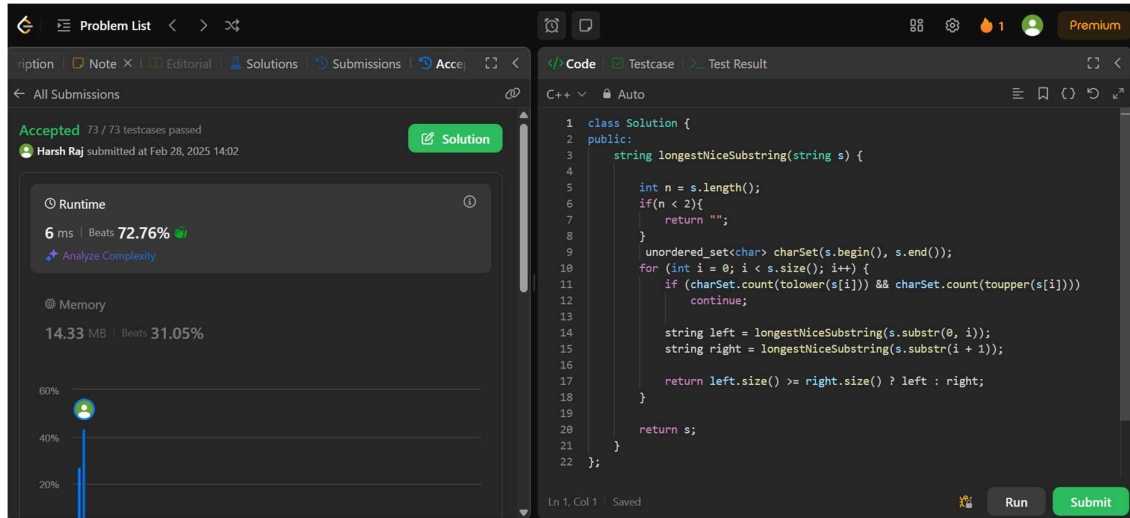


AP Assignment 4

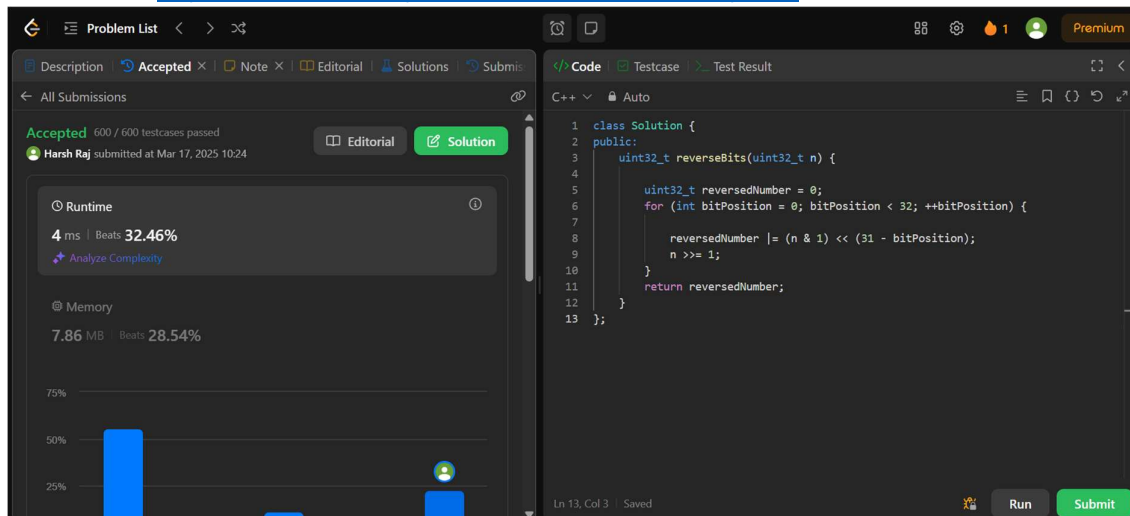
1. Longest Nice Substring: <https://leetcode.com/problems/longest-nice-substring/description/>



