gate dsa practice questions SET1

NAME:SHREENIDHI S

DATE-9/11/24

**1.MAXIMUM SUBARRAY(KADANE’S ALGORITHM)**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxSubArraySum(vector<int>& arr) {

    int sum = 0;

    int maxi = arr[0];

    int n = arr.size();

    for (int i = 0; i < n; i++) {

        sum += arr[i];

        maxi = max(maxi, sum);

        if (sum < 0) {

            sum = 0;

        }

    }

    return maxi;

}

int main() {

    vector<int> arr1 = {2, 3, -8, 7, -1, 2, 3};

    vector<int> arr2 = {-2, -4};

    vector<int> arr3 = {5, 4, 1, 7, 8};

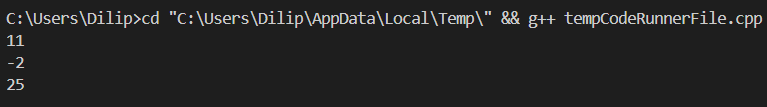
    cout << maxSubArraySum(arr1) << endl;

    cout << maxSubArraySum(arr2) << endl;

    cout << maxSubArraySum(arr3) << endl;

    return 0;

}



TC- O(N)

**2.MAXIMUM PRODUCT SUM**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxProductSubarray(vector<int>& arr) {

    int maxProduct = arr[0];

    int minProduct = arr[0];

    int result = arr[0];

    int n = arr.size();

    for (int i = 1; i < n; i++) {

        if (arr[i] < 0) {

            swap(maxProduct, minProduct);

        }

        maxProduct = max(arr[i], maxProduct \* arr[i]);

        minProduct = min(arr[i], minProduct \* arr[i]);

        result = max(result, maxProduct);

    }

    return result;

}

int main() {

    vector<int> arr1 = {-2, 6, -3, -10, 0, 2};

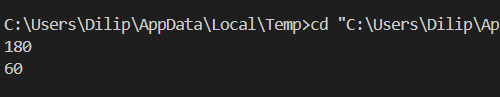
    vector<int> arr2 = {-1, -3, -10, 0, 60};

    cout << maxProductSubarray(arr1)<< endl;

    cout << maxProductSubarray(arr2)<< endl;

    return 0;

}



TC-O(N)

**3.SEARCH IN ROTATED SORTED ARRAY**

#include <iostream>

#include <vector>

using namespace std;

int searchInRotatedArray(vector<int>& arr, int key) {

    int n = arr.size();

    int low = 0, high = n - 1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {

            return mid;

        }

        if (arr[low] <= arr[mid]) {

            if (key >= arr[low] && key < arr[mid]) {

                high = mid-1;

            } else {

                low = mid+1;

            }

        }

        else {

            if (key > arr[mid] && key <= arr[high]) {

                low = mid+ 1;

            } else {

                high = mid-1;

            }

        }

    }

    return -1;

}

int main() {

    vector<int> arr1 = {4, 5, 6, 7, 0, 1, 2};

    vector<int> arr2 = {4, 5, 6, 7, 0, 1, 2};

    vector<int> arr3 = {50, 10, 20, 30, 40};

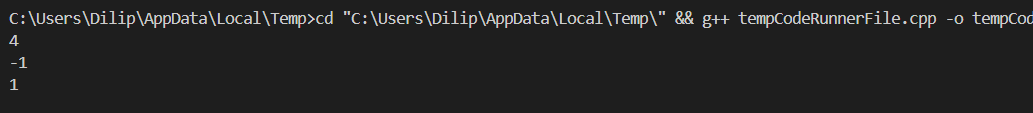
    cout << searchInRotatedArray(arr1, 0) << endl;  // Output: 4

    cout << searchInRotatedArray(arr2, 3) << endl;  // Output: -1

    cout << searchInRotatedArray(arr3, 10) << endl; // Output: 1

    return 0;

}



TC=O(log N)

