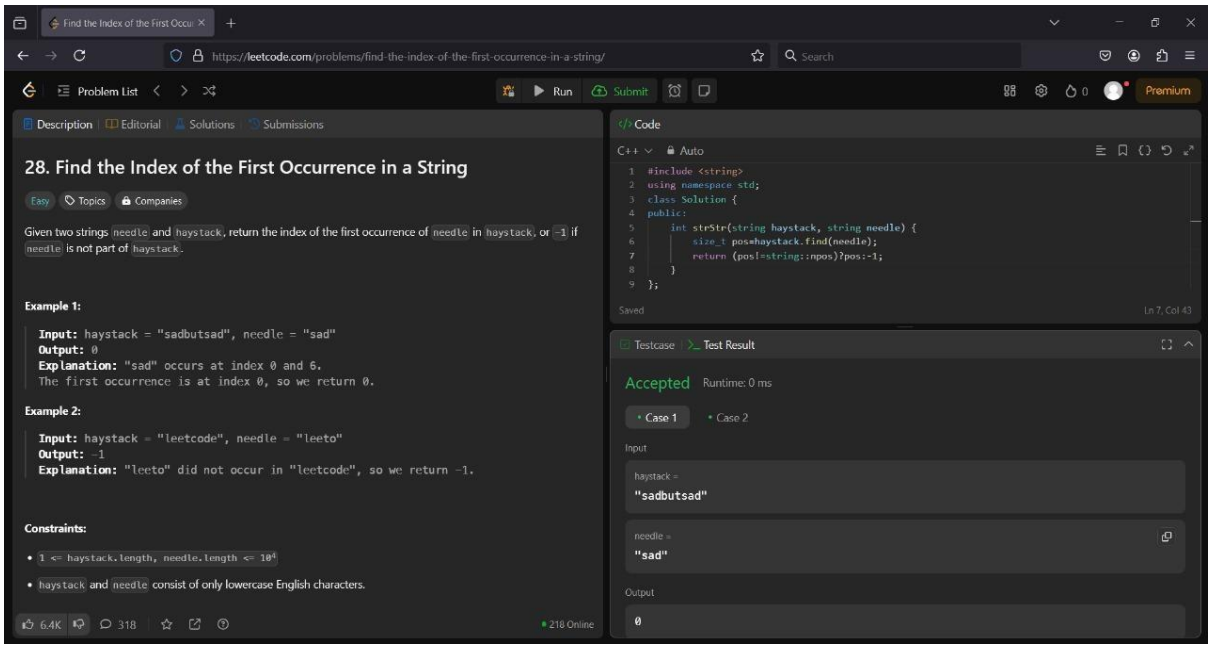
# DAA HOLIDAY ASSIGNMENT

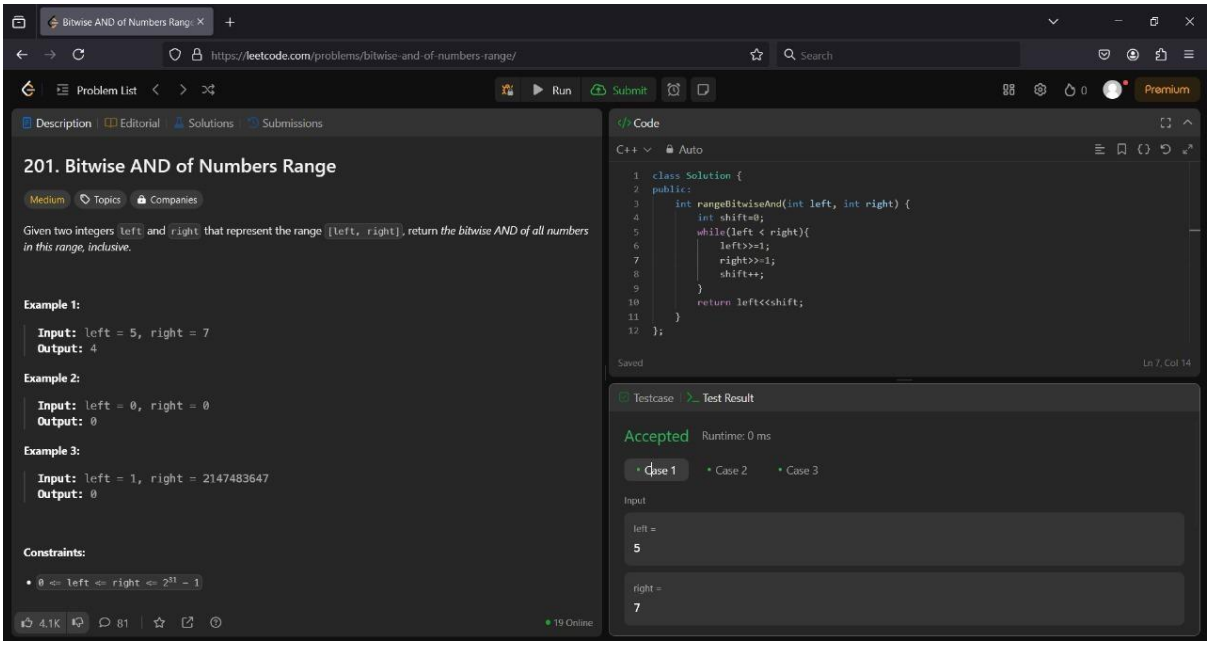
**P.SAI PRIYA**
**2211CS020404**

**AIML-THETA**

