Spiral Matrix

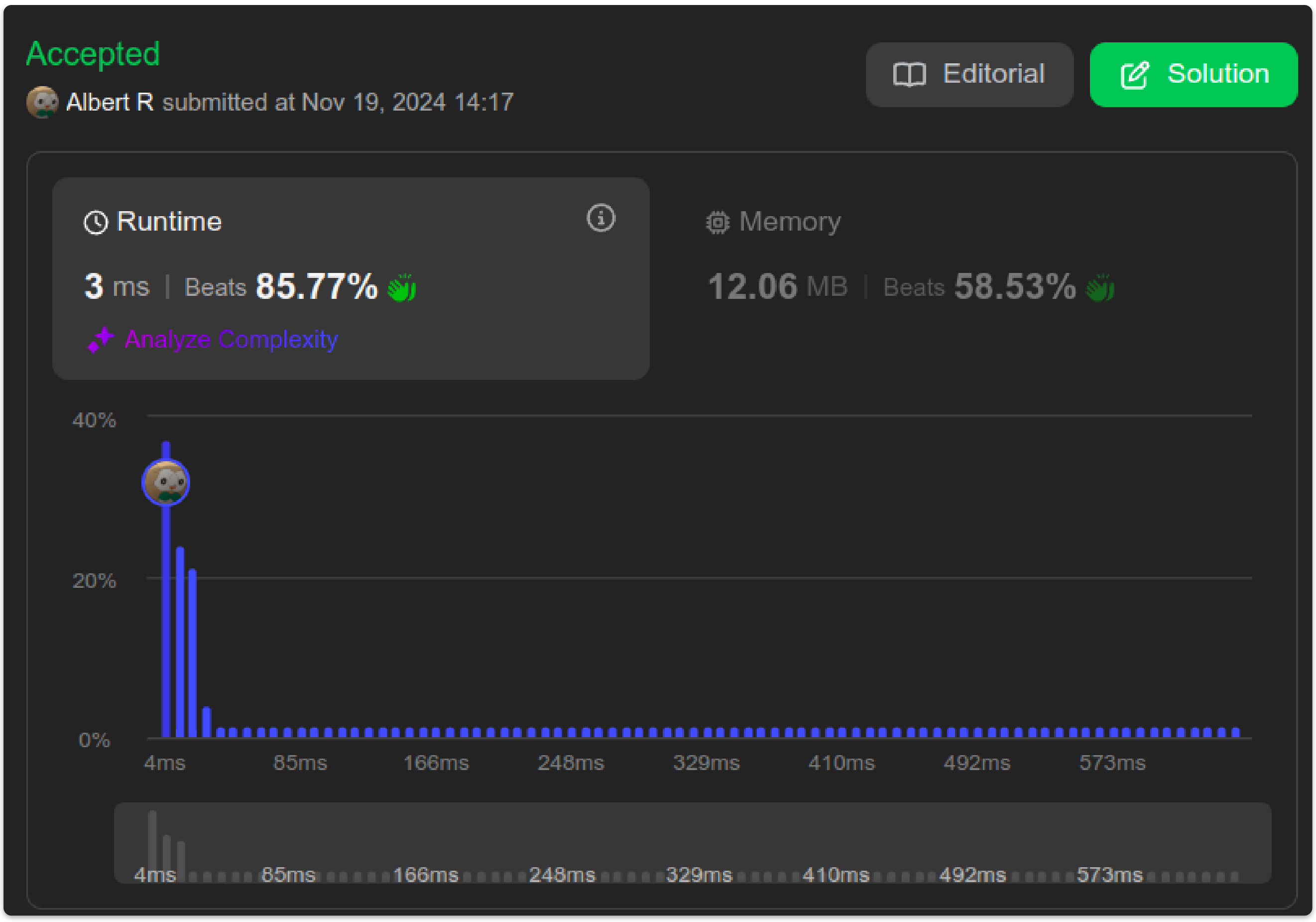
```cpp
#include<bits/stdc++.h>
using namespace std;

#define FASTIO ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0)

int main(){
    FASTIO;
    int m,n; cin>>m>>n;
    vector<vector<int>> matrix (m,vector(n,0));
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            cin>>matrix[i][j];
        }
    }
    vector<int> res;

    int i=0,j=0;

    while (i<=m/2 && j<=n/2){
        int flag=0;
        for (int k=j;k<=(n-j-1);k++){
            flag=1;
            res.push_back(matrix[i][k]);
        }
        if (flag!=1) break;
        for (int k=i+1;k<=(m-i-1);k++){
            flag=2;
            res.push_back(matrix[k][n-j-1]);
        }
        if (flag!=2 || j == (n-j-1)) break;
        for (int k=(n-j-1)-1;k>=j;k--){
            flag=3;
            res.push_back(matrix[m-i-1][k]);
        }
        if (flag!=3 || i == (m-i-1)) break;
        for (int k=(m-i-1)-1;k>i;k--){
            flag=4;
            res.push_back(matrix[k][j]);
        }
        if (flag!=4) break;
        i++;j++;
    }

    for (int i: res) cout<<i<<' ';
    return 1;
}
```

