```
//binary serch
//--------------

/*
#include <iostream>
#include <vector>

using namespace std;

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

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (array[mid] == target) {
            return mid; // Target found
        } else if (array[mid] < target) {
            left = mid + 1; // Discard left half
        } else {
            right = mid - 1; // Discard right half
        }
    }

    return -1; // Target not found
}

int main() {
    vector<int> sortedArray = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int target = 7;

    int result = binarySearch(sortedArray, target);

    if (result != -1) {
        cout << "Element found at index " << result << endl;
    } else {
        cout << "Element not found in the array" << endl;
    }

    return 0;
}
*/
//preafix suffix
//--------------
/*
#include <iostream>
#include <vector>

using namespace std;
```

```
// Function to calculate the prefix sum array
vector<int> calculatePrefixArray(const vector<int>& array) {
    int n = array.size();
    vector<int> prefixArray(n, 0);

    prefixArray[0] = array[0];
    for (int i = 1; i < n; ++i) {
        prefixArray[i] = prefixArray[i - 1] + array[i];
    }

    return prefixArray;
}

// Function to calculate the suffix sum array
vector<int> calculateSuffixArray(const vector<int>& array) {
    int n = array.size();
    vector<int> suffixArray(n, 0);

    suffixArray[n - 1] = array[n - 1];
    for (int i = n - 2; i >= 0; --i) {
        suffixArray[i] = suffixArray[i + 1] + array[i];
    }

    return suffixArray;
}

int main() {
    vector<int> array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Calculate and print the prefix array
    vector<int> prefixArray = calculatePrefixArray(array);
    cout << "Prefix Array: ";
    for (int num : prefixArray) {
        cout << num << " ";
    }
    cout << endl;

    // Calculate and print the suffix array
    vector<int> suffixArray = calculateSuffixArray(array);
    cout << "Suffix Array: ";
    for (int num : suffixArray) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}

*/
```

```
//Partial Sum
//-----------
/*
#include <iostream>
#include <vector>

using namespace std;

// Function to calculate the partial sum array
vector<int> calculatePartialSum(const vector<int>& array) {
    int n = array.size();
    vector<int> partialSum(n, 0);

    partialSum[0] = array[0];
    for (int i = 1; i < n; ++i) {
        partialSum[i] = partialSum[i - 1] + array[i];
    }

    return partialSum;
}

// Function to calculate the sum of elements in a range [start, end]
int calculateRangeSum(const vector<int>& partialSum, int start, int end) {
    if (start == 0) {
        return partialSum[end];
    } else {
        return partialSum[end] - partialSum[start - 1];
    }
}

int main() {
    vector<int> array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Calculate and print the partial sum array
    vector<int> partialSum = calculatePartialSum(array);
    cout << "Partial Sum Array: ";
    for (int num : partialSum) {
        cout << num << " ";
    }
    cout << endl;

    // Example: Calculate the sum of elements in the range [2, 7]
    int start = 2;
    int end = 7;
    int rangeSum = calculateRangeSum(partialSum, start, end);

    cout << "Sum of elements in the range [" << start << ", " << end << "]: " << rangeSum <<
endl;
```

```
    return 0;
}

*/
//Greedy Algorithm
//----------------
/*
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Structure to represent an activity
struct Activity {
    int start, finish;
};

// Function to compare activities based on their finish times
bool compareActivities(const Activity& a, const Activity& b) {
    return (a.finish < b.finish);
}

// Function to perform activity selection using the greedy algorithm
void selectActivities(const vector<Activity>& activities) {
    int n = activities.size();

    // Sort activities based on finish times
    vector<Activity> sortedActivities = activities;
    sort(sortedActivities.begin(), sortedActivities.end(), compareActivities);

    // The first activity always gets selected
    cout << "Selected Activities: (" << sortedActivities[0].start << ", " <<
sortedActivities[0].finish << ") ";

    // Consider the rest of the activities
    int j = 0;
    for (int i = 1; i < n; i++) {
        // If this activity has a start time greater than or equal to the finish time of the
previous selected activity, then select it
        if (sortedActivities[i].start >= sortedActivities[j].finish) {
            cout << "(" << sortedActivities[i].start << ", " << sortedActivities[i].finish <<
") ";
            j = i;
        }
    }
}

int main() {
    vector<Activity> activities = {
        {1, 4}, {3, 5}, {0, 6}, {5, 7}, {8, 9}, {5, 9}
    };

    cout << "Original Activities: ";
    for (const Activity& act : activities) {
        cout << "(" << act.start << ", " << act.finish << ") ";
    }
    cout << endl;

    selectActivities(activities);

    return 0;
}

*/
//Two Pointers
//-------------
/*
#include <iostream>
#include <vector>

using namespace std;

// Function to find a pair with the given sum in a sorted array
bool findPairWithSum(const vector<int>& sortedArray, int targetSum) {
    int left = 0;
    int right = sortedArray.size() - 1;

    while (left < right) {
        int currentSum = sortedArray[left] + sortedArray[right];

        if (currentSum == targetSum) {
            cout << "Pair with sum " << targetSum << " found: (" << sortedArray[left] << ", "
<< sortedArray[right] << ")" << endl;
            return true;
        } else if (currentSum < targetSum) {
            left++; // Move left pointer to increase the sum
        } else {
            right--; // Move right pointer to decrease the sum
        }
    }

    cout << "No pair found with sum " << targetSum << endl;
    return false;
}

int main() {
    vector<int> sortedArray = {-2, 1, 2, 4, 7, 11, 15};
    int targetSum = 10;
    cout << "Sorted Array: ";
    for (int num : sortedArray) {
        cout << num << " ";
    }
    cout << endl;

    findPairWithSum(sortedArray, targetSum);

    return 0;
}

*/
//Number Theory
//-------------
/*
#include <iostream>

using namespace std;

// Function to calculate the GCD of two numbers using Euclidean Algorithm
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    int num1 = 48;
    int num2 = 18;

    cout << "Numbers: " << num1 << " and " << num2 << endl;

    // Calculate and print the GCD
    int result = gcd(num1, num2);
    cout << "GCD: " << result << endl;

    return 0;
}
//what is gdc : Greatest Common Divisor
*/
```