



Discover. Learn. Empower.

ENGINEERING

Fast Learner Complex Problems

Student Name: Meenansh Gupta

UID: 22BCS16380

Branch: CSE

Section/Group: NTPP-603-B

Semester: 6th

Date of Performance: 31/03/25

Subject Name: AP-2

Subject Code: 22CSP-351

***Aim(i):** Set Matrix Zeroes: Given an $m \times n$ matrix, if an element is 0, set its entire row and column to 0.*

Source Code:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int rows = matrix.size();
        int cols = matrix[0].size();

        bool firstRowHasZero = false;
        bool firstColHasZero = false;

        // Check if the first row contains zero
        for (int c = 0; c < cols; c++) {
            if (matrix[0][c] == 0) {
                firstRowHasZero = true;
                break;
            }
        }

        // Check if the first column contains zero
        for (int r = 0; r < rows; r++) {
            if (matrix[r][0] == 0) {
                firstColHasZero = true;
                break;
            }
        }
    }
};
```

```

    }
}

// Use the first row and column as markers
for (int r = 1; r < rows; r++) {
    for (int c = 1; c < cols; c++) {
        if (matrix[r][c] == 0) {
            matrix[r][0] = 0;
            matrix[0][c] = 0;
        }
    }
}

// Set the marked rows to zero
for (int r = 1; r < rows; r++) {
    if (matrix[r][0] == 0) {
        for (int c = 1; c < cols; c++) {
            matrix[r][c] = 0;
        }
    }
}

// Set the marked columns to zero
for (int c = 1; c < cols; c++) {
    if (matrix[0][c] == 0) {
        for (int r = 1; r < rows; r++) {
            matrix[r][c] = 0;
        }
    }
}

// Set the first row to zero if needed
if (firstRowHasZero) {
    for (int c = 0; c < cols; c++) {
        matrix[0][c] = 0;
    }
}

// Set the first column to zero if needed
if (firstColHasZero) {
    for (int r = 0; r < rows; r++) {
        matrix[r][0] = 0;
    }
}
};

```

OUTPUT:

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
matrix =  
[[1,1,1],[1,0,1],[1,1,1]]
```

Output

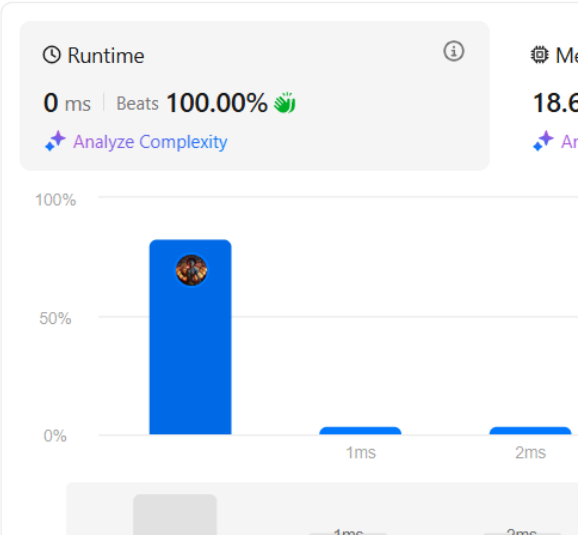
```
[[1,0,1],[0,0,0],[1,0,1]]
```

Expected

```
[[1,0,1],[0,0,0],[1,0,1]]
```

Contribute a testcase

Accepted 202 / 202 testcases passed
Meenansh_16380 submitted at Mar 31, 2025 16:14



Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

```
matrix =  
[[0,1,2,0],[3,4,5,2],[1,3,1,5]]
```

Output

```
[[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

Expected

```
[[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

Aim(ii): *Longest Substring Without Repeating Characters: Given a string s, find the length of the longest substring that does not contain any repeating characters*

Source Code:

