1. **Kadane's Algo -> Maximum SubArray:**

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> nums = {2, 3, -8, 7, -1, 2, 3};
    int maxSum = nums[0];
    int curSum = 0;
    for(auto n : nums){
        curSum = max(curSum,0);
        curSum += n;
        maxSum = max(maxSum,curSum);
    }
    cout<<maxSum;
    return 0;
}
```

Time Complexity: O(n)

Output:

```
PS C:\Users\aakas\Desktop\New folder> .\SumOfArray.exe
11
```

2. **Maximum Product Subarray:**

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> arr = {-2, 6, -3, -10, 0, 2};
    int j = arr.size()-1,i = 0;
    int l2r = 1, r2l = 1;
    int maxProd = 0, curProd = 0;
    while(i<arr.size() && j>=0){
        if(l2r == 0){
            l2r = arr[i];
            i++;
            continue;
        }
        if( r2l == 0){
            r2l = arr[j];
            j--;
            continue;
        }
        l2r *= arr[i];
        r2l *= arr[j];
        curProd = max(l2r,r2l);
        maxProd = max(curProd,maxProd);
        i++;
        j--;
    }
    cout<<maxProd;
    return 0;
}
```

Time Complexity: O(n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> .\ProductOfArray.exe
180
```

3. **Search in a sorted and rotated Array:**

```
#include <iostream>
#include <vector>
using namespace std;
int search(vector<int>& nums, int target) {
    int s = 0, e = nums.size() - 1;
    while (s <= e) {
        int mid = s + (e - s) / 2;
        if (nums[mid] == target) return mid;
        if (nums[s] <= nums[mid]) {
            if (target >= nums[s] && target <= nums[mid]) e = mid - 1;
            else s = mid + 1;
        } else {
            if (target >= nums[mid] && target <= nums[e]) s = mid + 1;
            else e = mid - 1;
        }
    }
    return -1;
}
int main() {
    vector<int> nums1 = {4, 5, 6, 7, 0, 1, 2};
    int target1 = 0;
    cout << search(nums1, target1) << endl;
    vector<int> nums2 = {4, 5, 6, 7, 0, 1, 2};
    int target2 = 3;
    cout << search(nums2, target2) << endl;
    return 0;
}
```

Time Complexity: O(log n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> .\SearchInRotatedArray
4
-1
```

4. **Container with Most Water:**

```
#include <iostream>
#include <vector>
#include <algorithm> // For min and max functions
using namespace std;
class Solution {
public:
    int maxArea(vector<int>& height) {
        int n = height.size();
```

```
        int start = 0;
        int end = n - 1;
        int area = 0;
        int result = 0;
        while (start < end) {
            int breadth = end - start;
            area = breadth * min(height[start], height[end]);
            result = max(area, result);

            if (height[start] <= height[end]) {
                start++;
            } else {
                end--;
            }
        }
        return result;
    }
};
int main() {
    Solution solution;
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    int maxWater = solution.maxArea(height);
    cout << "Maximum area: " << maxWater << endl;
    return 0;
}
```

Time Complexity: O(n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ -std=c++11 -o MaxArea MaxArea.cpp
PS C:\Users\aakas\Desktop\New folder> ./MaxArea
Maximum area: 49
```

5. **<u>Factorial:</u>**

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> factorialDP(int n) {
    vector<int> result(1, 1);
    for (int i = 1; i <= n; ++i) {
        int carry = 0;
        for (int j = 0; j < result.size(); ++j) {
            int prod = result[j] * i + carry;
            result[j] = prod % 10;
            carry = prod / 10;
        }
        while (carry) {
            result.push_back(carry % 10);
            carry /= 10;
        }
    }
    return result;
```

```
}
int main() {
    vector<int> fact;
    fact = factorialDP(100);
    for (auto it = fact.rbegin(); it != fact.rend(); ++it) {
        cout << *it;
    }
    cout << endl;
    fact = factorialDP(50);
    for (auto it = fact.rbegin(); it != fact.rend(); ++it) {
        cout << *it;
    }
    cout << endl;
    return 0;
}
```

Time Complexity: O(log n!)