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

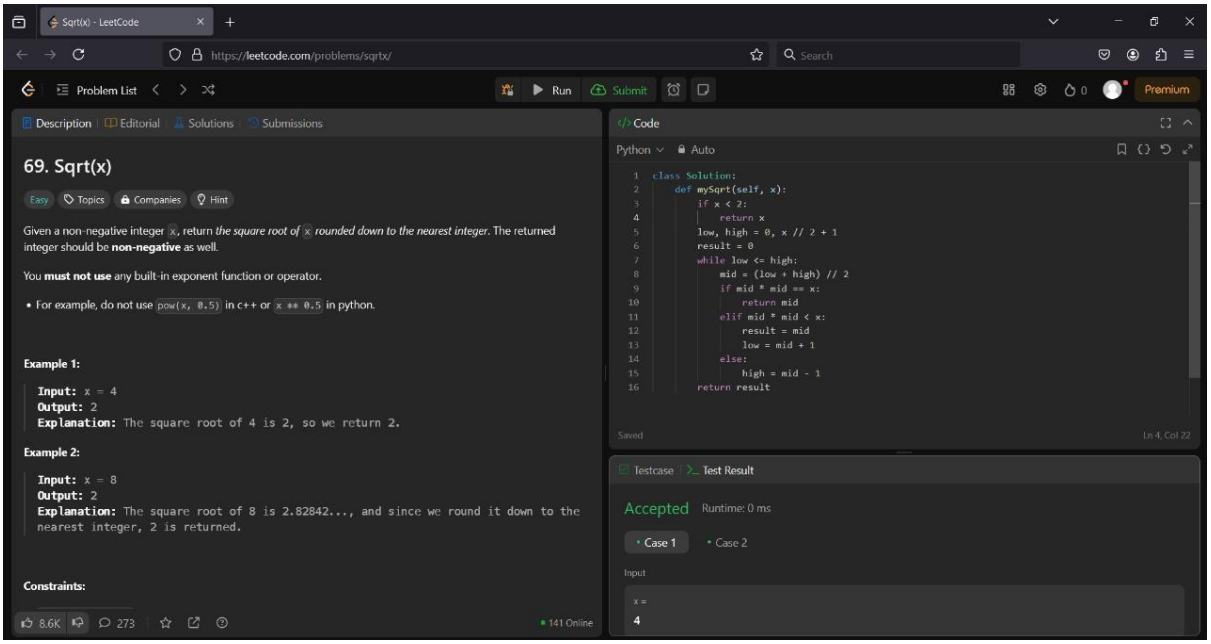



2) Bitwise AND of numbers range
https://leetcode.com/problems/bitwise-and-of-numbers-range/



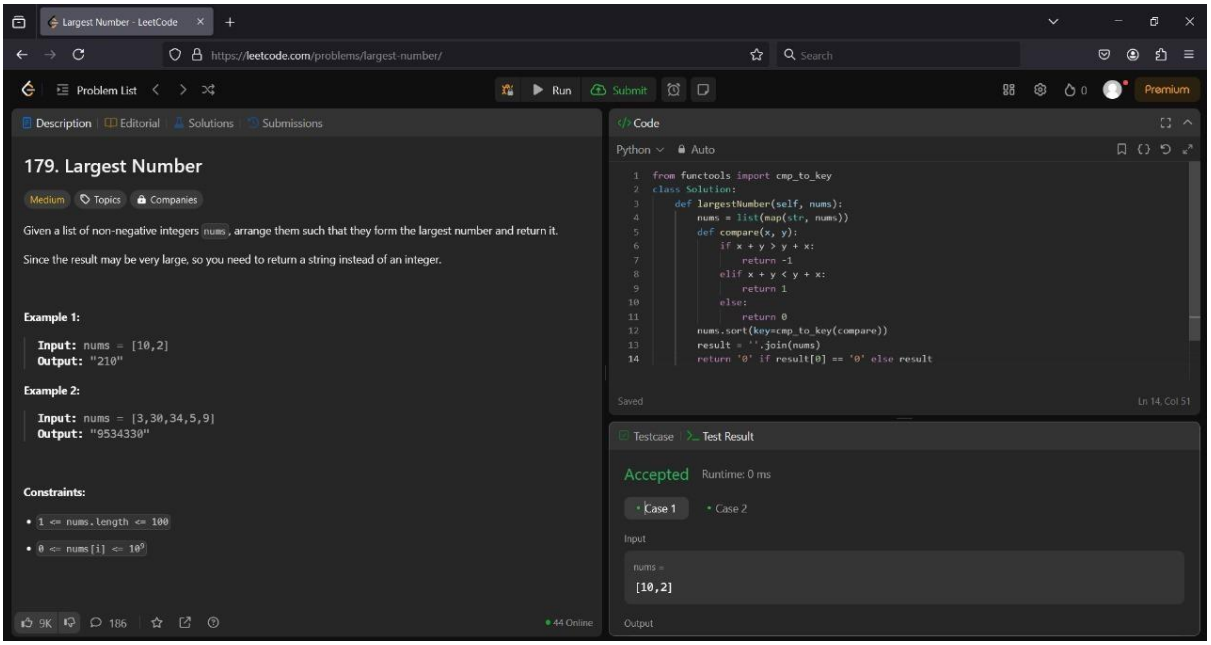

3)SQRT(X)