The screenshot shows the LeetCode interface for the 'Longest Nice Substring' problem. The submission is 'Accepted' with 73/73 testcases passed. The runtime is 6 ms, beating 72.76% of submissions. The memory usage is 14.33 MB, beating 31.05%. The code is written in C++ and implements a recursive solution using a set to track characters in the current substring.

```
1 class Solution {
2 public:
3     string longestNiceSubstring(string s) {
4
5         int n = s.length();
6         if(n < 2){
7             return "";
8         }
9         unordered_set<char> charSet(s.begin(), s.end());
10        for (int i = 0; i < s.size(); i++) {
11            if (charSet.count(tolower(s[i])) && charSet.count(toupper(s[i])))
12                continue;
13
14            string left = longestNiceSubstring(s.substr(0, i));
15            string right = longestNiceSubstring(s.substr(i + 1));
16
17            return left.size() >= right.size() ? left : right;
18        }
19
20        return s;
21    }
22};
```

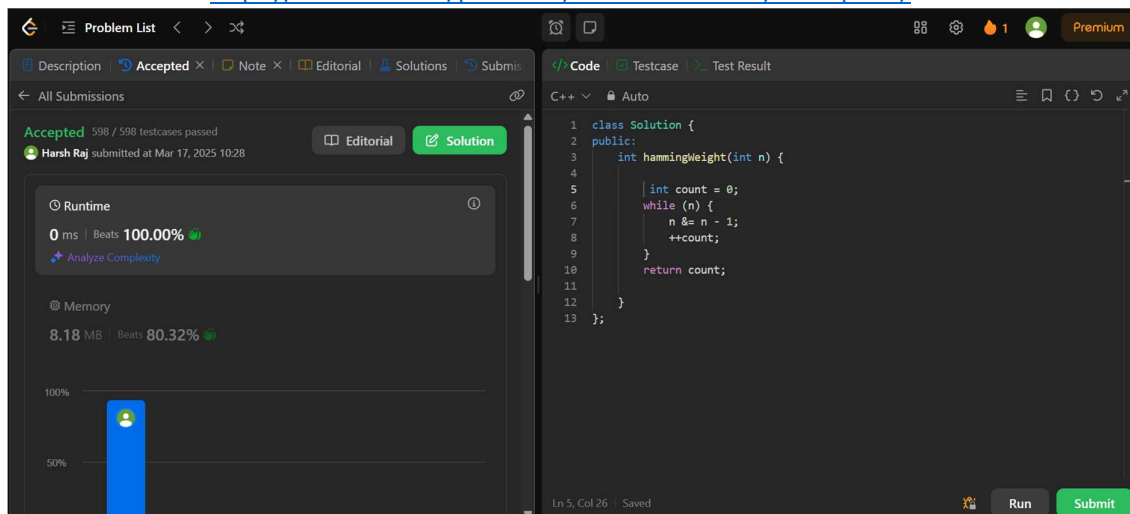
2. Reverse Bits: <https://leetcode.com/problems/reverse-bits/description/>



The screenshot shows the LeetCode interface for the 'Reverse Bits' problem. The submission is 'Accepted' with 600/600 testcases passed. The runtime is 4 ms, beating 32.46% of submissions. The memory usage is 7.86 MB, beating 28.54%. The code is written in C++ and implements a bit-reversal algorithm using a loop to shift bits.

```
1 class Solution {
2 public:
3     uint32_t reverseBits(uint32_t n) {
4
5         uint32_t reversedNumber = 0;
6         for (int bitPosition = 0; bitPosition < 32; ++bitPosition) {
7
8             reversedNumber |= (n & 1) << (31 - bitPosition);
9             n >>= 1;
10        }
11        return reversedNumber;
12    }
13};
```