**4.MAXIMUM WATER**

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

int MostWater(int arr[], int n) {

    int i = 0;

    int j = n - 1;

    int maxWater = INT\_MIN;

    while (i < j) {

        int water = (j - i) \* min(arr[i], arr[j]);

        maxWater = max(maxWater, water);

        if (arr[i] < arr[j]) {

            i++;

        } else {

            j--;

        }

    }

    return maxWater;

}

int main() {

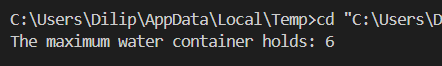
    int arr[] = {1, 5, 4, 3};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "The maximum water container holds: " << MostWater(arr, n) << endl;

    return 0;

}



TC- O(N)

**5.FACTORIAL OF A NUMBER**

#include <iostream>

#include <vector>

using namespace std;

void multiply(vector<int>& result, int number) {

    int carry = 0;

    int n=result.size();

    for (int i = 0; i <n; i++) {

        int product = result[i] \* number + carry;

        result[i] = product % 10;

        carry = product / 10;

    }

    while (carry) {

        result.push\_back(carry % 10);

        carry /= 10;

    }

}

void factorial(int n) {

    vector<int> result;

    result.push\_back(1);

    for (int i = 2; i <= n; i++) {

        multiply(result, i);

    }

    for (int i = n- 1; i >= 0; i--) {

        cout << result[i];

    }

    cout << endl;

}

int main() {

    int number;

    cout << "Enter a number:";

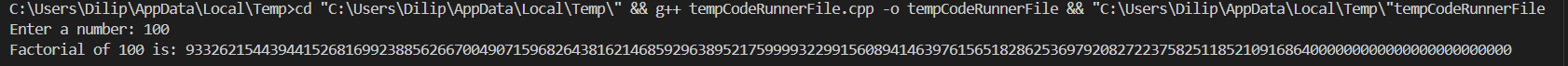
    cin >> number;

    cout << "Factorial of " << number << " is: ";

    factorial(number);

    return 0;

}



TC-O(N^2)

**6.TRAPPING RAIN WATER**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int trap(vector<int>& arr) {

    int n = arr.size();

    if (n == 0) return 0;

    int left = 0,

    int right = n - 1;

    int left\_max = 0,

    int right\_max = 0;

    int water\_trapped = 0;

    while (left <= right) {

        if (arr[left] <= arr[right]) {

            if (arr[left] >= left\_max) {

                left\_max = arr[left];

            } else {

                water\_trapped += left\_max - arr[left];

            }

            left++;

        }

         else {

            if (arr[right] >= right\_max) {

                right\_max = arr[right];

            } else {

                water\_trapped += right\_max - arr[right];

            }

            right--;

        }

    }

    return water\_trapped;

}

int main() {

    vector<int> arr1 = {3, 0, 1, 0, 4, 0, 2};

    vector<int> arr2 = {3, 0, 2, 0, 4};

    vector<int> arr3 = {1, 2, 3, 4};

    vector<int> arr4 = {10, 9, 0, 5};

    cout << "Trapped water in arr1: " << trap(arr1) << endl;

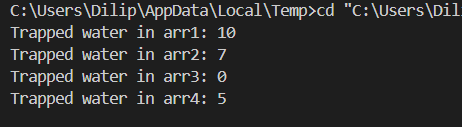
    cout << "Trapped water in arr2: " << trap(arr2) << endl;

    cout << "Trapped water in arr3: " << trap(arr3) << endl;

    cout << "Trapped water in arr4: " << trap(arr4) << endl;

    return 0;

}



Tc-o(N)

**7.chocolate distribution**

#include <iostream>

#include <algorithm>

#include <climits>

using namespace std;

int chocolateDistribution(int arr[], int n, int m) {

    if (m > n) {

        return -1;

    }

    sort(arr, arr + n);

    int min\_diff = INT\_MAX;

    for (int i = 0; i + m - 1 < n; i++) {

        int diff = arr[i + m - 1] - arr[i];

        min\_diff = min(min\_diff, diff);

    }

    return min\_diff;

