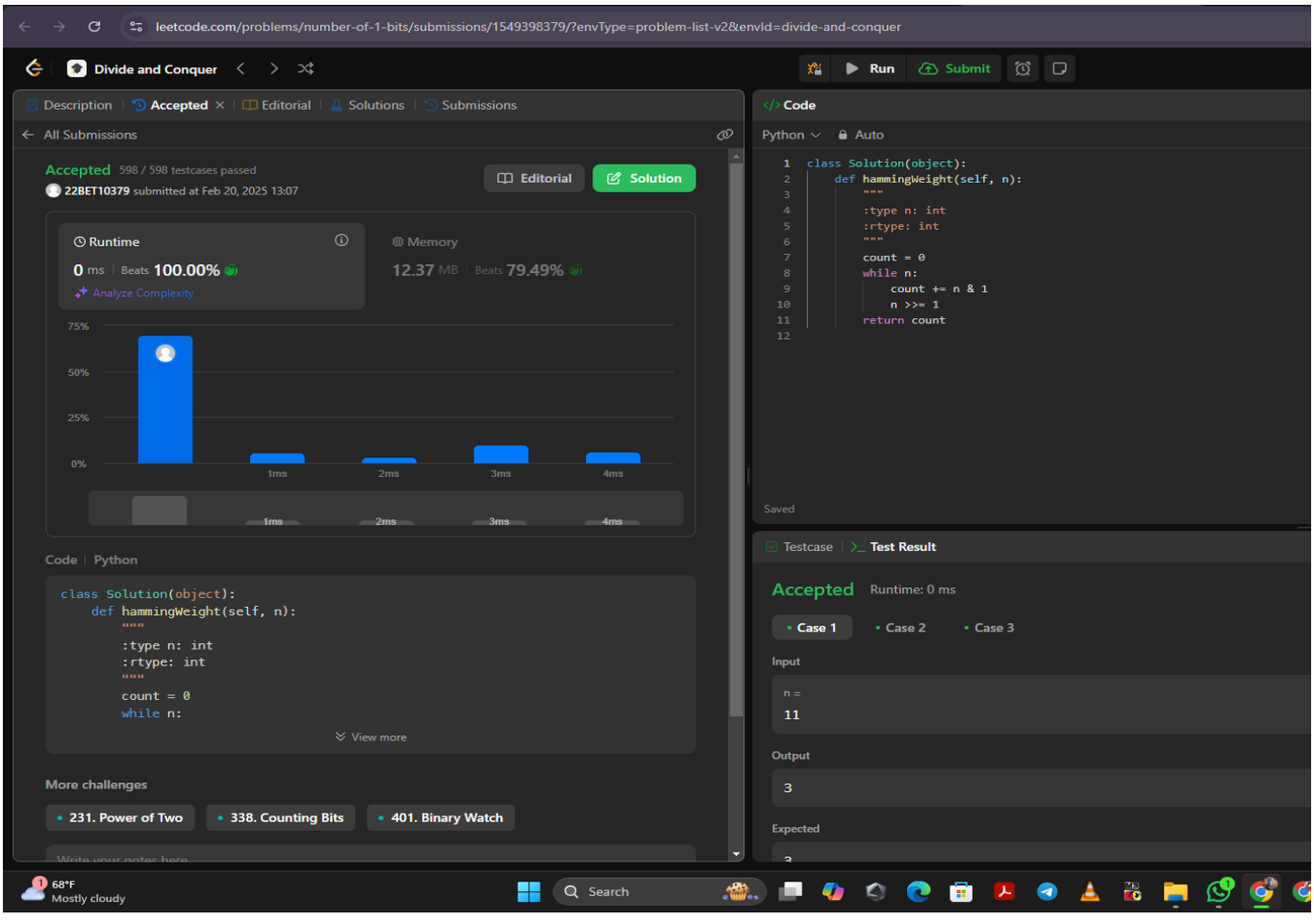
1. Objective:

Count the number of '1' bits in the 32-bit binary representation of n.