3. Number of 1 Bits: <https://leetcode.com/problems/number-of-1-bits/description/>

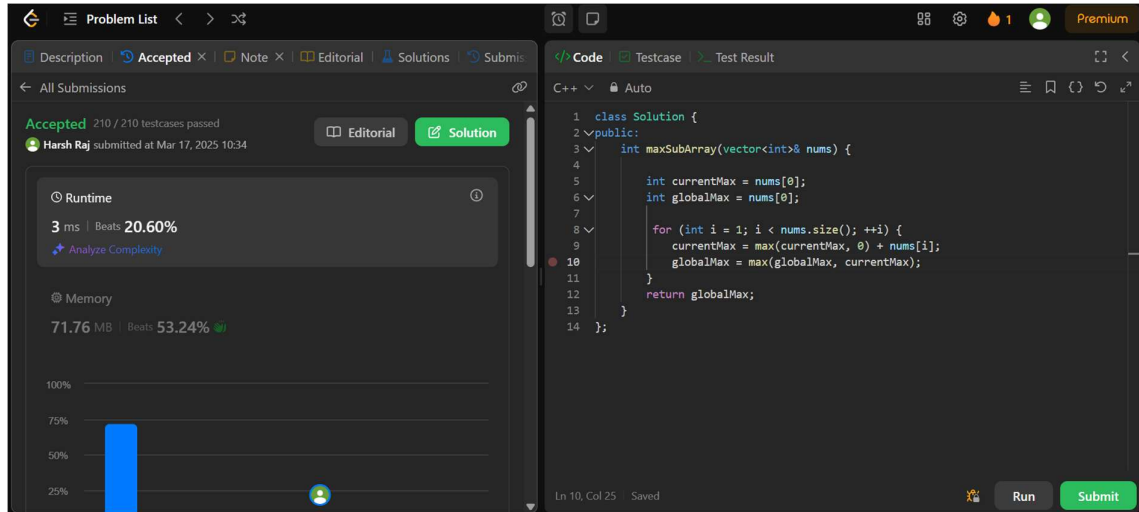


The screenshot shows the LeetCode interface for the 'Number of 1 Bits' problem. The submission is 'Accepted' with 598/598 testcases passed. The runtime is 0 ms, beating 100.00% of submissions. The memory usage is 8.18 MB, beating 80.32% of submissions. The code is written in C++ and implements a simple loop to count the number of 1 bits in the input number.

```
1 class Solution {
2 public:
3     int hammingWeight(int n) {
4
5         int count = 0;
6         while (n) {
7             n &= n - 1;
8             ++count;
9         }
10        return count;
11    }
12};
```

AP Assignment 4

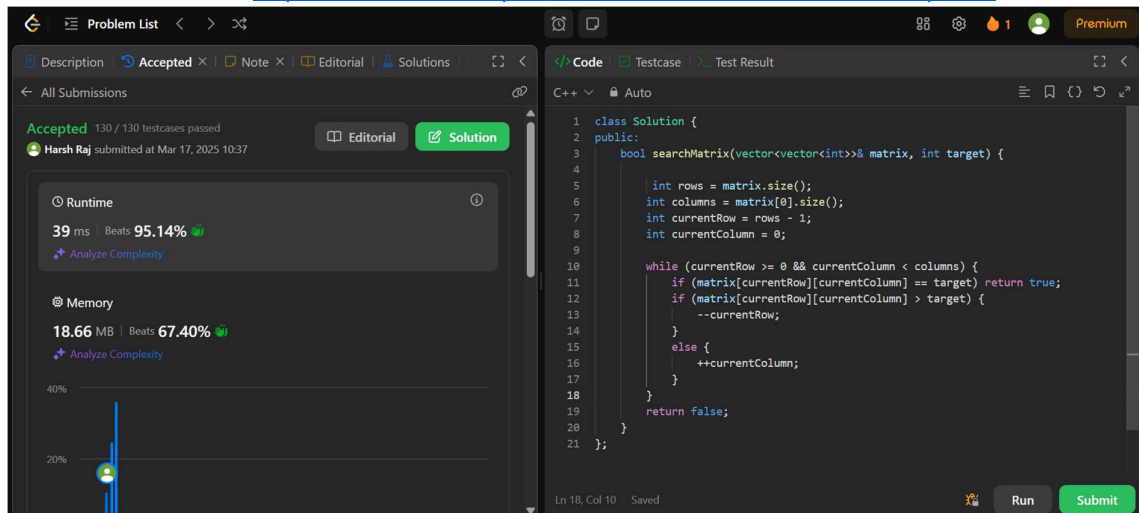
4. Maximum Subarray: <https://leetcode.com/problems/maximum-subarray/description/>



The screenshot shows the LeetCode interface for the 'Maximum Subarray' problem. On the left, the 'Runtime' section indicates 3 ms and 20.60% beats, while the 'Memory' section shows 71.76 MB and 53.24% beats. A bar chart at the bottom of the memory section shows the user's performance relative to others. The right pane displays the C++ code for the solution, which uses a single-pass algorithm to find the maximum subarray sum.

```
1 class Solution {
2 public:
3     int maxSubArray(vector<int>& nums) {
4
5         int currentMax = nums[0];
6         int globalMax = nums[0];
7
8         for (int i = 1; i < nums.size(); ++i) {
9             currentMax = max(currentMax, 0) + nums[i];
10            globalMax = max(globalMax, currentMax);
11        }
12        return globalMax;
13    }
14};
```