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.length();
        int maxLength = 0;
        unordered_set<char> charSet;
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (charSet.count(s[right]) == 0) {
                charSet.insert(s[right]);
                maxLength = max(maxLength, right - left + 1);
            } else {
                while (charSet.count(s[right])) {
                    charSet.erase(s[left]);
                    left++;
                }
                charSet.insert(s[right]);
            }
        }

        return maxLength;
    }
};
```

OUTPUT:

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

S =
"abcabcbb"

Output

3

Expected

3

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

S =
"bbbbbb"

Output

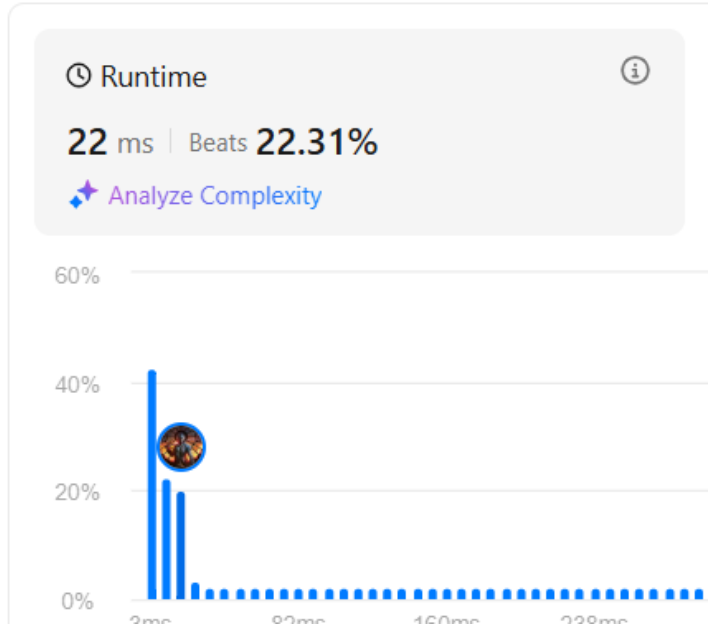
1

Expected

1

Accepted 987 / 987 testcases passed

Meenansh_16380 submitted at Mar 31, 2025 16:17



Aim(iii): *Reverse Linked List II: Given the head of a singly linked list and two integers left and right, reverse the nodes of the list from position left to right.*

Source Code:

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) {
            return head;
        }

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prev = dummy;

        for (int i = 0; i < left - 1; i++) {
            prev = prev->next;
        }

        ListNode* cur = prev->next;

        for (int i = 0; i < right - left; i++) {
            ListNode* temp = cur->next;
            cur->next = temp->next;
            temp->next = prev->next;
            prev->next = temp;
        }

        return dummy->next;
    }
};
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

head =
[1,2,3,4,5]

left =
2

right =
4

Output

[1,4,3,2,5]

Expected

[1,4,3,2,5]

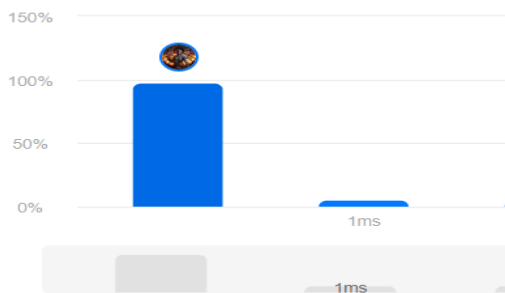
Accepted 44 / 44 testcases passed

Meenansh_16380 submitted at Mar 31, 2025 16:21

Runtime

0 ms | Beats 100.00%

Analyze Complexity



Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

head =
[5]

left =
1

right =
1

Output

[5]

Expected

[5]

Aim(iv): *Detect a Cycle in a Linked List: Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.*

Source Code:

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head;
        ListNode *fast = head;

        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return true; // Cycle detected
            }
        }
        return false; // No cycle
    }
};
```

OUTPUT:

Accepted Runtime: 5 ms

Accepted Runtime: 5 ms

• Case 1 • Case 2 • Case 3 • Case 1 • Case 2 • Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

true

Expected

true

Input

head =
[1,2]

pos =
0

Output

true

Expected

true

Accepted 29 / 29 testcases passed

Meenansh_16380 submitted at Mar 31, 2025 16:25

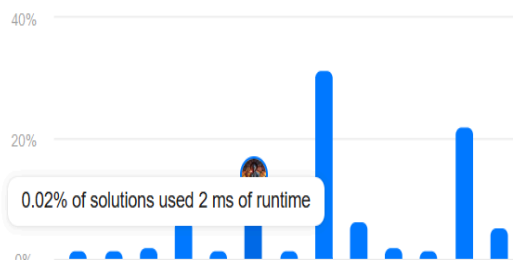
Runtime

6 ms | Beats 90.76%

Analyze Complexity

Memory

11.77 MB | Be



Aim(v): *The Skyline Problem: Given a list of buildings represented as [left, right, height], where each building is a rectangle, return the key points of the skyline. A key point is represented as [x, y], where x is the xcoordinate where the height changes to y.*

Source Code:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int,int>>line;
        for(auto it:buildings){
            line.push_back({it[0],-it[2]});
            line.push_back({it[1],it[2]});
        }
        sort(line.begin(),line.end(),[&](pair<int,int>&a,pair<int,int>&b){
            if(a.first==b.first){
                return a.second<b.second;
            }
            return a.first<b.first;
        });
        multiset<int>st;
        vector<vector<int>>res;
        int curr=0;
        for(auto it:line){
            int ht=it.second;
            if(ht<0){
                st.insert(-1*ht);
            }else if(ht>=0){
                st.erase(st.find(ht));
            }
            int new_ht=st.empty()?0:*st.rbegin();
            if(new_ht!=curr){
                curr=new_ht;
                res.push_back({it.first,new_ht});
            }
        }
        return res;
    }
};
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

```
buildings =  
[[0, 2, 3], [2, 5, 3]]
```

Output

```
[[0, 3], [5, 0]]
```

Expected

```
[[0, 3], [5, 0]]
```

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

```
buildings =  
[[2, 9, 10], [3, 7, 15], [5, 12, 12], [15, 20, 10], [19, 24, 8]]
```

Output

```
[[2, 10], [3, 15], [7, 12], [12, 0], [15, 10], [20, 8], [24, 0]]
```

Expected

```
[[2, 10], [3, 15], [7, 12], [12, 0], [15, 10], [20, 8], [24, 0]]
```

Accepted 44 / 44 testcases passed

 Meenansh_16380 submitted at Mar 31, 2025 16:28

🕒 Runtime

ⓘ

24 ms | Beats 35.39%

🚀 [Analyze Complexity](#)



Aim(vi): *Longest Increasing Subsequence II: Given an integer array nums, find the length of the longest strictly increasing subsequence. A subsequence is derived from the array by deleting some or no elements without changing the order of the remaining elements.*

Source Code:

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        vector<int> res;

        for (int n : nums) {
            if (res.empty() || res.back() < n) {
                res.push_back(n);
            } else {
                int idx = binarySearch(res, n);
                res[idx] = n;
            }
        }

        return res.size();
    }

private:
    int binarySearch(const vector<int>& arr, int target) {
        int left = 0;
        int right = arr.size() - 1;

        while (left <= right) {
            int mid = (left + right) / 2;
            if (arr[mid] == target) {
                return mid;
            } else if (arr[mid] > target) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
};
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

```
nums =  
[10,9,2,5,3,7,101,18]
```

Output

4

Expected

4

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

```
nums =  
[0,1,0,3,2,3]
```