2. Code:

```
class Solution(object):
    def hammingWeight(self, n):
        """
        :type n: int
        :rtype: int
        """
        count = 0
        while n:
            count += n & 1
            n >>= 1
        return count
```

3. Result:



Problem 1.3.4: Max Subarray

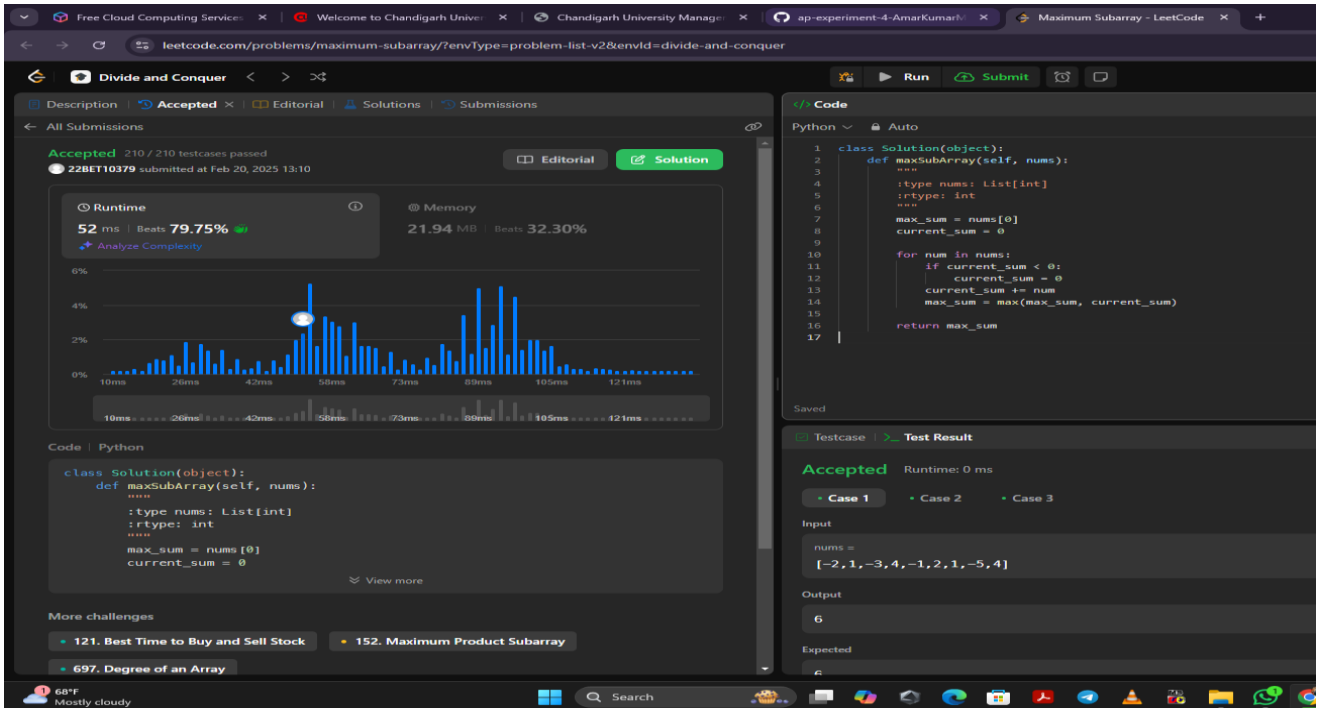
Problem Statement: Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

1. **Objective:** Identify the contiguous subarray within the given array that has the maximum sum.

2. **Code:**

```
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        max_sum = nums[0]
        current_sum = 0

        for num in nums:
            if current_sum < 0:
                current_sum = 0
            current_sum += num
            max_sum = max(max_sum, current_sum)

        return max_sum
```

3. **Result:**



Problem 1.4.5: Beautiful Array

Problem Statement: An array nums of length n is beautiful if:

nums is a permutation of the integers in the range [1, n].