}

int main() {

    int arr1[] = {7, 3, 2, 4, 9, 12, 56};

    int m1 = 3;

    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    cout << chocolateDistribution(arr1, n1, m1) << endl;

    int arr2[] = {7, 3, 2, 4, 9, 12, 56};

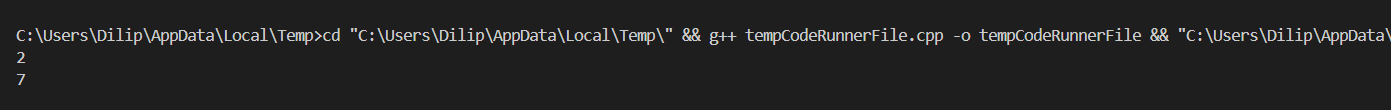
    int m2 = 5;

    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    cout <<  chocolateDistribution(arr2, n2, m2) << endl;

    return 0;

}



TC-O(nlogn)

**8.MERGE OVERLAPPING INTERVALS**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<vector<int>> mergeIntervals(vector<vector<int>>& intervals) {

    if (intervals.empty()) return {};

    sort(intervals.begin(), intervals.end());

    vector<vector<int>> merged;

    merged.push\_back(intervals[0]);

    for (int i = 1; i < intervals.size(); i++) {

        if (merged.back()[1] >= intervals[i][0]) {

            merged.back()[1] = max(merged.back()[1], intervals[i][1]);

        } else {

            merged.push\_back(intervals[i]);

        }

    }

    return merged;

}

int main() {

    vector<vector<int>> arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

    vector<vector<int>> result1 = mergeIntervals(arr1);

    for (const auto& interval : result1) {

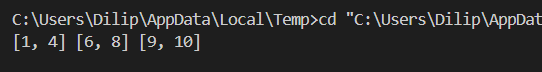
        cout << "[" << interval[0] << ", " << interval[1] << "] ";

    }

    cout << endl;

    return 0;

}



TC-O(n logn)

**9.BOOLEAN MATRIX**

#include <iostream>

#include <vector>

using namespace std;

int main() {

    vector<vector<int>> matrix = {

        {0, 0, 0},

        {1, 0, 1},

        {0, 0, 0}

    };

    int col0 = 1;

    int m = matrix.size();

    int n = matrix[0].size();

    for (int i = 0; i < m; i++) {

        for (int j = 0; j < n; j++) {

            if (matrix[i][j] == 1) {

                matrix[i][0] = 1;

                if (j != 0) {

                    matrix[0][j] = 1;

                } else {

                    col0 = 0;

                }

            }

        }

    }

    for (int i = 1; i < m; i++) {

        for (int j = 1; j < n; j++) {

            if (matrix[i][j] != 1) {

                if (matrix[i][0] == 1 || matrix[0][j] == 1) {

                    matrix[i][j] = 1;

                }

            }

        }

    }

    if (matrix[0][0] == 1) {

        for (int j = 0; j < n; j++) {

            matrix[0][j] = 1;

        }

    }

    if (col0 == 0) {

        for (int i = 0; i < m; i++) {

            matrix[i][0] = 1;

        }

    }

    cout << "Modified Matrix:" << endl;

    for (const auto& row : matrix) {

        for (int val : row) {

            cout << val << " ";

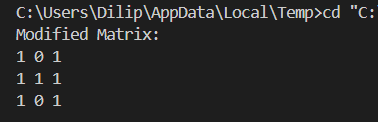
        }

        cout << endl;

    }

    return 0;

}



TC-O(M\*N)

**10.SPIRAL MATRIX**

#include <iostream>

#include <vector>

using namespace std;

void printSpiralMatrix(vector<vector<int>>& matrix) {

    int m = matrix.size();

    int n = matrix[0].size();

    int top = 0, bottom = m - 1, left = 0, right = n - 1;

    while (top <= bottom && left <= right) {

        for (int i = left; i <= right; i++) {

            cout << matrix[top][i] << " ";

