DAA HOLIDAY ASSIGNMENT

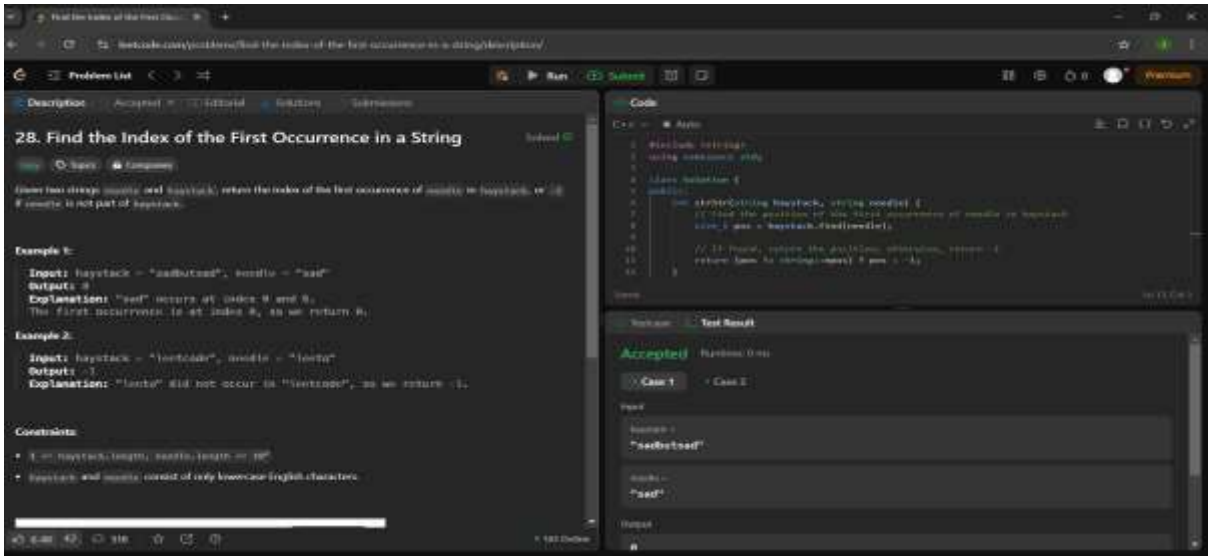
P.JANAKI

2211CS020388

AIML-THETA

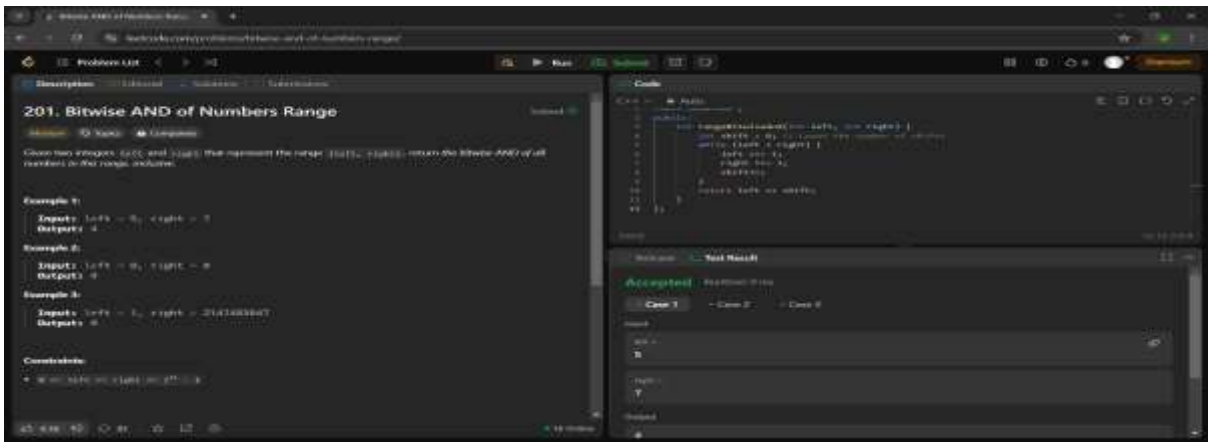
1) find-the-index-of-the-first-occurrence-in-a-string