Time Complexity: $O(M * N)$
Space Complexity: $O(1)$

Longest Substring without repeating characters

Sliding Window

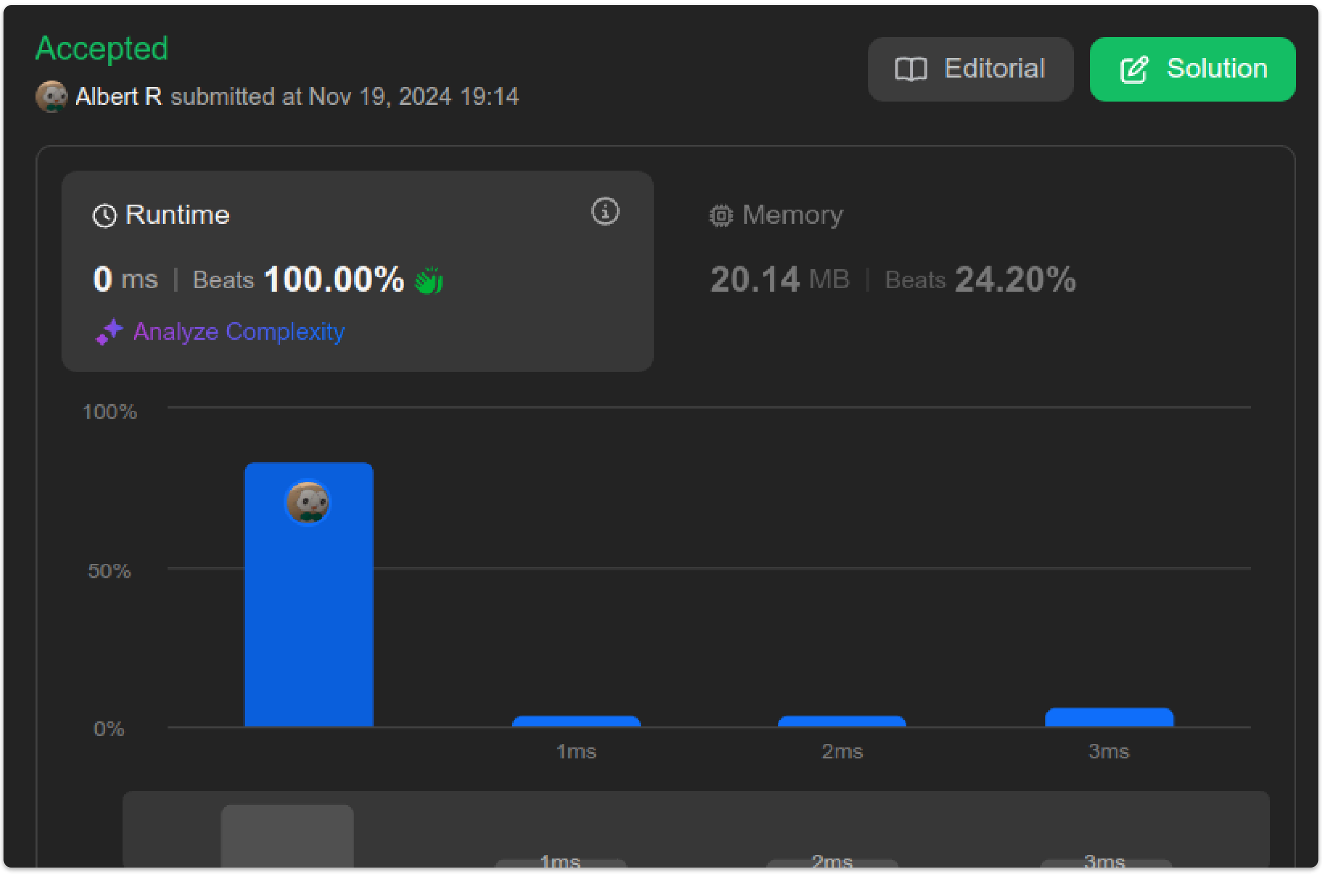
```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        int n = s.size();
        if (n==0) return 0;
        int l=0, r=0, mx=0;
        vector<int> ctr (256,0);
        while (r<n){
            if (l<r && ctr[s[r]]) {mx=max(mx,(r-l));ctr[s[l]]--;l++;}
            else {
                ctr[s[r]]++;
                r++;
            }
        }
        mx = max(mx,r-l);
        return mx;
    }
};
```