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ -o factorial factorial.cpp
PS C:\Users\aakas\Desktop\New folder> ./factorial
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
Factorial of 50 is: 30414093201713378043612608166064768844377641568960512000000000000
```

6. **<u>Trapping Rain Water:</u>**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int trap(vector<int>& height) {
    int n = height.size();
    if (n == 0) return 0;
    int left = 0, right = n - 1;
    int left_max = 0, right_max = 0;
    int water_trapped = 0;
    while (left <= right) {
        if (height[left] <= height[right]) {
            if (height[left] >= left_max) {
                left_max = height[left];
            } else {
                water_trapped += left_max - height[left];
            }
            left++;
        } else {
            if (height[right] >= right_max) {
                right_max = height[right];
            } else {
                water_trapped += right_max - height[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};
    cout << trap(arr1) << endl;
    cout << trap(arr2) << endl;
    cout << trap(arr3) << endl;

    return 0;
}
```

Time Complexity: O(n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ -o trapping_rainwater trapping_rainwater.cpp
>>
PS C:\Users\aakas\Desktop\New folder> ./trapping_rainwater
>>
10
7
0
```

7. **<u>Chocolate Distribution Problem:</u>**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int chocolateDistribution(vector<int>& arr, int m) {
    int n = arr.size();
    if (n < m) return -1;
    sort(arr.begin(), arr.end());
    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() {
    vector<int> arr = {7, 3, 2, 4, 9, 12, 56};
    int m = 3;
    cout << chocolateDistribution(arr, m) << endl;
    return 0;
}
```

Time Complexity: O(n log n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ chocolate_distribution.cpp -o chocolate_distribution
>>
PS C:\Users\aakas\Desktop\New folder> ./chocolate_distribution
>>
2
```

8. **Merge Overlapping Intervals:**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<vector<int>> mergeIntervals(vector<vector<int>>& iv) {
    sort(iv.begin(), iv.end());
    vector<vector<int>> res;
    for (auto& i : iv) {
        if (res.empty() || res.back()[1] < i[0])
            res.push_back(i);
        else
            res.back()[1] = max(res.back()[1], i[1]);
    }
    return res;
}
int main() {
    vector<vector<int>> arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
    vector<vector<int>> arr2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
    for (auto& i : mergeIntervals(arr1))
        cout << "[" << i[0] << ", " << i[1] << "] ";
    cout << endl;
    for (auto& i : mergeIntervals(arr2))
        cout << "[" << i[0] << ", " << i[1] << "] ";
    cout << endl;
    return 0;
}
```

Time Complexity: O(n log n)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ merge_intervals.cpp -o merge_intervals
PS C:\Users\aakas\Desktop\New folder> ./merge_intervals
[1, 4] [6, 8] [9, 10]
[1, 6] [7, 8]
```

9. **Boolean Matrix:**

```
#include <iostream>
#include <vector>
using namespace std;
void modifyMatrix(vector<vector<int>>& mat) {
    int m = mat.size(), n = mat[0].size();
    vector<int> row(m, 0), col(n, 0);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
```

```
            if (mat[i][j] == 1) {
                row[i] = 1;
                col[j] = 1;
            }
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (row[i] == 1 || col[j] == 1)
                mat[i][j] = 1;
}
int main() {
    vector<vector<int>> mat1 = {{1, 0}, {0, 0}};
    vector<vector<int>> mat2 = {{0, 0, 0}, {0, 0, 1}};
    vector<vector<int>> mat3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};
    modifyMatrix(mat1);
    modifyMatrix(mat2);
    modifyMatrix(mat3);
    for (auto& row : mat1) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    cout << endl;
    for (auto& row : mat2) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    cout << endl;
    for (auto& row : mat3) {
        for (int cell : row) cout << cell << " ";
        cout << endl;
    }
    return 0;
}
```

Time Complexity: O(N*M)

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ boolean_matrix.cpp -o boolean_matrix
PS C:\Users\aakas\Desktop\New folder> ./boolean_matrix
1 1
1 0

0 0 1
1 1 1

1 1 1 1
1 1 1 1
1 0 1 1
```