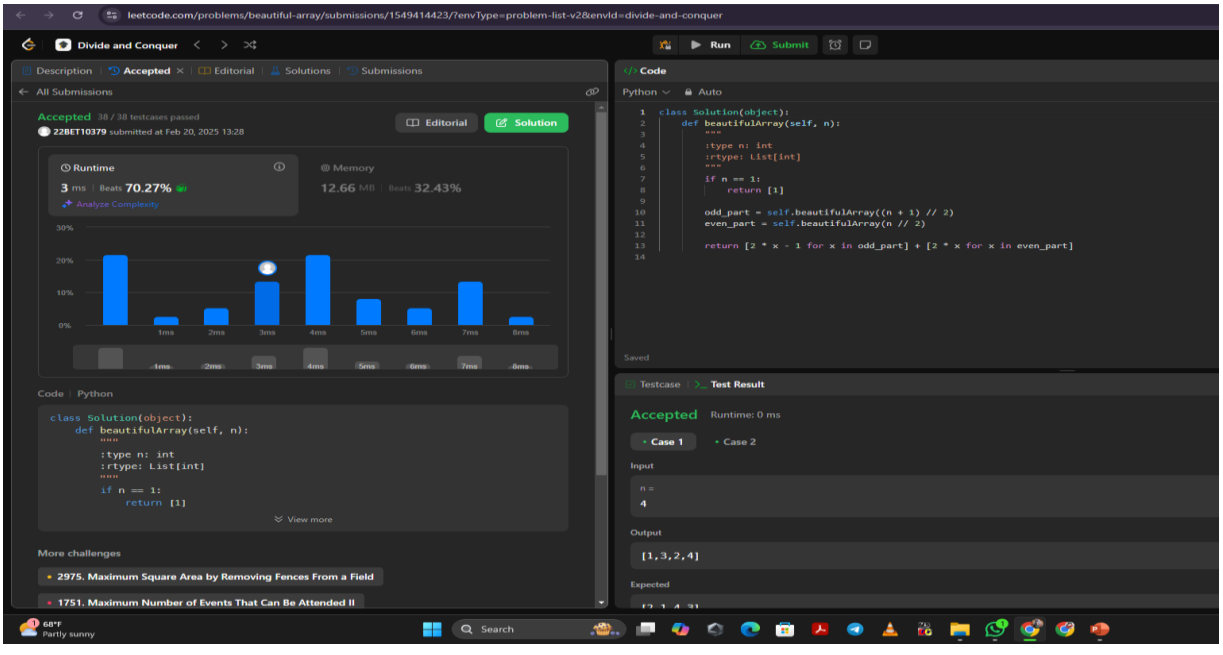
For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

1. **Objective:** Given the integer n, return any beautiful array nums of length n. There will be at least one valid answer for the given n.

2. **Code:**

```
class Solution(object):
    def beautifulArray(self, n):
        """
        :type n: int
        :rtype: List[int]
        """
        if n == 1:
            return [1]

        odd_part = self.beautifulArray((n + 1) // 2)
        even_part = self.beautifulArray(n // 2)

        return [2 * x - 1 for x in odd_part] + [2 * x for x in
even_part]
```