https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/description/



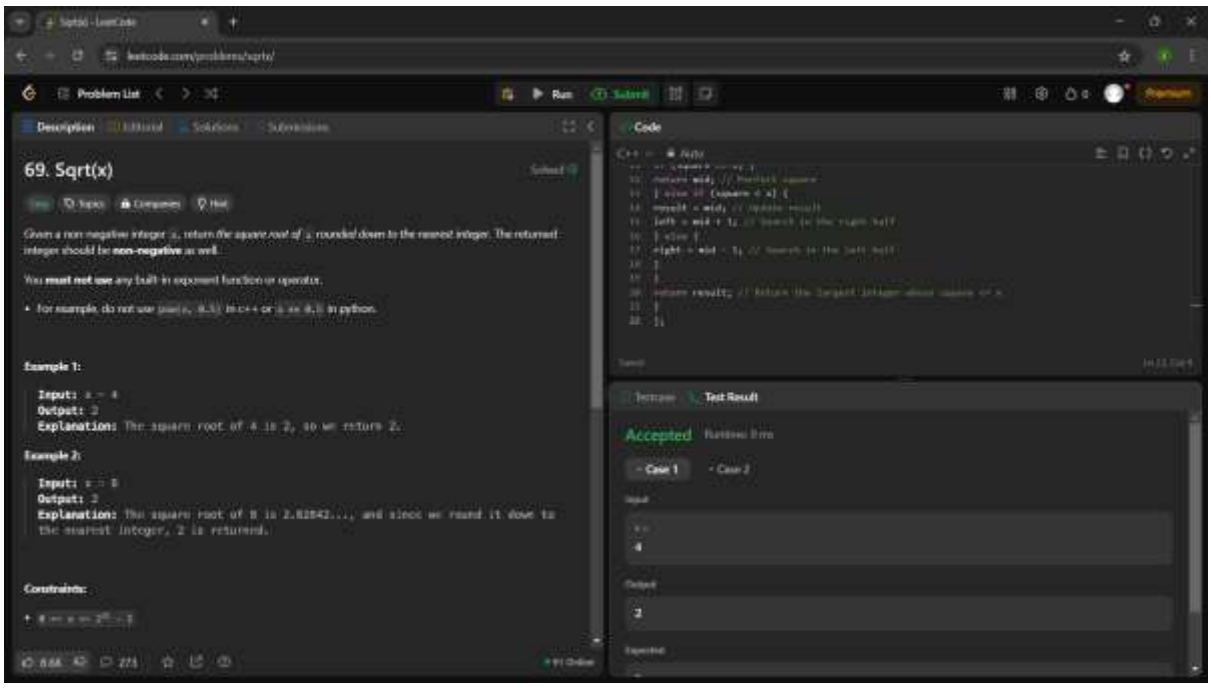
2)Bitwise AND of numberrange

https://leetcode.com/problems/bitwise-and-of-numbers-range/description/



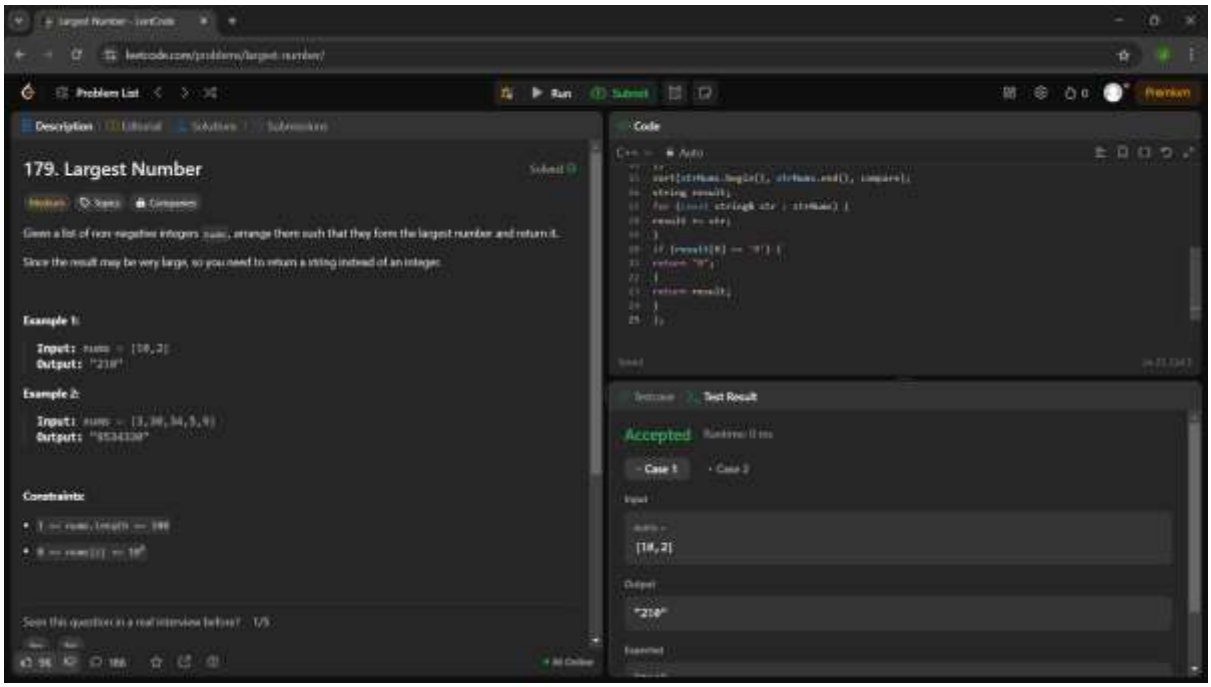
3)SQRT(X)

https://leetcode.com/problems/sqrtx



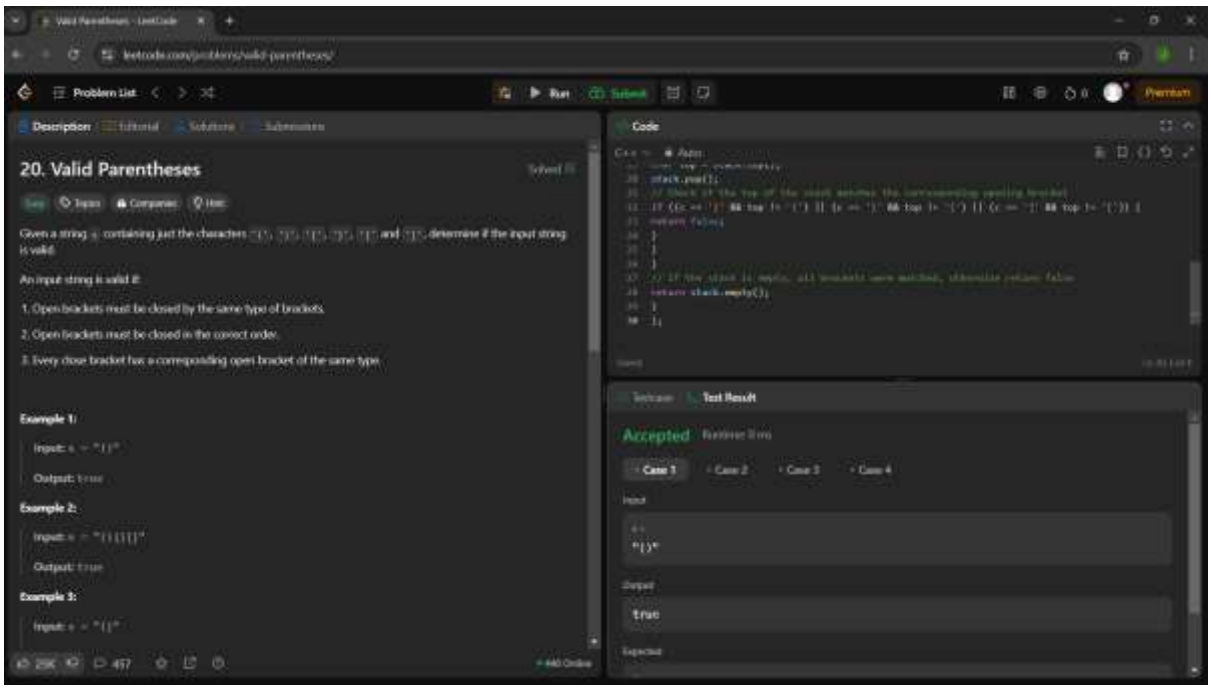
4)Largest Number

https://leetcode.com/problems/largest-number/description/



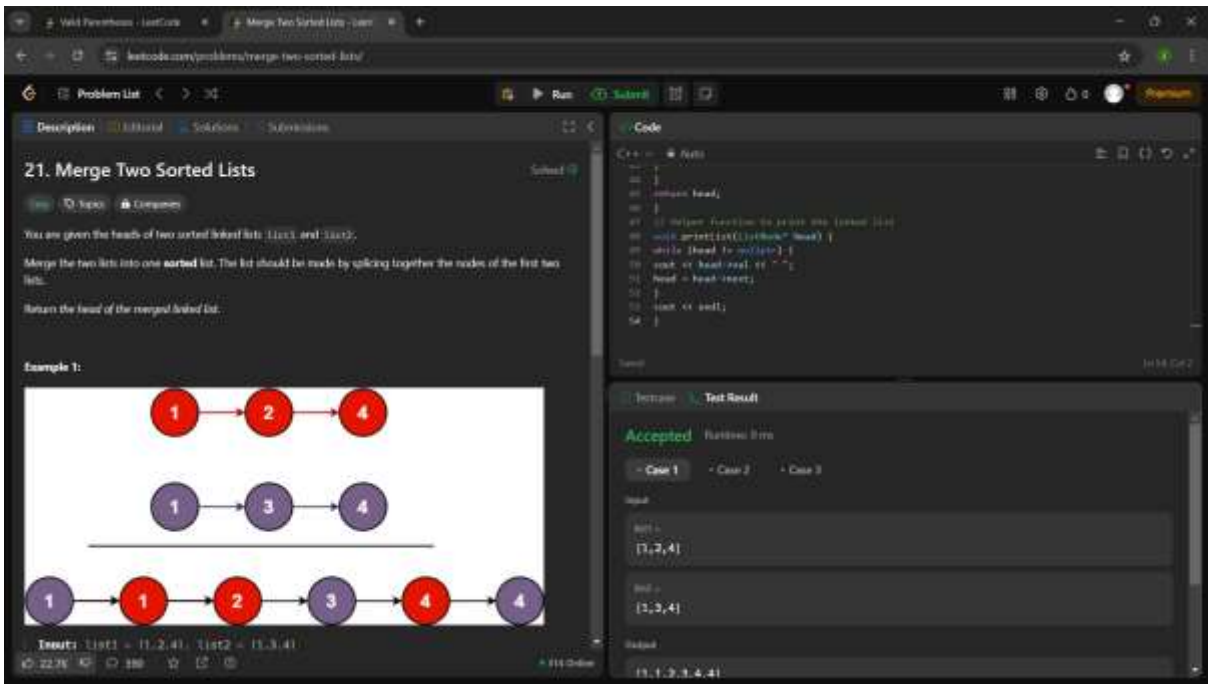
5)Valid Parenthesis

https://leetcode.com/problems/valid-parentheses/



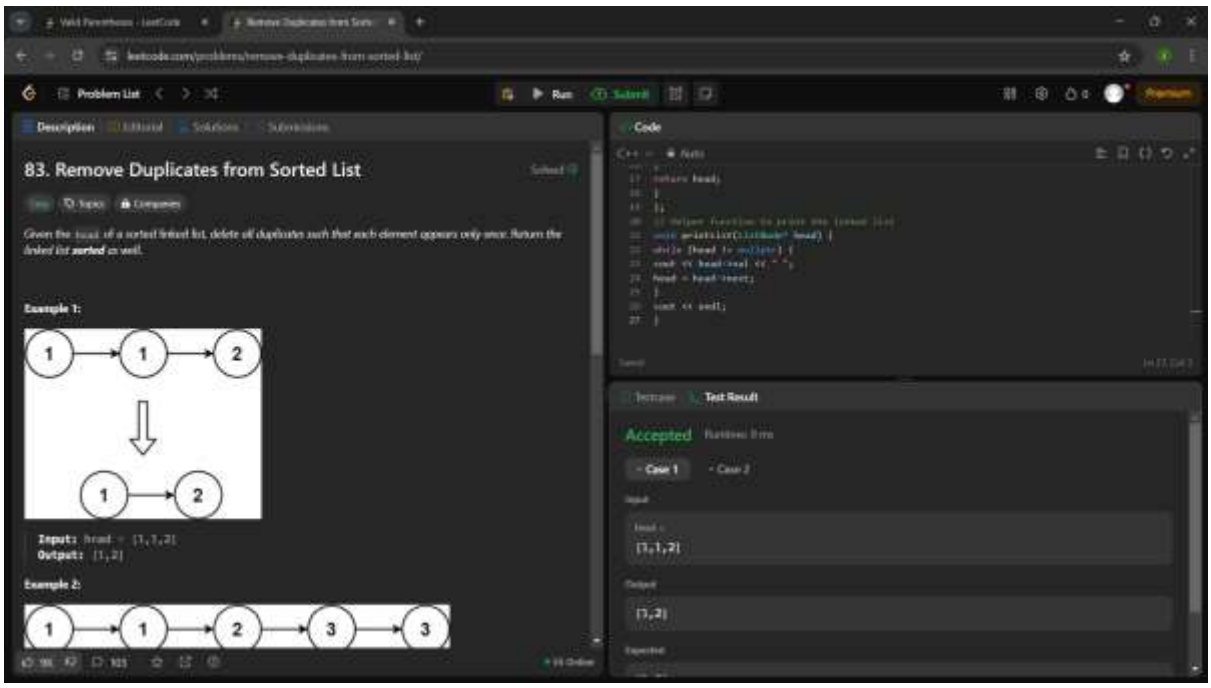
6)Merge Two Sorted Lists

https://leetcode.com/problems/merge-two-sorted-lists/



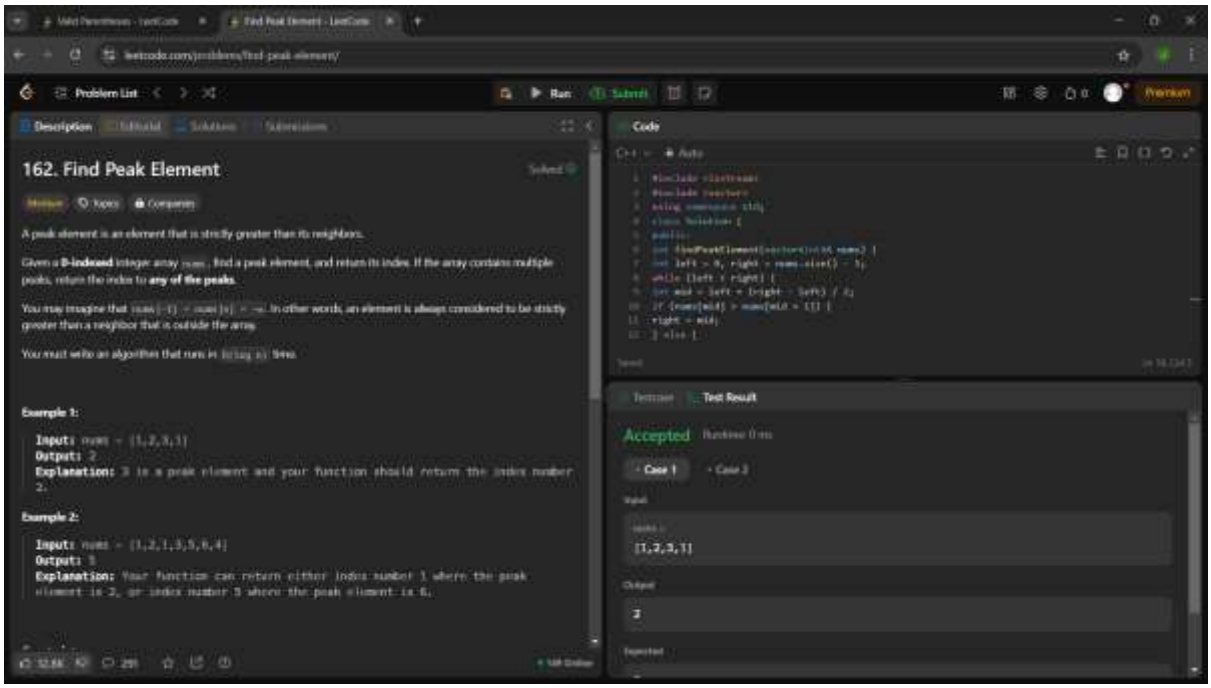
7)Remove Duplicates from sorted list

https://leetcode.com/problems/remove-duplicates-from-sorted-list/



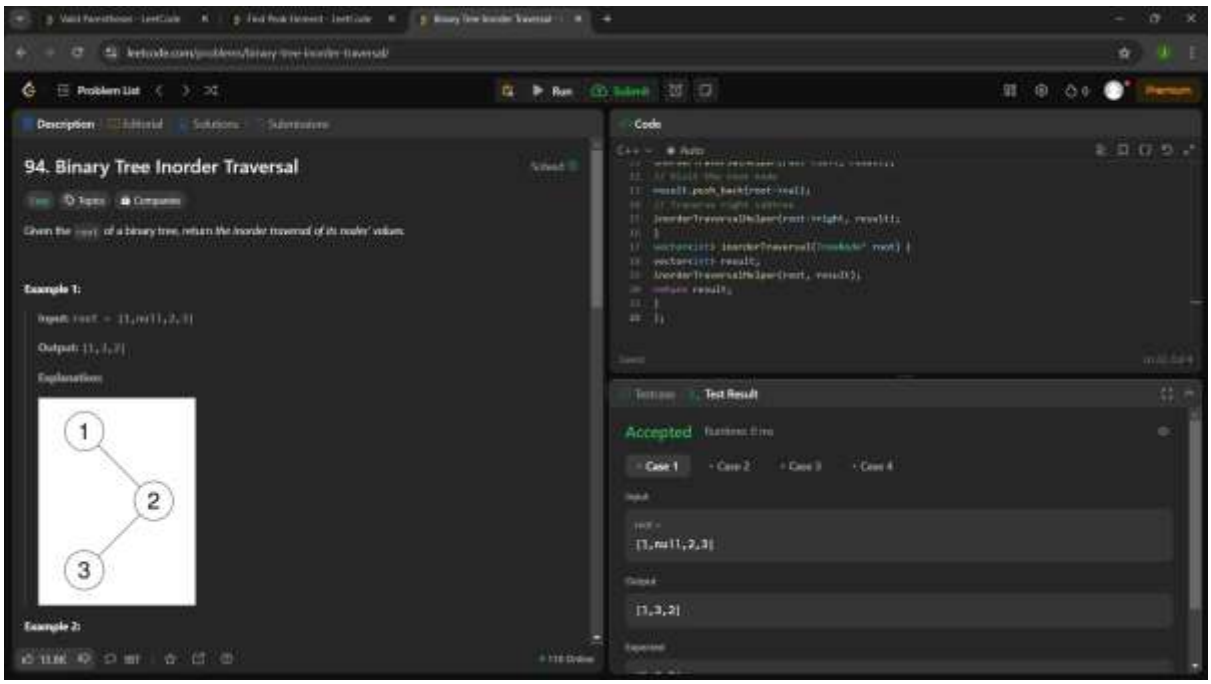
8)Find peak Element