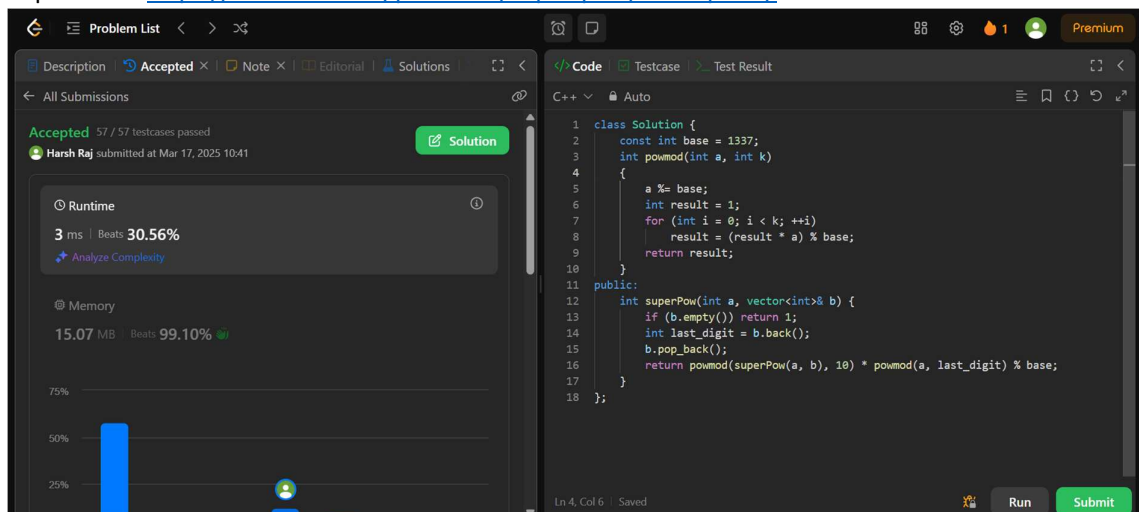
5. Search a 2D Matrix II: <https://leetcode.com/problems/search-a-2d-matrix-ii/description/>



The screenshot shows the LeetCode interface for the 'Search a 2D Matrix II' problem. The 'Runtime' section shows 39 ms and 95.14% beats, and the 'Memory' section shows 18.66 MB and 67.40% beats. The right pane displays the C++ code for the solution, which uses a step-by-step search starting from the top-right element of the matrix.

```
1 class Solution {
2 public:
3     bool searchMatrix(vector<vector<int>>& matrix, int target) {
4
5         int rows = matrix.size();
6         int columns = matrix[0].size();
7         int currentRow = rows - 1;
8         int currentColumn = 0;
9
10        while (currentRow >= 0 && currentColumn < columns) {
11            if (matrix[currentRow][currentColumn] == target) return true;
12            if (matrix[currentRow][currentColumn] > target) {
13                --currentRow;
14            }
15            else {
16                ++currentColumn;
17            }
18        }
19        return false;
20    }
21};
```

6. Super Pow: <https://leetcode.com/problems/super-pow/description/>



The screenshot shows the LeetCode interface for the 'Super Pow' problem. The 'Runtime' section shows 3 ms and 30.56% beats, and the 'Memory' section shows 15.07 MB and 99.10% beats. The right pane displays the C++ code for the solution, which uses a recursive-like approach with a helper function 'powmod' to calculate the result efficiently.

```
1 class Solution {
2     const int base = 1337;
3     int powmod(int a, int k)
4     {
5         a %= base;
6         int result = 1;
7         for (int i = 0; i < k; ++i)
8             result = (result * a) % base;
9         return result;
10    }
11 public:
12     int superPow(int a, vector<int>& b) {
13         if (b.empty()) return 1;
14         int last_digit = b.back();
15         b.pop_back();
16         return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
17     }
18};
```

AP Assignment 4

7. Beautiful Array: <https://leetcode.com/problems/beautiful-array/description/>

The screenshot shows the LeetCode interface for the 'Beautiful Array' problem. On the left, the submission status is 'Accepted' with 38/38 testcases passed. The user 'Harsh Raj' submitted it on Mar 17, 2025 at 10:44. The runtime is 10 ms, beating 21.37% of solutions. The memory is 16.68 MB, beating 18.37% of solutions. A bar chart shows the user's performance relative to others. On the right, the C++ code is displayed, showing a recursive function 'beautifulArray' that builds the result by merging two halves.

```
1 class Solution {
2 public:
3     vector<int> beautifulArray(int n) {
4
5         if (n == 1) return {1};
6         vector<int> leftHalf = beautifulArray((n + 1) >> 1);
7         vector<int> rightHalf = beautifulArray(n >> 1);
8         vector<int> result(n);
9         int currentIndex = 0;
10        for (int& element : leftHalf) {
11            result[currentIndex++] = element * 2 - 1;
12        }
13        for (int& element : rightHalf) {
14            result[currentIndex++] = element * 2;
15        }
16        return result;
17    }
18 }
```