3.**Result:**



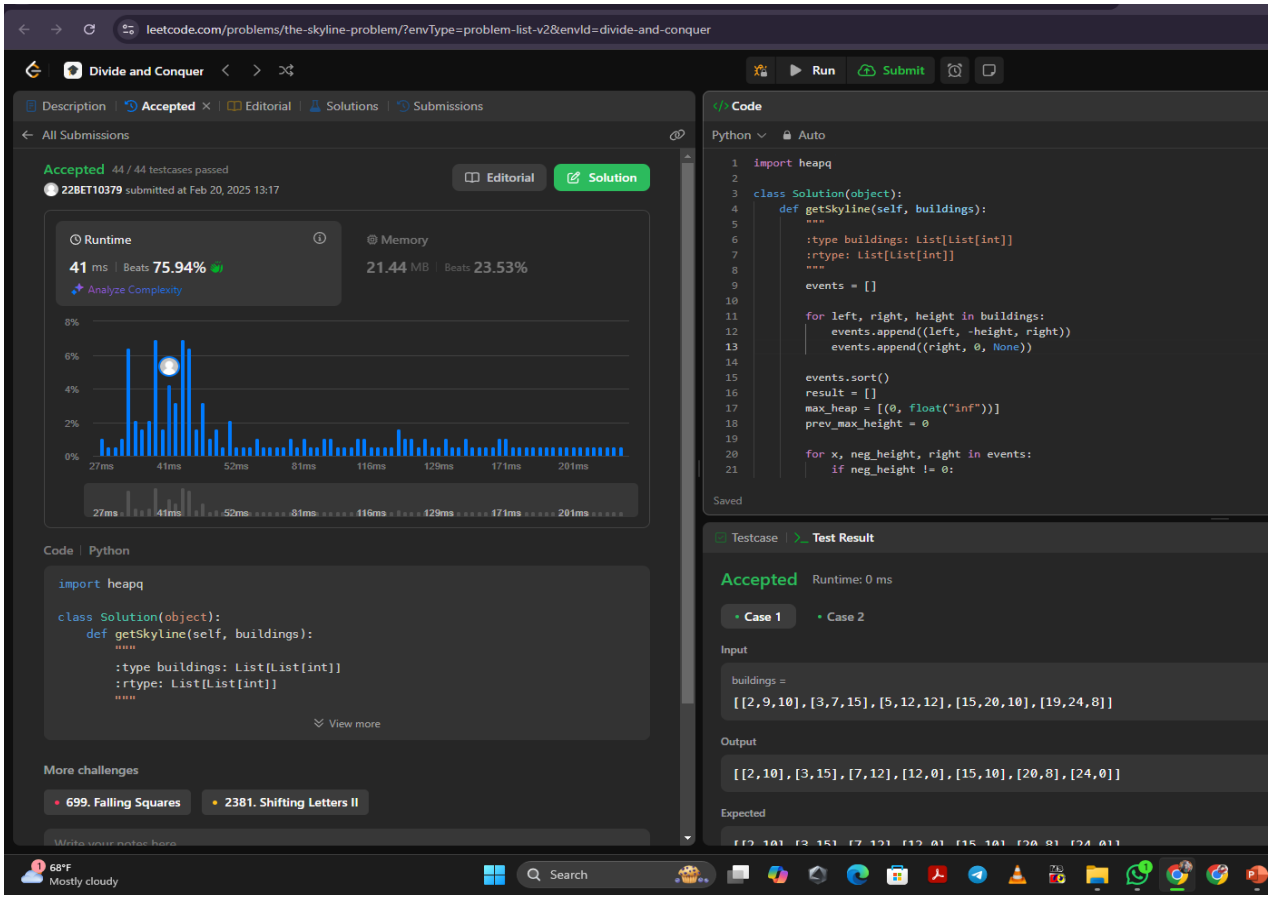
Problem 1.4.6: The Skyline Problem

Problem Statement: Given a list of buildings, where each building is represented as a triplet [L,R,H][L, R, H][L,R,H] (with LLL as the left x-coordinate, RRR as the right x-coordinate, and HHH as the height), your task is to output the skyline formed by these buildings. The skyline is a list of "key points" [x,y][x, y][x,y] that represent where the height of the skyline changes. Key points should be output in sorted order by the x-coordinate.

1. **Objective:** Determine the key points that form the outer contour (skyline) when the buildings are viewed from a distance.
2. **Code:**

```
import heapq

class Solution(object):
    def getSkyline(self, buildings):
        """
        :type buildings: List[List[int]]
        :rtype: List[List[int]]
        """
        events = []

        for left, right, height in buildings:
            events.append((left, -height, right))
            events.append((right, 0, None))

        events.sort()
        result = []
        max_heap = [(0, float("inf"))]
        prev_max_height = 0

        for x, neg_height, right in events:
            if neg_height != 0:
                heapq.heappush(max_heap, (neg_height, right))
            while max_heap[0][1] <= x:
                heapq.heappop(max_heap)

            current_max_height = -max_heap[0][0]

            if current_max_height != prev_max_height:
                result.append([x, current_max_height])
                prev_max_height = current_max_height

        return result
```