Output


4


Expected

4

Accepted 55 / 55 testcases passed

 Meenansh_16380 submitted at Mar 31, 2025 16:31

 Editorial

 Solution

🕒 Runtime

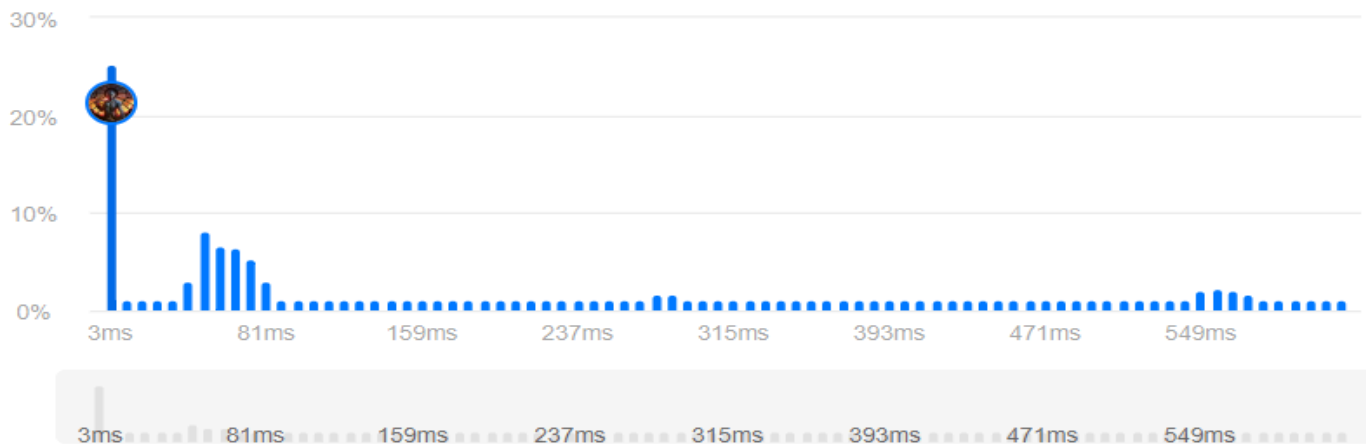


4 ms | Beats 77.46% 🌿

🔍 Analyze Complexity

💾 Memory

13.92 MB | Beats 94.26% 🌿



Aim(vii): *Search a 2D Matrix II: Given an $m \times n$ matrix where each row is sorted in ascending order from left to right and each column is sorted in ascending order from top to bottom, and an integer target, determine if the target exists in the matrix.*

Source Code:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = m ? matrix[0].size() : 0, r = 0, c = n - 1;
        while (r < m && c >= 0) {
            if (matrix[r][c] == target) {
                return true;
            }
            matrix[r][c] > target ? c-- : r++;
        }
        return false;
    }
};
```

OUTPUT:

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

```
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
```

```
target =
5
```

Output

```
true
```

Expected

```
true
```

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

```
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
```

```
target =
20
```

Output

```
false
```

Expected

```
false
```

Aim(viii): *Word Break: Given a string s and a dictionary $wordDict$ containing a list of words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. The same word can be reused multiple times.*

Source Code:

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        vector<bool> dp(s.size() + 1, false);
        dp[0] = true;

        for (int i = 1; i <= s.size(); i++) {
            for (const string& w : wordDict) {
                int start = i - w.length();
                if (start >= 0 && dp[start] && s.substr(start, w.length()) == w) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.size()];
    }
};
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

$s =$
"leetcode"

$wordDict =$
["leet","code"]

Output

true

Expected

true

Accepted

Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

$s =$
"applepenapple"

$wordDict =$
["apple","pen"]

Output

true

Expected

true

Aim(ix): *Longest Increasing Path in a Matrix: Given an $m \times n$ integer matrix, find the length of the longest strictly increasing path. You can move up, down, left, or right from each cell. Diagonal moves and moves outside the boundaries are not allowed*

Source Code:

```
class Solution {
public:
    int dfs(int row, int col, vector<vector<int>>& matrix, vector<vector<int>>& dp){
        int m = matrix.size(), n = matrix[0].size();
        int ans = 1;
        if(dp[row][col] != -1) return dp[row][col];
        for(int tempR=-1; tempR<=1; tempR++){
            for(int tempC=-1; tempC<=1; tempC++){
                if(abs(tempR) + abs(tempC) != 1) continue;
                int Nrow = row + tempR;
                int Ncol = col + tempC;
                if(Nrow>=0 && Nrow<m && Ncol>=0 && Ncol<n && matrix[Nrow][Ncol] > matrix[row][col]){
                    ans = max(ans, 1+dfs(Nrow, Ncol, matrix, dp));
                }
            }
        }
        return dp[row][col] = ans;
    }
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        int m = matrix.size(), n = matrix[0].size();
        int result = 0;
        vector<vector<int>> dp(m, vector<int>(n, -1));
        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                result = max(result, dfs(i, j, matrix, dp));
            }
        }
        return result;
    }
};
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

```
matrix =  
[[9,9,4],[6,6,8],[2,1,1]]
```

Output

4

Expected

4

Accepted

Runtime: 0 ms

• Case 1

• Case 2

• Case 3

Input

```
matrix =  
[[3,4,5],[3,2,6],[2,2,1]]
```

Output

4

Expected

4

Accepted 139 / 139 testcases passed

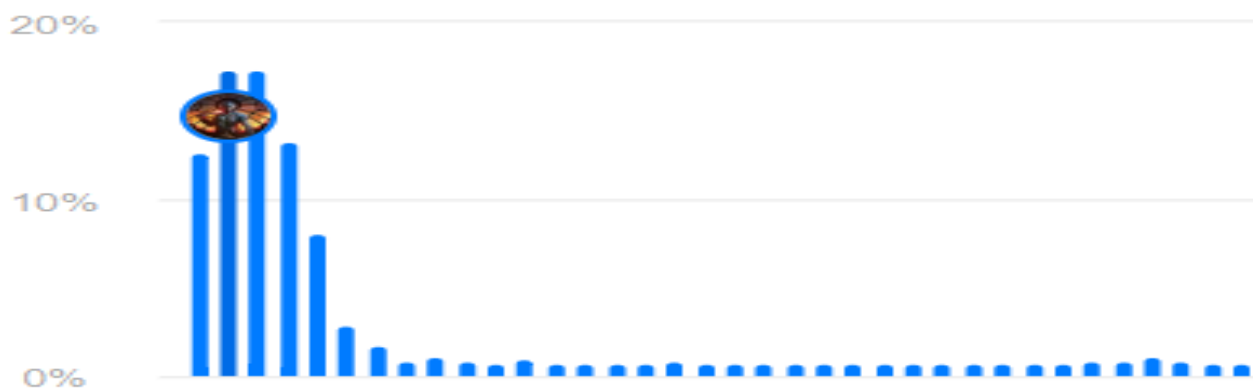
 Meenansh_16380 submitted at Mar 31, 2025 16:40

🕒 Runtime



8 ms | Beats 78.50% 🏆

🔗 Analyze Complexity



Aim(x): *Trapping Rain Water: Given n non-negative integers representing an elevation map where the width of each bar is 1, compute the total amount of water that can be trapped after raining.*

Source Code:

```
class Solution {
public:
    int trap(vector<int>& height) {
        int left = 0;
        int right = height.size() - 1;
        int leftMax = height[left];
        int rightMax = height[right];
        int water = 0;

        while (left < right) {
            if (leftMax < rightMax) {
                left++;
                leftMax = max(leftMax, height[left]);
                water += leftMax - height[left];
            } else {
                right--;
                rightMax = max(rightMax, height[right]);
                water += rightMax - height[right];
            }
        }

        return water;
    }
};
```

OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

height =
[0,1,0,2,1,0,1,3,2,1,2,1]

Output

6

Expected

6

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

height =
[4,2,0,3,2,5]

Output

9

Expected

9