https://leetcode.com/problems/find-peak-element/



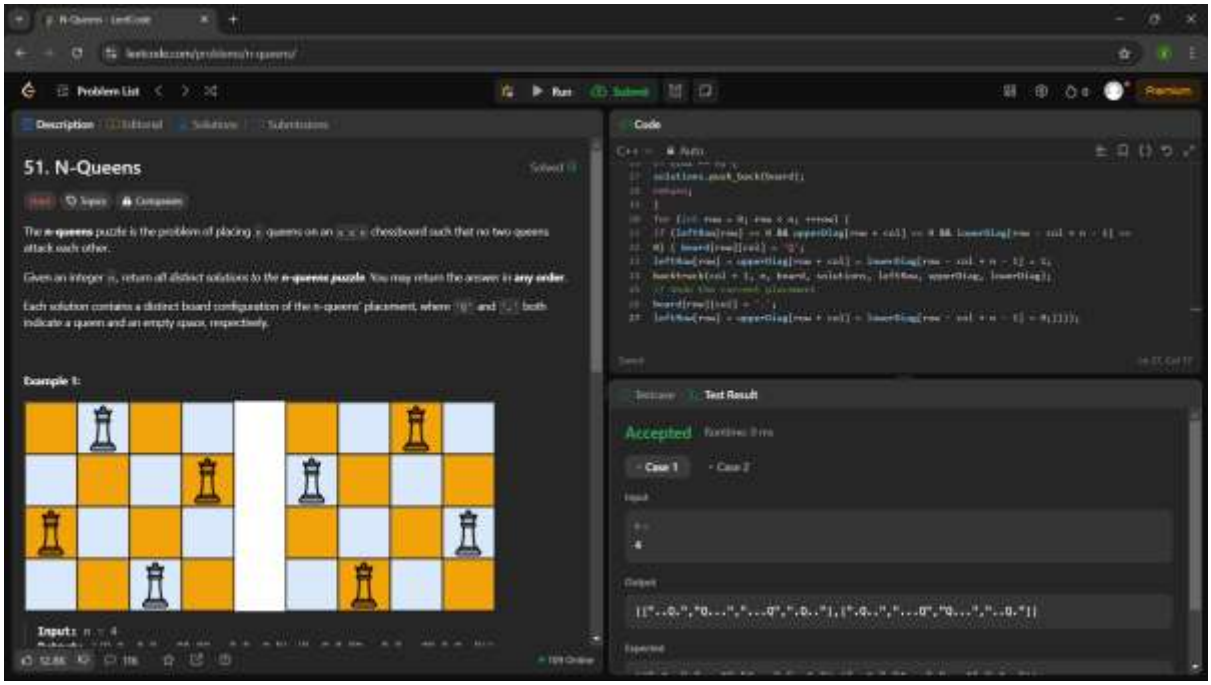
9)Binary Tree Inorder Traversal

https://leetcode.com/problems/binary-tree-inorder-traversal/



10)N-Queens

https://leetcode.com/problems/n-queens/



P Janaki
2211CS020388
AIML-Theta

DAA Case-Study

Scenario:-

An e-Commerce platform is implementing a feature where products need to be sorted by various attributes (e.g, price, rating and name). The product list contains millions of items, and the sorting operation needs to be efficient and scalable.

1. What are the time and space complexities of the commonly used sorting algorithms (QuickSort, MergeSort)?