        }

        top++;

        for (int i = top; i <= bottom; i++) {

            cout << matrix[i][right] << " ";

        }

        right--;

        if (top <= bottom) {

            for (int i = right; i >= left; i--) {

                cout << matrix[bottom][i] << " ";

            }

            bottom--;

        }

        if (left <= right) {

            for (int i = bottom; i >= top; i--) {

                cout << matrix[i][left] << " ";

            }

            left++;

        }

    }

}

int main() {

    vector<vector<int>> matrix = {

        {1, 2, 3, 4},

        {5, 6, 7, 8},

        {9, 10, 11, 12},

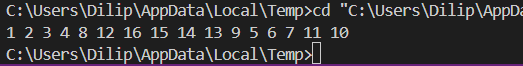
        {13, 14, 15, 16}

    };

    printSpiralMatrix(matrix);

    return 0;

}



TC-O(M\*N)

**11.VALID PARANTHESES**

#include <iostream>

#include <stack>

using namespace std;

bool isBalanced(string str) {

    stack<char> st;

    for (char ch : str) {

        if (ch == '(') {

            st.push(ch);

        } else {

            if (st.empty()) return false;

            st.pop();

        }

    }

    return st.empty();

}

int main() {

    string str1 = "((()))()()";

    if (isBalanced(str1)) {

        cout << "Balanced" << endl;

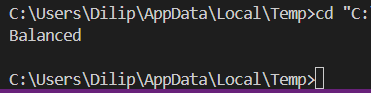
    } else {

        cout << "Not Balanced" << endl;

    }

    return 0;

}



TC-O(N)

**12.VALID ANAGRAM**

#include <iostream>

#include <vector>

using namespace std;

bool isAnagram(string s, string t) {

    vector<int> count(26, 0);

    if(s.length() != t.length()) {

        return false;

    }

    for(char ch : s) {

        count[ch - 'a']++;

    }

    for(char ch : t) {

        count[ch - 'a']--;

    }

    for(int i = 0; i < 26; i++) {

        if(count[i] != 0) {

            return false;

        }

    }

    return true;

}

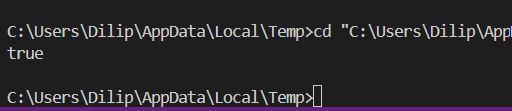
int main() {

    string s1 = "geeks", s2 = "kseeg";

    cout << (isAnagram(s1, s2) ? "true" : "false") << endl;

    return 0;

}



TC-O(N)

**13.LONGEST PALINDROMIC SUBSTRING**

#include <iostream>

#include <string>

using namespace std;

string longestPalindrome(string s) {

    int n = s.size();

    if (n == 0) return "";

    int start = 0, maxLength = 1;

    auto expandAroundCenter = [&](int left, int right) {

        while (left >= 0 && right < n && s[left] == s[right]) {

            left--;

            right++;

        }

        return right - left - 1;

    };

    for (int i = 0; i < n; ++i) {

        int len1 = expandAroundCenter(i, i);

        int len2 = expandAroundCenter(i, i + 1);

        int len = max(len1, len2);

        if (len > maxLength) {

            maxLength = len;

            start = i - (maxLength - 1) / 2;

        }

    }

    return s.substr(start, maxLength);

}

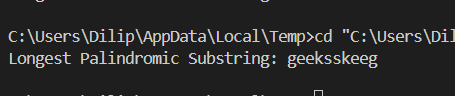
int main() {

    string str = "forgeeksskeegfor";

    cout << "Longest Palindromic Substring: " << longestPalindrome(str) << endl;

     return 0;

}



TC-O(N^2)

**14.LONGEST COMMON PREFIX USING SORTING**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

string longestCommonPrefix(vector<string>& arr) {

    int n=arr.size();

    if (arr.empty()) return "-1";

    sort(arr.begin(), arr.end());

    string first = arr[0];

    string last = arr[n- 1];

    int i = 0;

    while (i < first.size() && i < last.size() && first[i] == last[i]) {

        i++;

