NAME- Ashish kumar Singh

CLASS- 22BCS-IOT-614/B

UID – 22BCS16892

SUBJECT- AP LAB

QUESTION -1:

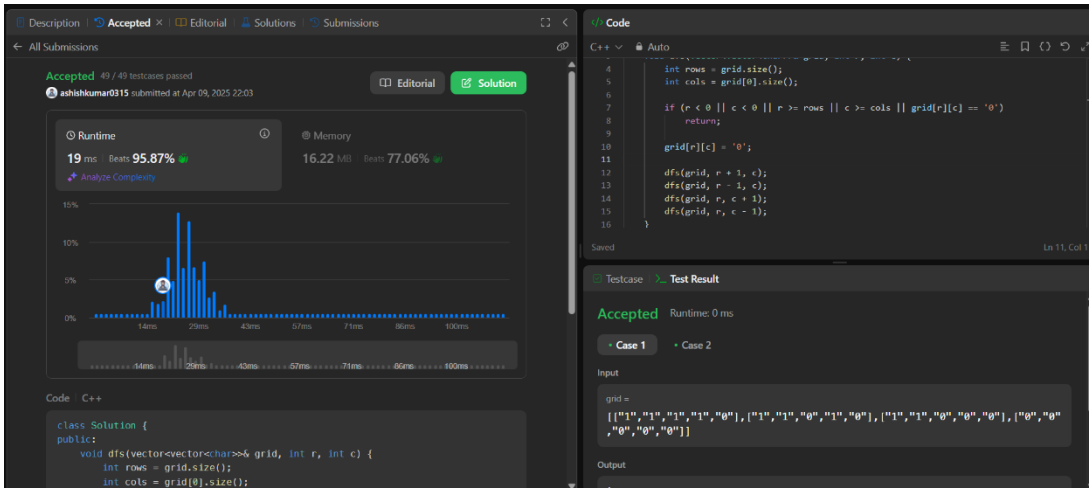
200. Number of Islands

CODE:

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int r, int c) {
        int rows = grid.size();
        int cols = grid[0].size();
        if (r < 0 || c < 0 || r >= rows || c >= cols || grid[r][c] == '0')
            return;
        grid[r][c] = '0';
        dfs(grid, r + 1, c);
        dfs(grid, r - 1, c);
        dfs(grid, r, c + 1);
        dfs(grid, r, c - 1);
    }
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty()) return 0;
        int rows = grid.size();
        int cols = grid[0].size();
        int count = 0;
        for (int r = 0; r < rows; ++r) {
            for (int c = 0; c < cols; ++c) {
                if (grid[r][c] == '1') {
                    ++count;
```

```
                dfs(grid, r, c);
            }   }   }
 return count;
  }};
```

OUTPUT:



QUESTION -2:

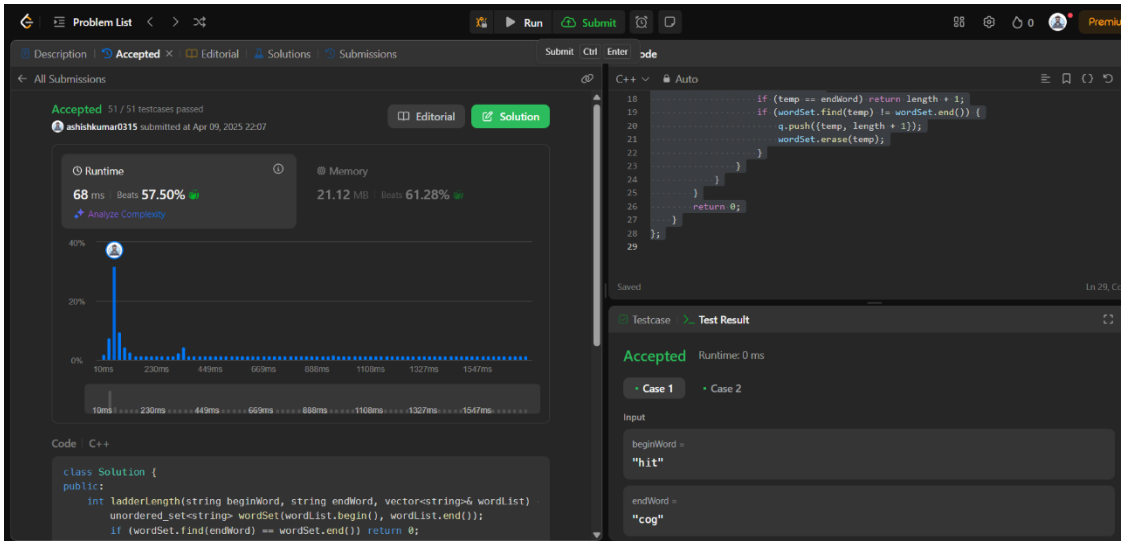
127. Word Ladder

CODE:

```
class Solution {
public:
   int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
      unordered_set<string> wordSet(wordList.begin(), wordList.end());
      if (wordSet.find(endWord) == wordSet.end()) return 0;

      queue<pair<string, int>> q;
      q.push({beginWord, 1});

      while (!q.empty()) {
         auto [word, length] = q.front();
         q.pop();
```

```
        for (int i = 0; i < word.length(); ++i) {
            string temp = word;
            for (char c = 'a'; c <= 'z'; ++c) {
                temp[i] = c;
                if (temp == endWord) return length + 1;
                if (wordSet.find(temp) != wordSet.end()) {
                    q.push({temp, length + 1});
                    wordSet.erase(temp);
                }}}}
        return 0;
    }
};
```

OUTPUT:



QUESTION -3:

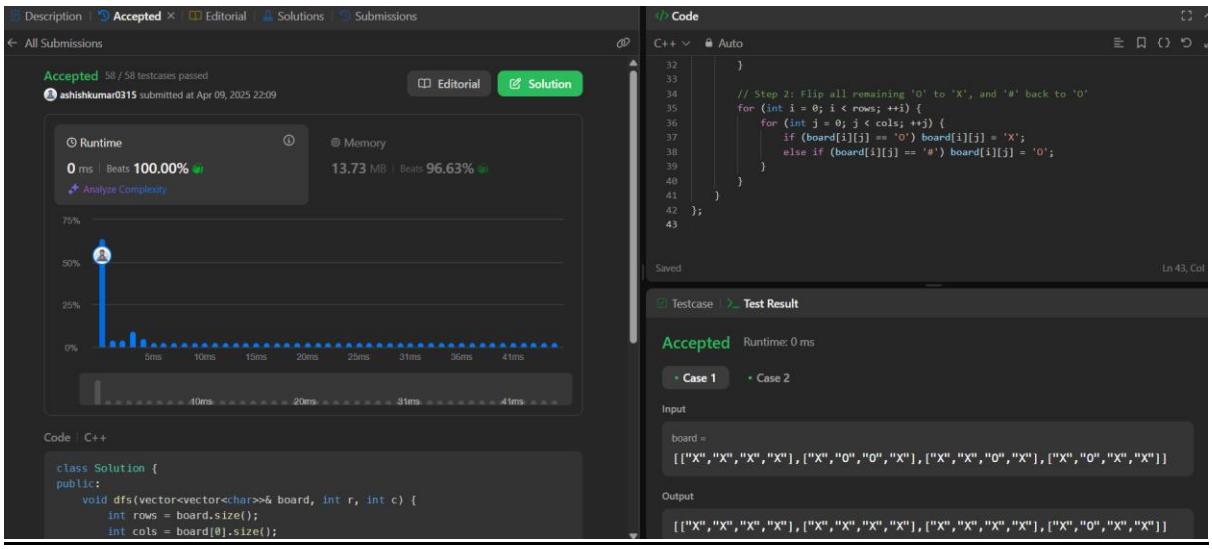
130. Surrounded Regions

CODE:

```
class Solution {
public:
    void dfs(vector<vector<char>>& board, int r, int c) {
        int rows = board.size();
```