10. **<u>Spiral Matrix:</u>**

```
#include <iostream>
#include <vector>
using namespace std;
void printSpiral(const vector<vector<int>>& matrix) {
```

```
    int m = matrix.size();
    if (m == 0) return;
    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++;
        }
    }
    cout << endl;
}
int main() {
    vector<vector<int>> matrix1 = {{1, 2, 3, 4},
                                   {5, 6, 7, 8},
                                   {9, 10, 11, 12},
                                   {13, 14, 15, 16}};
    vector<vector<int>> matrix2 = {{1, 2, 3, 4, 5, 6},
                                   {7, 8, 9, 10, 11, 12},
                                   {13, 14, 15, 16, 17, 18}};
    printSpiral(matrix1);
    printSpiral(matrix2);
    return 0;
}
```

Time Complexity: O(N*M)

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ spiral_matrix.cpp -o spiral_matrix
PS C:\Users\aakas\Desktop\New folder> ./spiral_matrix
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

11. **<u>Valid Parentheses:</u>**

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
bool isBalanced(const string& str) {
    stack<char> s;
    for (char c : str) {
        if (c == '(') {
            s.push(c);
        } else if (c == ')') {
            if (s.empty()) return false;
```

```
            s.pop();
        }
    }
    return s.empty();
}
int main() {
    string str1 = "((()))()()";
    string str2 = "())((())";
    cout << (isBalanced(str1) ? "Balanced" : "Not Balanced") << endl;
    cout << (isBalanced(str2) ? "Balanced" : "Not Balanced") << endl;
    return 0;
}
```

Time Complexity: O(N)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ balanced_parentheses.cpp -o balanced_parentheses
PS C:\Users\aakas\Desktop\New folder> ./balanced_parentheses
Balanced
Not Balanced
```

12. **<u>Anagram Check:</u>**

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;
bool areAnagrams(string s1, string s2) {
    if (s1.length() != s2.length()) return false;
    sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());
    return s1 == s2;
}
int main() {
    string s1 = "geeks", s2 = "kseeg";
    cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;
    s1 = "allergy";
    s2 = "allergic";
    cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;
    s1 = "g";
    s2 = "g";
    cout << (areAnagrams(s1, s2) ? "true" : "false") << endl;
    return 0;
}
```

Time Complexity: O(N)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ anagram_check.cpp -o anagram_check
PS C:\Users\aakas\Desktop\New folder> ./anagram_check
true
false
true
```

13. **Longest Palindromic String:**

```
#include <iostream>
#include <string>
using namespace std;
string longestPalindrome(string str) {
    int n = str.size();
    if (n < 2) return str;
    int start = 0, maxLen = 1;
    for (int i = 0; i < n; ++i) {
        int l = i, r = i;
        while (r < n - 1 && str[r] == str[r + 1]) ++r;
        i = r;
        while (l > 0 && r < n - 1 && str[l - 1] == str[r + 1]) {
            --l;
            ++r;
        }
        if (r - l + 1 > maxLen) {
            start = l;
            maxLen = r - l + 1;
        }
    }
    return str.substr(start, maxLen);
}

int main() {
    cout << longestPalindrome("forgeeksskeegfor") << endl;
    cout << longestPalindrome("Geeks") << endl;
    cout << longestPalindrome("abc") << endl;
    cout << longestPalindrome("") << endl;
    return 0;
}
```