Time Complexity: $O(N)$
Space Complexity: $O(1)$ (256 Unicode array)

Remove linked list elements

```
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        if (head==nullptr) return head;
        while (head->val == val) {
            if (head->next==nullptr) return nullptr;
            ListNode* temp = new ListNode(head->next->val, head->next-
>next);
            head = temp;
        }
        if (head==nullptr || head->next==nullptr) return head;
        ListNode *curr = head->next, *prev=head;
        while (curr->next){
            //cout<<(prev->val)<<' '<<(curr->val)<<endl;
            if (curr->val == val){
                prev->next = curr->next;
            }
            else prev = prev -> next;
            curr = curr->next;
        }
        if (curr->val == val) prev -> next = nullptr;
        return head;
    }
};
```



Accepted

Albert R submitted at Nov 19, 2024 19:14

Runtime

0 ms | Beats 100.00%

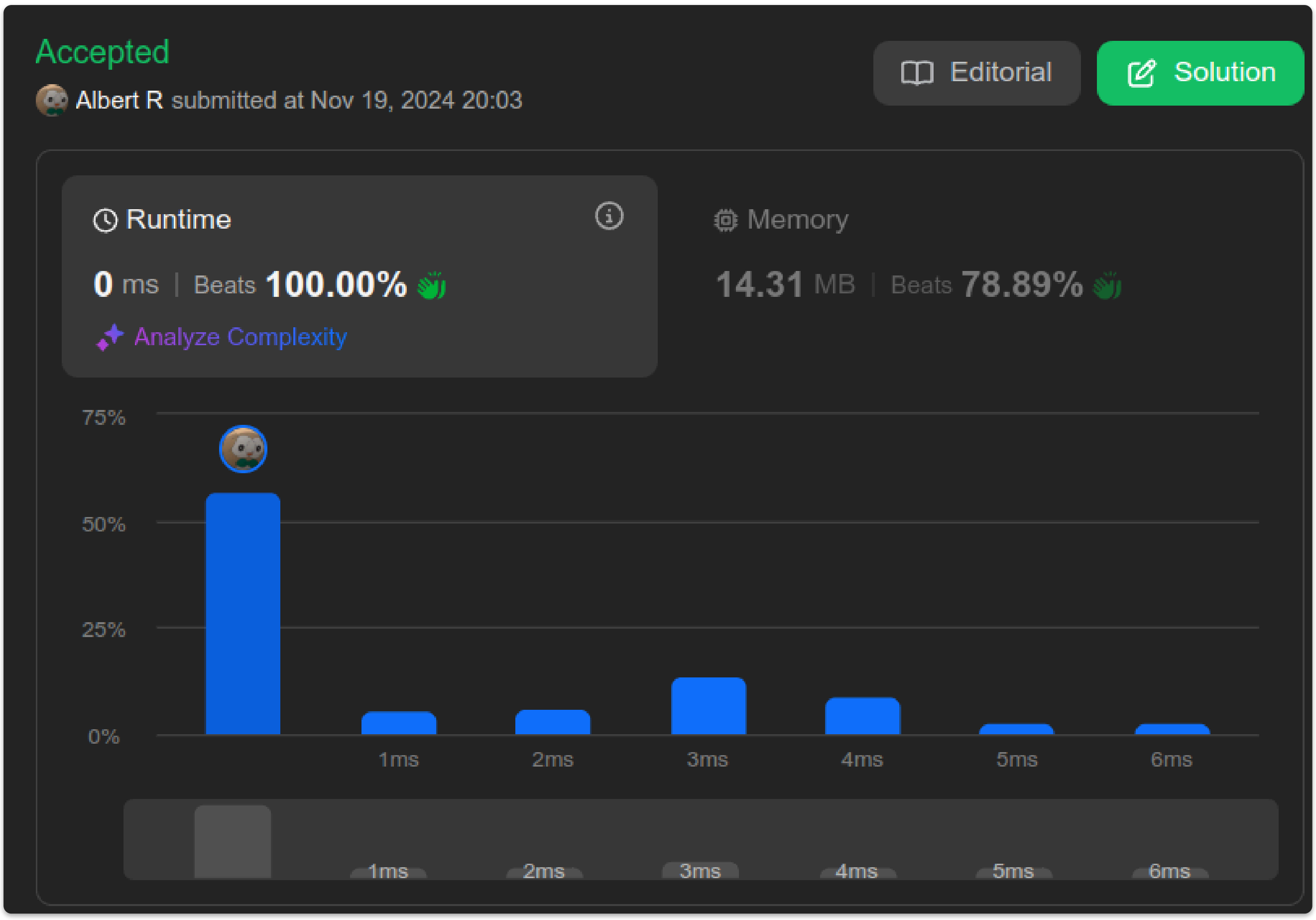
Memory

20.14 MB | Beats 24.20%

Time Complexity: $O(N)$
Space Complexity: $O(1)$

Minimum path sum

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        ios_base::sync_with_stdio(false); cin.tie(0); cout.tie(0);
        int m = grid.size(), n=grid[0].size();
        vector<int> dp (n,8e6);
        for (int i=0; i<m; i++){
            for (int j=0; j<n; j++){
                if (i==0 && j==0) dp[0] = grid[0][0];
                else {
                    if (j==0) dp[j]+=grid[i][j];
                    else dp[j] = min(dp[j-1], dp[j])+grid[i][j];
                }
            }
            cout<<endl;
        }
        return dp[n-1];
    }
};
```


Accepted

Albert R submitted at Nov 19, 2024 20:03

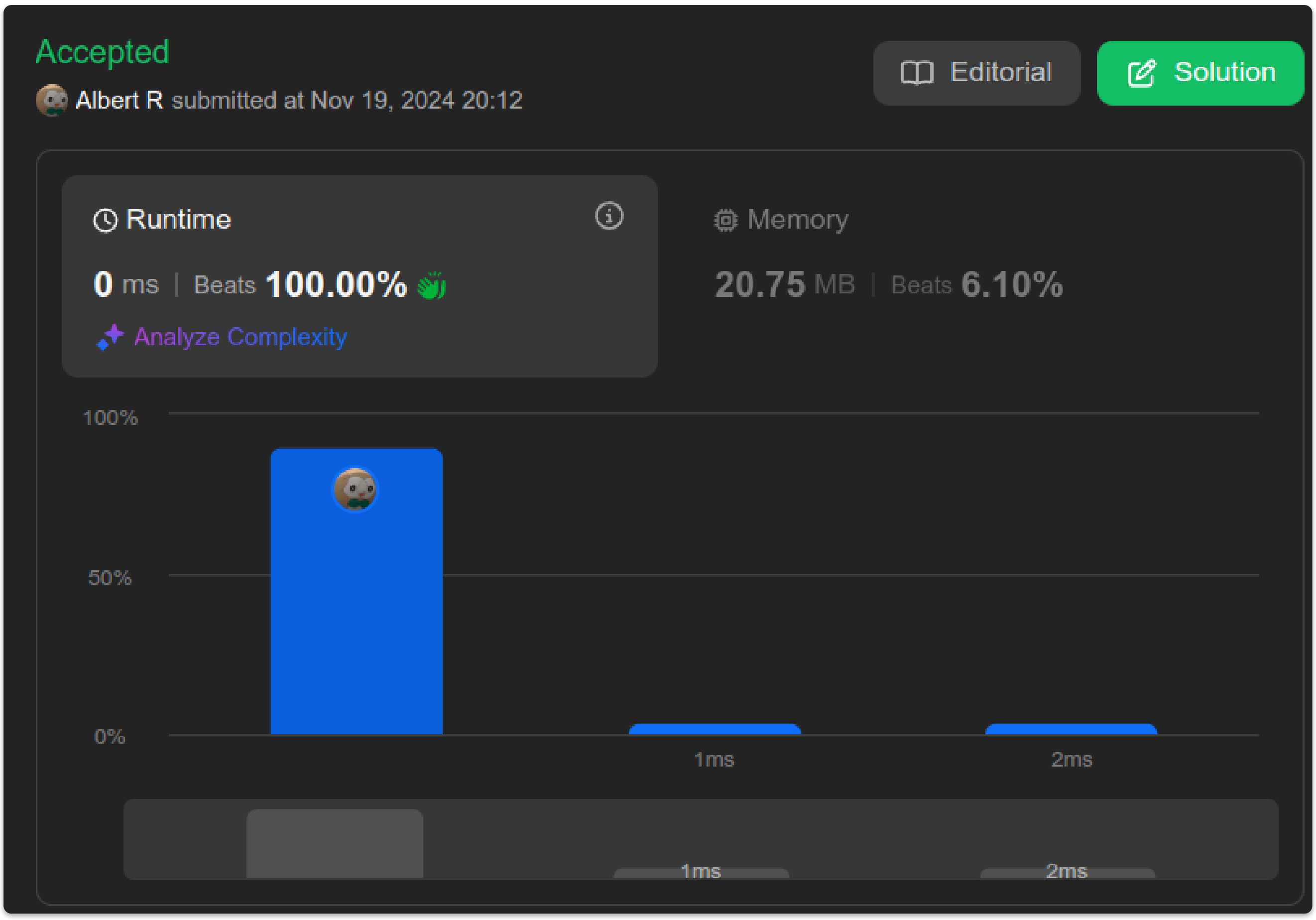
0 ms | Beats 100.00%

14.31 MB | Beats 78.89%

Time Complexity: $O(m*n)$
Space Complexity: $O(n)$

Validate Binary Search Tree

```
class Solution {
public:
    void inOrder(vector<int> &arr, TreeNode* root){
        if (root==nullptr) return;
        inOrder(arr,root->left);
        arr.push_back(root->val);
        inOrder(arr,root->right);