8. The Skyline Problem: <https://leetcode.com/problems/the-skyline-problem/description/>

The screenshot shows the LeetCode interface for the 'The Skyline Problem'. The submission status is 'Accepted' with 44/44 testcases passed. The user 'Harsh Raj' submitted it on Mar 17, 2025 at 10:53. The runtime is 18 ms, beating 55.18% of solutions. The memory is 28.76 MB, beating 50.38% of solutions. A bar chart shows the user's performance relative to others. On the right, the C++ code is displayed, showing a function 'getSkyline' that processes building points and uses a priority queue to maintain the current skyline height.

```
1 class Solution {
2 public:
3     vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
4         vector<vector<int>> ans;
5         multiset<int> pq{0};
6
7         vector<pair<int, int>> points;
8
9         for(auto b: buildings){
10             points.push_back({b[0], -b[2]});
11             points.push_back({b[1], b[2]});
12         }
13         sort(points.begin(), points.end());
14         int ongoingHeight = 0;
15         for(int i = 0; i < points.size(); i++){
16             int currentPoint = points[i].first;
17             int heightAtCurrentPoint = points[i].second;
18
19             if(heightAtCurrentPoint < 0){
20                 pq.insert(-heightAtCurrentPoint);
21             } else {
22                 pq.erase(pq.find(heightAtCurrentPoint));
23             }
24         }
25     }
```

9. Reverse Pairs: <https://leetcode.com/problems/reverse-pairs/description/>

The screenshot shows the LeetCode interface for the 'Reverse Pairs' problem. The submission status is 'Accepted'. The user 'Harsh Raj' submitted it on Mar 17, 2025 at 11:33. The runtime is 233 ms, beating 82.25% of solutions. The memory is 112.98 MB, beating 81.07% of solutions. A bar chart shows the user's performance relative to others. On the right, the C++ code is displayed, showing a merge sort algorithm that counts reverse pairs during the merging process.

```
1 class Solution {
2 private:
3     void merge(vector<int>& nums, int low, int mid, int high, int& reversePairsCount) {
4         int j = mid+1;
5         for(int i=low; i<=mid; i++){
6             while(j<=high && nums[i] > 2*(long long)nums[j]){
7                 j++;
8             }
9             reversePairsCount += j-(mid+1);
10        }
11        int size = high-low+1;
12        vector<int> temp(size, 0);
13        int left = low, right = mid+1, k=0;
14        while(left<=mid && right<=high){
15            if(nums[left] < nums[right]){
16                temp[k++] = nums[left++];
17            }
18            else{
19                temp[k++] = nums[right++];
20            }
21        }
22        while(left<=mid){
23            // Copy remaining elements from left half
24        }
25    }
```

AP Assignment 4

10. Longest Increasing Subsequence II: <https://leetcode.com/problems/longest-increasing-subsequence-ii/description/>

The screenshot displays a LeetCode submission interface for the problem "Longest Increasing Subsequence II". The submission is marked as "Accepted" with 84 / 84 testcases passed. The user "Harsh Raj" submitted the solution on Mar 17, 2025 at 11:27. The runtime is 492 ms, beating 5.59% of solutions, and the memory usage is 214.44 MB, beating 9.11% of solutions. A bar chart shows the performance distribution. The code is written in C++ and implements a Segment Tree to efficiently calculate the longest increasing subsequence.

Runtime: 492 ms | Beats 5.59%

Memory: 214.44 MB | Beats 9.11%

```
1 #include <vector>
2 #include <algorithm>
3
4 using namespace std;
5 class Node {
6 public:
7     int left;
8     int right;
9     int value;
10 };
11
12 class SegmentTree {
13 private:
14     vector<Node*> tree;
15
16     void pushUp(int index) {
17         tree[index]->value = max(tree[index * 2]->value, tree[index * 2 + 1]->value);
18     }
19
20 public:
21     SegmentTree(int n) {
22         tree.resize(4 * n);
23     }
24 }
```