https://leetcode.com/problems/sqrtx/



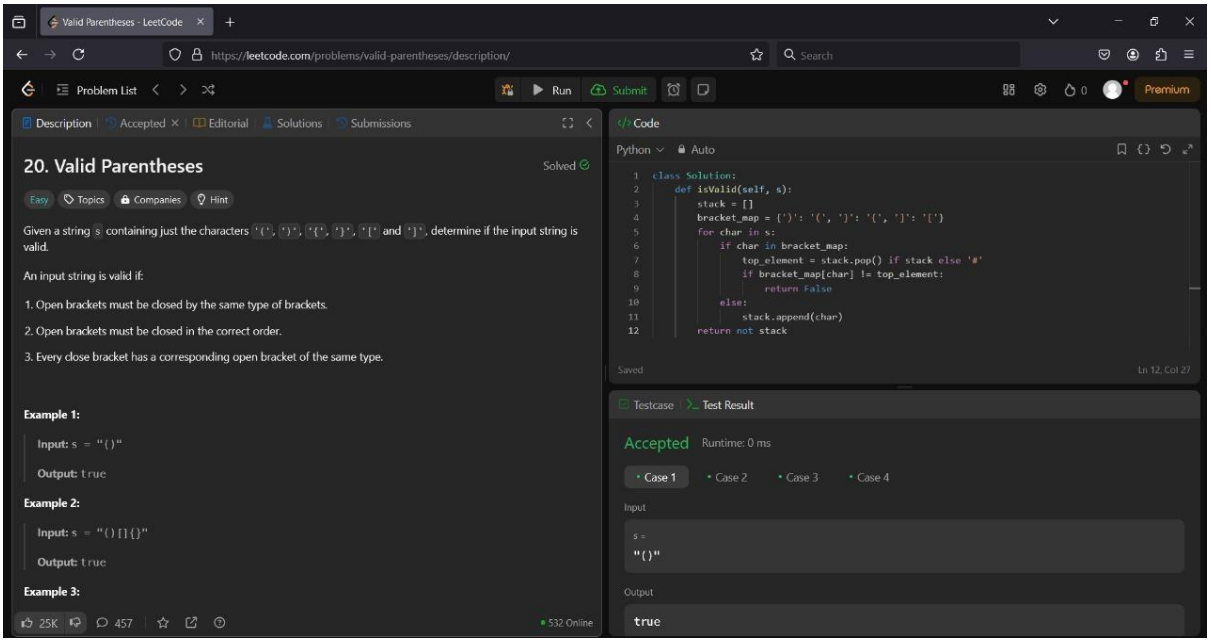

4)Largest Number

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



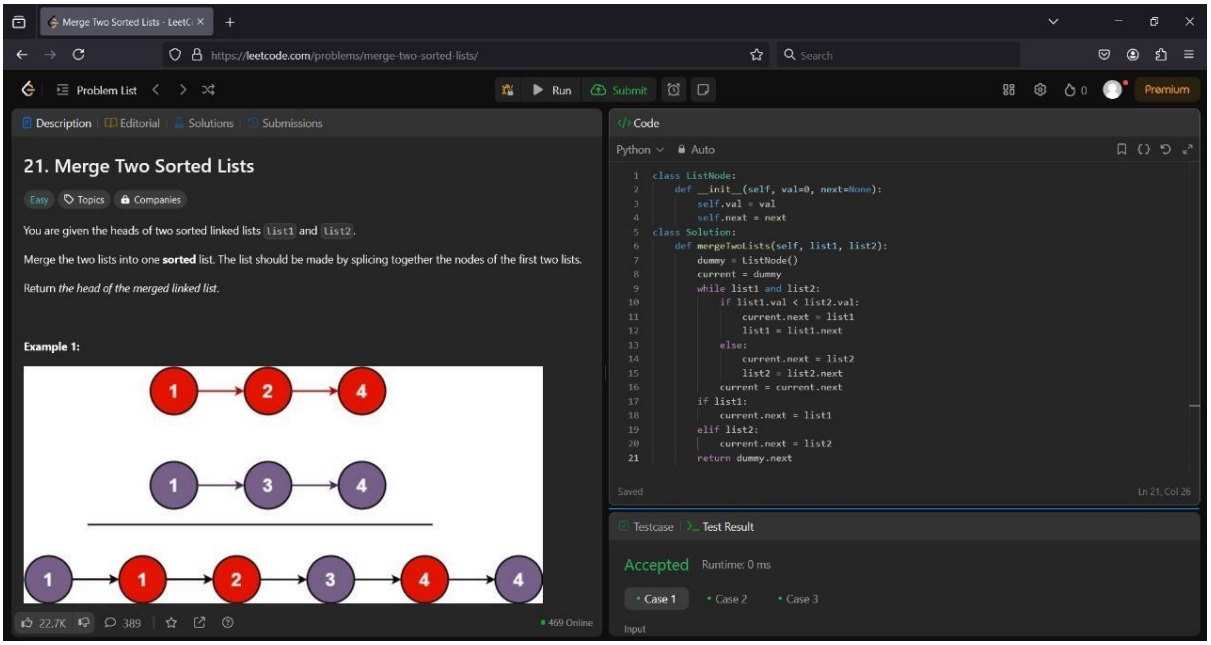

5)Valid Parenthesis

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



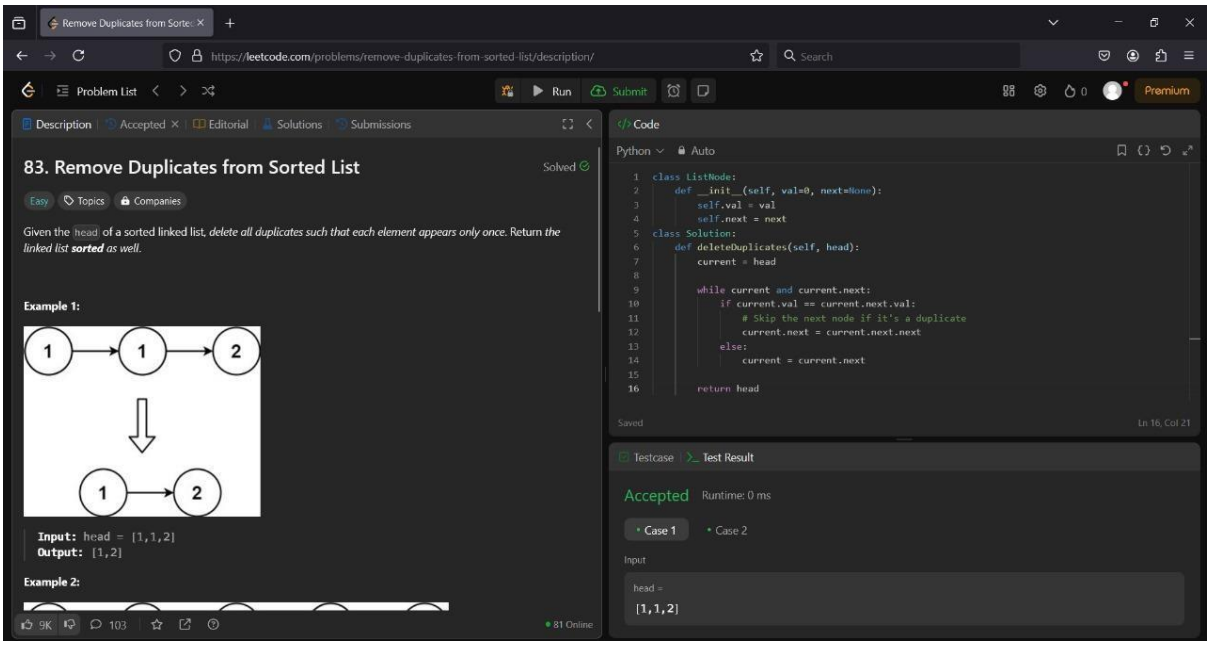

6)Merge Two Sorted Lists

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



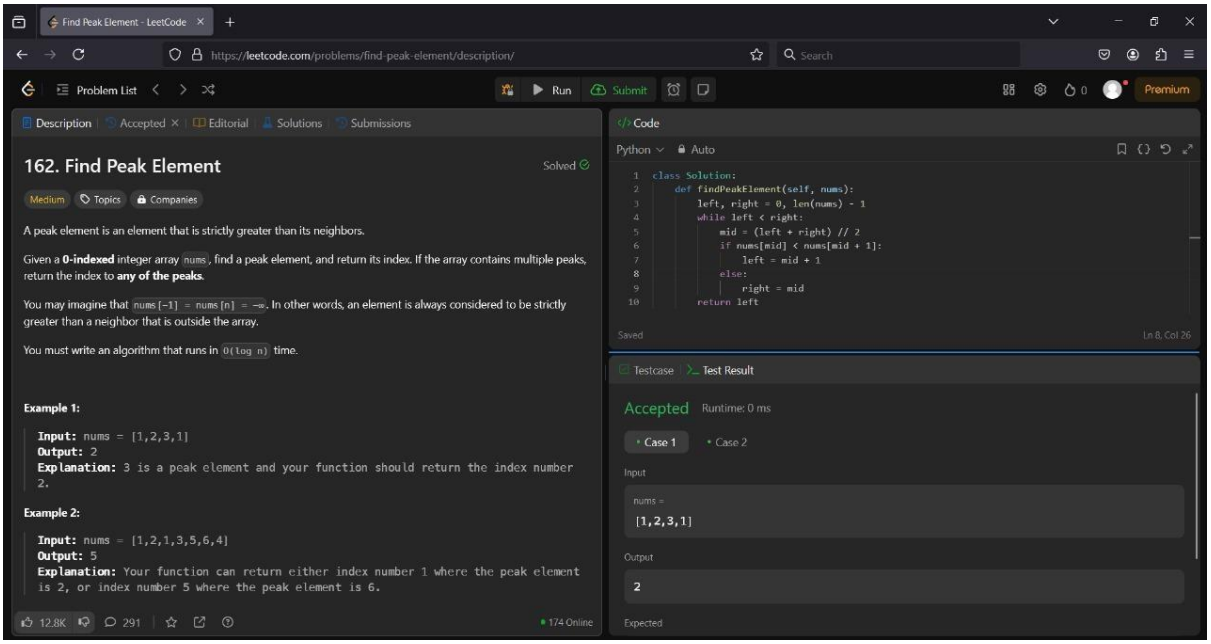

7)Remove Duplicates from sorted list

https://leetcode.com/problems/remove-duplicates-from-sortedlist/description/



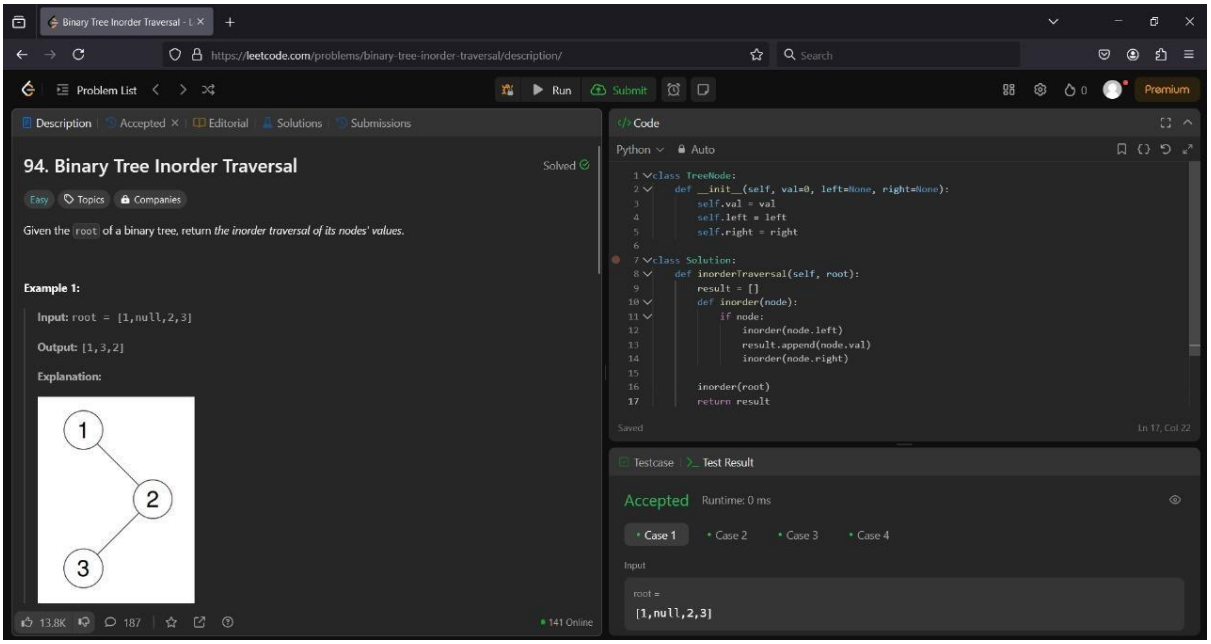

8)Find peak Element

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



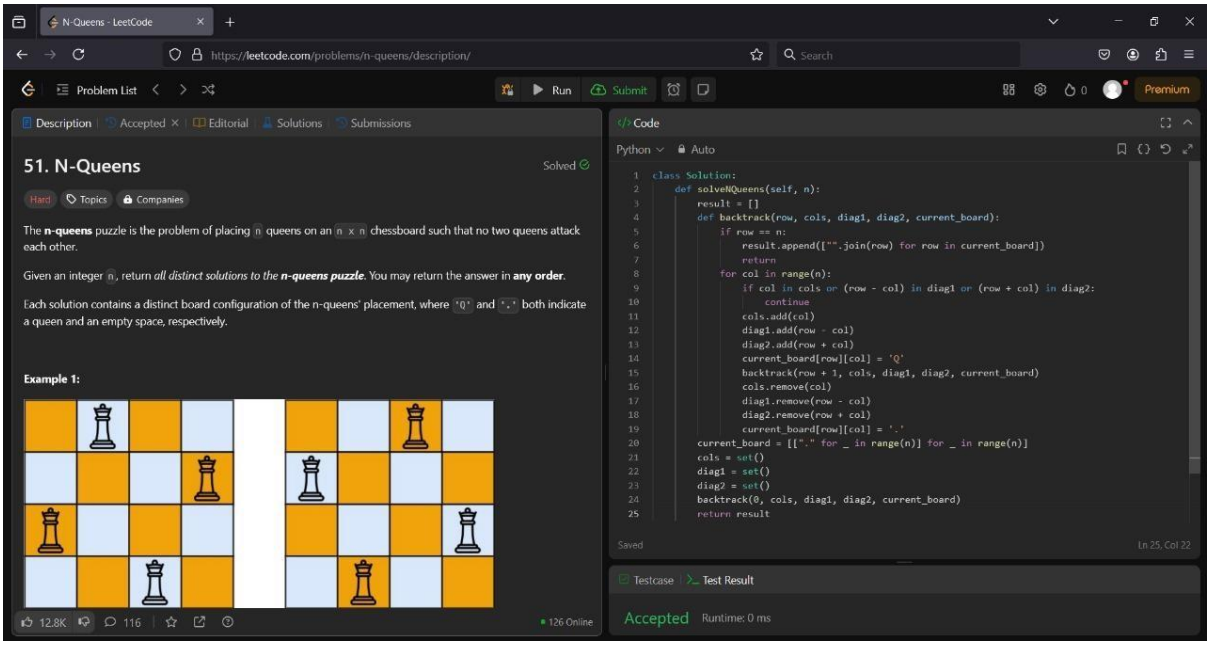

9)Binary Tree Inorder Traversal

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




10)N-Queens

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