    }
    bool isValidBST(TreeNode* root) {
        vector<int> vsa;
        inOrder(vsa,root);
        for (int i=1; i<vsa.size(); i++){
            if (vsa[i-1]>=vsa[i]) return false;
        }
        return true;
    }
};
```

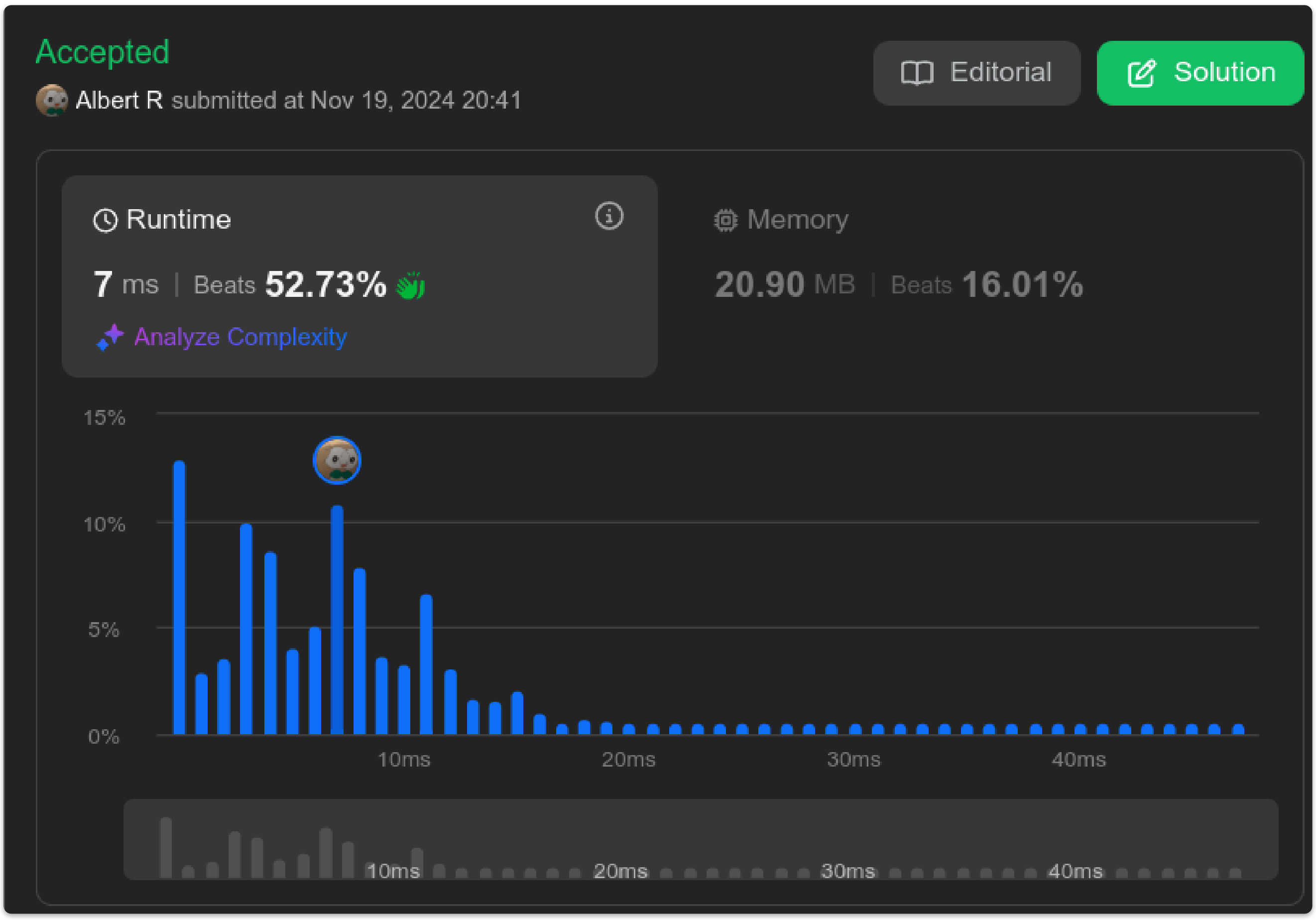

Accepted

Albert R submitted at Nov 19, 2024 20:12

Time Complexity: $O(N)$
Space Complexity: $O(N)$ (inorder sorted array)

Course Schedule

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<int> indegree (numCourses,0);
        unordered_map<int,list<int>> adjlist;

        for (int _ = 0; _<numCourses; _++){
            list<int> f;
            adjlist[_] = f;
        }

        for (int i=0;i<prerequisites.size();i++){
            vector<int> edge = prerequisites[i];
            adjlist[edge[0]].push_back(edge[1]);
            indegree[edge[1]] +=1;
        }

        queue<int> q;
        for (int i=0;i<indegree.size();i++){
            if (indegree[i] ==0) q.push(i);
        }
        int count = 0;

        while (!q.empty()){
            int u = q.front();
            q.pop();

            list<int>::iterator x;
            for (x = adjlist[u].begin();x!=adjlist[u].end();x++){
                if (--indegree[*x] ==0){
                    q.push(*x);
                }
            }
            count++;
        }

        return (count==numCourses)?true:false;
    }
};
```


Accepted

Albert R submitted at Nov 19, 2024 20:41

Runtime

7 ms | Beats 52.73%

Memory

20.90 MB | Beats 16.01%

Time Complexity: $O(n * n)$
Space Complexity: $O(n)$