3. Result:



Problem 1.4.6: Reverse Pairs

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

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

0 <= i < j < nums.length and

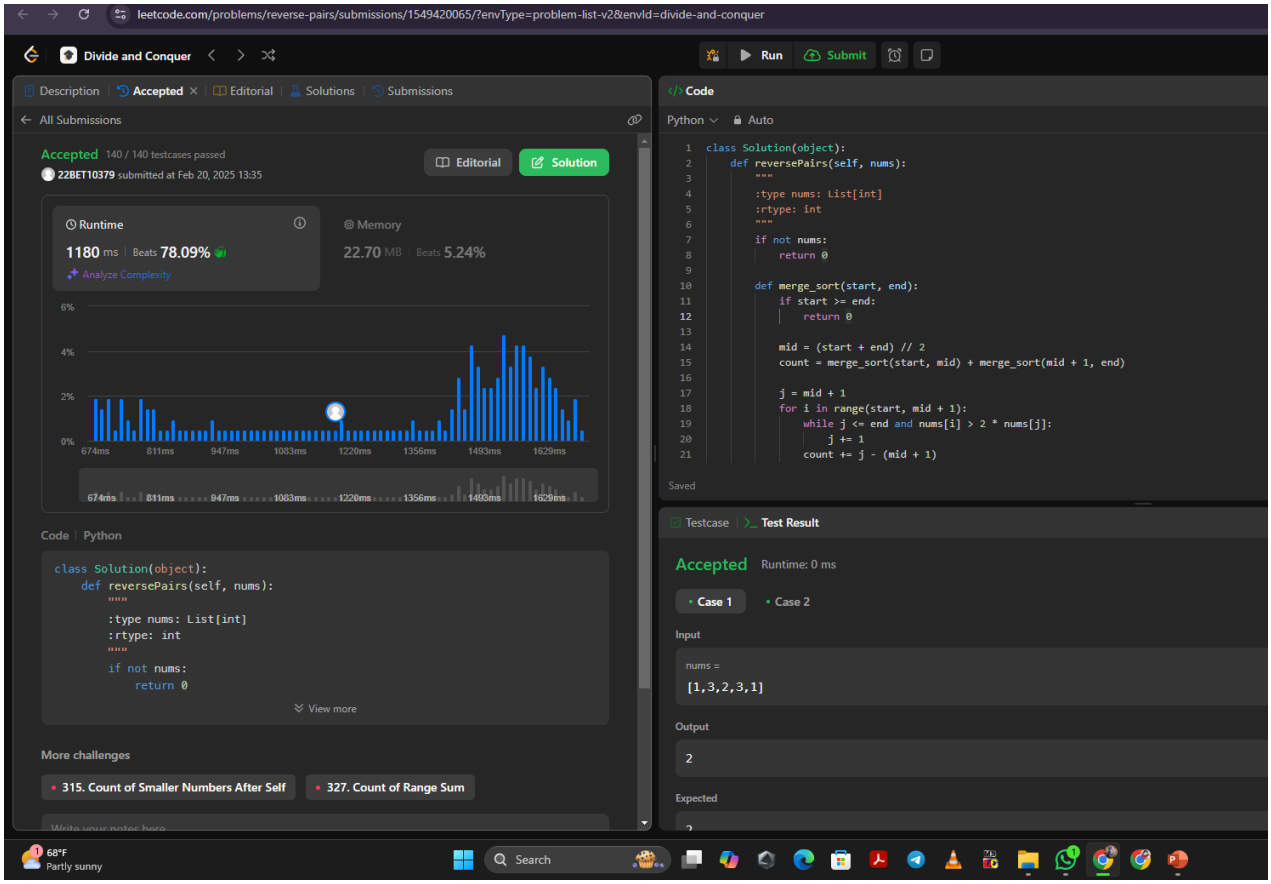
nums[i] > 2 * nums[j].

2. Code

```
class Solution(object):
    def reversePairs(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
```

```
def merge_sort(start, end):
        if start >= end:
            return 0

        mid = (start + end) // 2
        count = merge_sort(start, mid) + merge_sort(mid +
1, end)

        j = mid + 1
        for i in range(start, mid + 1):
            while j <= end and nums[i] > 2 * nums[j]:
                j += 1
            count += j - (mid + 1)

        temp = []
        i, j = start, mid + 1
        while i <= mid and j <= end:
            if nums[i] <= nums[j]:
                temp.append(nums[i])
                i += 1
            else:
                temp.append(nums[j])
                j += 1
        while i <= mid:
            temp.append(nums[i])
            i += 1
        while j <= end:
            temp.append(nums[j])
            j += 1

        nums[start:end+1] = temp
        return count

    return merge_sort(0, len(nums) - 1)
```

3.Result



4. Learning Outcomes:

- Understanding Reverse Pairs: Identifying pairs where nums[i] > 2 * nums[j].
- Merge Sort Usage: Applying divide and conquer for efficient counting.
- Optimized Counting: Using a two-pointer technique during merging.
- Time Complexity: Improving from $O(n^2)$ (brute force) to O(n log n) (merge sort).
- Sorting for Pair Counting: Leveraging sorted subarrays to count efficiently.
- Practical Applications: Useful in inversions counting, range queries, and stock market analysis.