Time Complexity: O(N^2)
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ longest_palindromic_substring.cpp -o longest_palindromic_substring
PS C:\Users\aakas\Desktop\New folder> ./longest_palindromic_substring
geeksskeeg
ee
a
```

14. **Longest Common Prefix Sum:**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
string longestCommonPrefix(vector<string>& arr) {
    int n = arr.size();
    if (n == 0) return "-1";
    sort(arr.begin(), arr.end());
    string first = arr[0], last = arr[n - 1];
    int len = min(first.size(), last.size());
    int i = 0;
```

```
    while (i < len && first[i] == last[i]) i++;
    return i ? first.substr(0, i) : "-1";
}
int main() {
    vector<string> arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
    vector<string> arr2 = {"hello", "world"};
    cout << longestCommonPrefix(arr1) << endl;
    cout << longestCommonPrefix(arr2) << endl;
    return 0;
}
```

Time Complexity: O(N log N + M**)**
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ longest_common_prefix.cpp -o longest_common_prefix
PS C:\Users\aakas\Desktop\New folder> ./longest_common_prefix
gee
-1
```

15. **<u>Delete middle element:</u>**

```
#include <iostream>
#include <stack>
using namespace std;
void deleteMiddle(stack<int>& s, int k) {
    if (k == 1) {
        s.pop();
        return;
    }
    int temp = s.top();
    s.pop();
    deleteMiddle(s, k - 1);
    s.push(temp);
}
void deleteMiddleElement(stack<int>& s) {
    int middle = s.size() / 2 + 1;
    deleteMiddle(s, middle);
}
int main() {
    stack<int> s;
    s.push(1);
    s.push(2);
    s.push(3);
    s.push(4);
    s.push(5);
    deleteMiddleElement(s);
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    return 0;
}
```

Time Complexity: O(N**)**

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ delete_middle_stack.cpp -o delete_middle_stack
PS C:\Users\aakas\Desktop\New folder> ./delete_middle_stack
5 4 2 1
```

16. **<u>Next Greater Element:</u>**

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
vector<int> nextGreaterElement(vector<int>& arr) {
    int n = arr.size();
    vector<int> nge(n, -1);
    stack<int> s;
    for (int i = 0; i < n; i++) {
        while (!s.empty() && arr[s.top()] < arr[i]) {
            nge[s.top()] = arr[i];
            s.pop();
        }
        s.push(i);
    }
    return nge;
}
int main() {
    vector<int> arr = {4, 5, 2, 25};
    vector<int> result = nextGreaterElement(arr);
    for (int i = 0; i < arr.size(); i++) {
        cout << arr[i] << " -> " << result[i] << endl;
    }
    return 0;
}
```

Time Complexity: O(N**)**
Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ next_greater_element.cpp -o next_greater_element
PS C:\Users\aakas\Desktop\New folder> ./next_greater_element
4 -> 5
5 -> 25
2 -> 25
25 -> -1
```

17. **<u>Right View of Binary Tree:</u>**

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
```

```
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
void rightView(Node* root) {
    if (root == nullptr) return;
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        int n = q.size();
        for (int i = 1; i <= n; i++) {
            Node* node = q.front();
            q.pop();
            if (i == n) {
                cout << node->data << " ";
            }
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);
    root->left->right->left = new Node(7);
    root->left->right->right = new Node(8);
    rightView(root);
    return 0;
}
```

Time Complexity: O(N**)**

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ right_view.cpp -o right_view
PS C:\Users\aakas\Desktop\New folder> ./right_view
1 3 6 8
```

18. **<u>Maximum Depth:</u>**

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
int maxDepth(Node* root) {
    if (root == nullptr) {
        return 0;
```

```
    }
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return max(leftDepth, rightDepth) + 1;
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);
    cout << "Maximum Depth of the Binary Tree: " << maxDepth(root) << endl;
    return 0;
}
```

Time Complexity: O(N**)**

Output:

```
PS C:\Users\aakas\Desktop\New folder> g++ max_depth.cpp -o max_depth
PS C:\Users\aakas\Desktop\New folder> ./max_depth
Maximum Depth of the Binary Tree: 3
```