    }

    string prefix = first.substr(0, i);

    return (prefix.empty() ? "-1" : prefix);

}

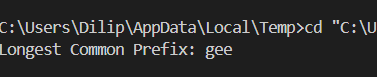
int main() {

    vector<string> arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

    cout << "Longest Common Prefix: " << longestCommonPrefix(arr1) << endl;

    return 0;

}



TC-O(n logN)

**15.FIND MIDDLE ELEMENT OF STACK**

#include <iostream>

#include <stack>

using namespace std;

void deleteMiddle(stack<int>& st, int count, int size) {

    if (st.empty() || count == size / 2) {

        st.pop();

        return;

    }

    int top = st.top();

    st.pop();

    deleteMiddle(st, count + 1, size);

    st.push(top);

}

void deleteMiddleElement(stack<int>& st) {

    int size = st.size();

    if (size == 0) return;

    deleteMiddle(st, 0, size);

}

void printStack(stack<int> st) {

    while (!st.empty()) {

        cout << st.top() << " ";

        st.pop();

    }

    cout << endl;

}

int main() {

    stack<int> st;

    st.push(1);

    st.push(2);

    st.push(3);

    st.push(4);

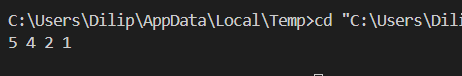
    st.push(5);

    deleteMiddleElement(st);

    printStack(st);

    return 0;

}



TC-O(N)

**16.NEXT GREATER ELEMENT**

#include <iostream>

#include <unordered\_map>

#include <stack>

#include <vector>

using namespace std;

vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

    unordered\_map<int, int> nge;

    stack<int> st;

    for (int num : nums2) {

        while (!st.empty() && st.top() < num) {

            nge[st.top()] = num;

            st.pop();

        }

        st.push(num);

    }

    while (!st.empty()) {

        nge[st.top()] = -1;

        st.pop();

    }

    vector<int> result;

    for (int num : nums1) {

        result.push\_back(nge[num]);

    }

    return result;

}

int main() {

    vector<int> nums1 = {4, 5, 2, 25};

    vector<int> nums2 = {13, 7, 6, 12, 4, 5, 2, 25};

    vector<int> result = nextGreaterElement(nums1, nums2);

    for (int x : result) {

        cout << x << " ";

    }

    return 0;

}



TC-O(N+M)

**17.RIGHT VIEW OF BINARY TREE**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

    TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

};

vector<int> rightSideView(TreeNode\* root) {

    vector<int> ans;

    if (!root) return ans;

    queue<TreeNode\*> q;

    q.push(root);

    while (!q.empty()) {

        int size = q.size();

        for (int i = 0; i < size; i++) {

            TreeNode\* node = q.front();

            q.pop();

            if (i == size - 1) {

                ans.push\_back(node->val);

            }

            if (node->left) q.push(node->left);

            if (node->right) q.push(node->right);

        }

    }

    return ans;

}

int main() {

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

    root->left->right = new TreeNode(4);

    root->right->right = new TreeNode(5);

    vector<int> result = rightSideView(root);

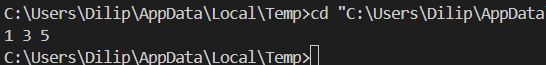
    for (int val : result) {

        cout << val << " ";

    }

    return 0;

}



TC-O(N+M)

**18.MAXIMUM DEPTH/HEIGHT OF BINARY TREE**

#include <iostream>

#include <algorithm>

using namespace std;

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode() : val(0), left(nullptr), right(nullptr) {}

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

    TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

};

int maxDepth(TreeNode\* root) {

    if (root == nullptr) return 0;

    int lh = maxDepth(root->left);

    int rh = maxDepth(root->right);

    return 1 + max(lh, rh);

}

int main() {

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

    root->left->left = new TreeNode(4);

    root->left->right = new TreeNode(5);

    cout << "Maximum Depth of Binary Tree: " << maxDepth(root) << endl;

    return 0;

}



TC-O(N)