N-Queens - LeetCode
https://leetcode.com/problems/n-queens/description/
Search
Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
51. N-Queens
Solved
Hard
Topics
Companies
The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.
Given an integer n, return all distinct solutions to the n-queens puzzle. You may return the answer in any order.
Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.
Example 1:
12.8K
116
126 Online
Code
Python
Auto
class Solution:
def solveNQueens(self, n):
result = []
def backtrack(row, cols, diag1, diag2, current_board):
if row == n:
result.append(["".join(row) for row in current_board])
return
for col in range(n):
if col in cols or (row - col) in diag1 or (row + col) in diag2:
continue
cols.add(col)
diag1.add(row - col)
diag2.add(row + col)
current_board[row][col] = 'Q'
backtrack(row + 1, cols, diag1, diag2, current_board)
cols.remove(col)
diag1.remove(row - col)
diag2.remove(row + col)
current_board[row][col] = '.'
current_board = [["." for _ in range(n)] for _ in range(n)]
cols = set()
diag1 = set()
diag2 = set()
backtrack(0, cols, diag1, diag2, current_board)
return result
Saved
Ln 25, Col 22
Testcase
Test Result
Accepted
Runtime: 0 ms

P.Saipriya
221CS020404
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 (Quick sort, Merge Sort)?
2. How do the characteristics of the data (e.g. range of prices, product name lengths) impact the choice of sorting algorithm?

1) Time and space complexities of common sorting Algorithms.

Quick sort:

Time complexity (Best) : $O(n \log n)$.
Time complexity (Average) : $O(n \log n)$
Time complexity (worst) : $O(n^2)$ (unbalanced pivot)
Space Complexity : $O(\log n)$ auxiliary

Merge Sort:

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 (eg., product prices), counting sort or valix sort may be more efficient than comparison-based algorithms, achieving linear time complexity O(n+k)
- For a large or arbitary range, comparison-based algorithms like Quick Sort or Merge sort are prefered.

2. product Name Lengths (String Sorting):

- Lexicographic sorting involves comparing strings character by character. Algorithms like Quick sort and Merge sort still perform well but may be slower due to increased comparison 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: Quick sort performs well with randomized pivot selection strategies.
- Nearly sorted Data: Inserting sort (or) Trim sort may out perform other due to better handling of such input.

4. Stability Requirements:

- Merge sort is stable, meaning equal elements retain their relative order. Quick sort 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 Merge sort