2. How do the characteristics of the data (e.g range of prices, product name length) impact the choice of sorting algorithm?

1. Time and Space complexities of Common Sorting Algorithm

Quick Sort:-

Time Complexity (Best):- $O(\log n)$

Time complexity (Average):- $O(n \log n)$

Time Complexity (Worst):- $O(n^2)$ (unbalanced pivot)

Space complexity:- $O(\log n)$ auxiliary.

MergeSort:-

Time Complexity (Best):- $O(n \log n)$

Time Complexity (Average):- $O(n \log n)$

Time complexity (Worst):- $O(n \log n)$

Space complexity: $O(n)$ auxiliary

2. Impact of Data Characteristics on Sorting Algorithm choice.

1. Range of Prices or Numerical Data:-

- For a small range of values (e.g. product prices), Counting sort or Radix sort may be more efficient than comparison-based algorithms, achieving linear time complexity O(n+k).
- For a large or arbitrary range, Comparison-based algorithms like Quick Sort or Merge Sort are preferred.

2. Product Name Lengths (String Sorting):-

- Lexicographic sorting involves comparing strings character by character. Algorithms like QuickSort and MergeSort still perform well but may be slower due to increased comparsion times.
- Radix Sort can be used for fixed-length strings or cases where the length is bounded and known, providing near-linear time complexity.

3. Distribution and Size of Data:-

- Uniformly Distributed Data: QuickSort performs well with randomized pivot selection strategies.
- Nearly Sorted Data: InsertionSort (or) Tim Sort may out perform other due to better handling of such input.

4. Stability Requirements:-

- MergeSort is stable, meaning equal elements retain their relative order. QuickSort is generally not stable unless modified.

5. Memory Constraints:-

- If memory usage is a concern, Quick Sort (with-in place Sorting) is often more suitable than MergeSort.