```
    int cols = board[0].size();
    if (r < 0 || c < 0 || r >= rows || c >= cols || board[r][c] != 'O') return;
    board[r][c] = '#'; // Mark as visited and safe
    dfs(board, r + 1, c);
    dfs(board, r - 1, c);
    dfs(board, r, c + 1);
    dfs(board, r, c - 1);
}
void solve(vector<vector<char>>& board) {
    if (board.empty()) return;
    int rows = board.size();
    int cols = board[0].size();
    // Step 1: Mark 'O's connected to the border with '#'
    for (int i = 0; i < rows; ++i) {
        if (board[i][0] == 'O') dfs(board, i, 0);
        if (board[i][cols - 1] == 'O') dfs(board, i, cols - 1);
    }
    for (int j = 0; j < cols; ++j) {
        if (board[0][j] == 'O') dfs(board, 0, j);
        if (board[rows - 1][j] == 'O') dfs(board, rows - 1, j);
    }
    // Step 2: Flip all remaining 'O' to 'X', and '#' back to 'O'
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            if (board[i][j] == 'O') board[i][j] = 'X';
            else if (board[i][j] == '#') board[i][j] = 'O';
        }}}};
```

OUTPUT:



QUESTION -4:

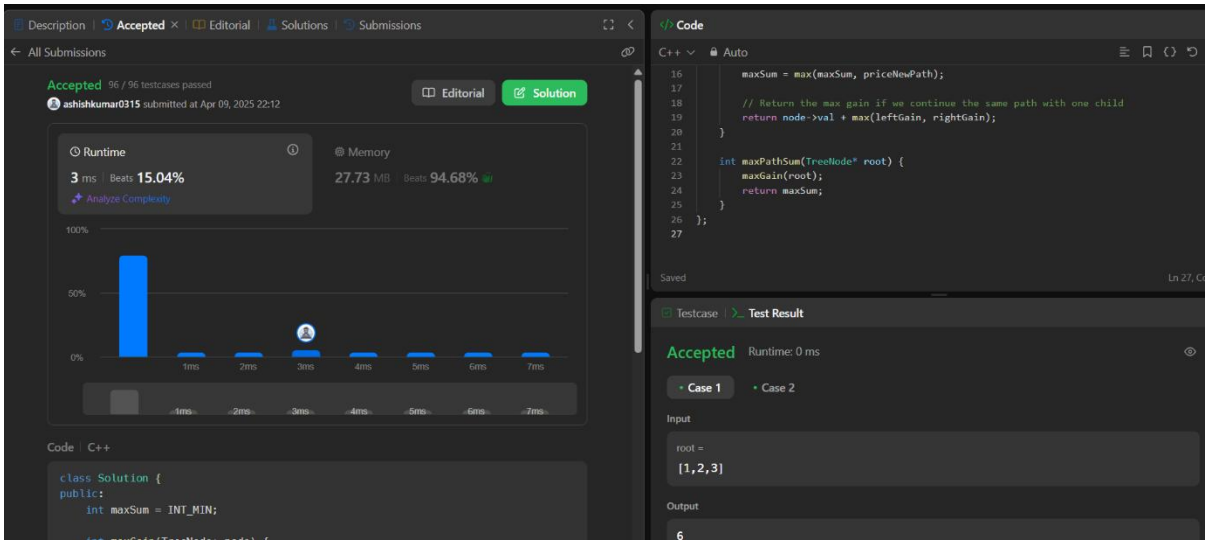
124. Binary Tree Maximum Path Sum

CODE:

```
class Solution {
public:
    int maxSum = INT_MIN;

    int maxGain(TreeNode* node) {
        if (!node) return 0;

        // Recursively get the max gain from left and right subtrees
        int leftGain = max(maxGain(node->left), 0);
        int rightGain = max(maxGain(node->right), 0);

        // Price of the new path that passes through the current node
        int priceNewPath = node->val + leftGain + rightGain;

        // Update global maxSum if the new path is better
        maxSum = max(maxSum, priceNewPath);
```

```
        // Return the max gain if we continue the same path with one child
        return node->val + max(leftGain, rightGain);
    }

    int maxPathSum(TreeNode* root) {
        maxGain(root);
        return maxSum;
    }
};
```

OUTPUT:



QUESTION -5:

547. Number of Provinces

CODE:

```
class Solution {
public:
    void dfs(vector<vector<int>>& isConnected, vector<bool>& visited, int city) {
        visited[city] = true;
        for (int i = 0; i < isConnected.size(); ++i) {
            if (isConnected[city][i] == 1 && !visited[i]) {
                dfs(isConnected, visited, i);
            }
```

```
    }}
int findCircleNum(vector<vector<int>>& isConnected) {
    int n = isConnected.size();
    vector<bool> visited(n, false);
    int provinces = 0;
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            ++provinces;
            dfs(isConnected, visited, i);
        }
    }
    return provinces;
  }
};
```

OUTPUT: