1

kadane max sum subarray

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int maxSubarraySum(vector<int>& arr) {
    int maxSoFar = arr[0], currentMax = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        currentMax = max(arr[i], currentMax + arr[i]);
        maxSoFar = max(maxSoFar, currentMax);
    }
    return maxSoFar;
}
int main() {
    vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    cout << maxSubarraySum(arr) << endl;  // Output: 6
    return 0;
}
```

2

2. Two-pointer Technique

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
pair<int, int> twoSum(vector<int>& arr, int target) {
    sort(arr.begin(), arr.end());
    int left = 0, right = arr.size() - 1;
    while (left < right) {
        int sum = arr[left] + arr[right];
        if (sum == target) return {arr[left], arr[right]};
        else if (sum < target) left++;
        else right--;
    }
    return {-1, -1};  // Pair not found
}
int main() {
    vector<int> arr = {10, 20, 35, 50, 75, 80};
    int target = 70;
    auto result = twoSum(arr, target);
    if (result.first != -1)
        cout << result.first << ", " << result.second << endl;
 // Output: 20, 50
    else
        cout << "No pair found" << endl;
    return 0;
}
```

3

Dutch nation flag 1 0 2

```
#include <iostream>
#include <vector>
using namespace std;
void sortColors(vector<int>& arr) {
    int low = 0, mid = 0, high = arr.size() - 1;
    while (mid <= high) {
        if (arr[mid] == 0) swap(arr[low++], arr[mid++]);
        else if (arr[mid] == 1) mid++;
        else swap(arr[mid], arr[high--]);
    }
}
int main() {
    vector<int> arr = {2, 0, 2, 1, 1, 0};
    sortColors(arr);
    for (int num : arr) cout << num << " ";  // Output: 0 0 1
1 2 2
    return 0;
}
```

4

Merge overlapping intervals

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<pair<int, int>> mergeIntervals(vector<pair<int,
int>>& intervals) {
    vector<pair<int, int>> merged;
    if (intervals.empty()) return merged;
    sort(intervals.begin(), intervals.end());  // Sort by
starting time
    merged.push_back(intervals[0]);
    for (int i = 1; i < intervals.size(); i++) {
        if (merged.back().second >= intervals[i].first) {
            merged.back().second =
max(merged.back().second, intervals[i].second);
        } else {
            merged.push_back(intervals[i]);
        }
    }
    return merged;
}
int main() {
    vector<pair<int, int>> intervals = {{1, 3}, {2, 6}, {8, 10},
{15, 18}};
    auto merged = mergeIntervals(intervals);
    for (auto interval : merged)
        cout << "[" << interval.first << ", " <<
interval.second << "] ";
    // Output: [1, 6] [8, 10] [15, 18]
    return 0;
}
```

5

PREFIX sum

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> prefixSumArray(vector<int>& arr) {
    vector<int> prefixSum(arr.size());
    prefixSum[0] = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        prefixSum[i] = prefixSum[i - 1] + arr[i];
    }
    return prefixSum;
}
int rangeSum(vector<int>& prefixSum, int left, int right) {
    return left == 0 ? prefixSum[right] : prefixSum[right] -
prefixSum[left - 1];
}
int main() {
    vector<int> arr = {1, 2, 3, 4, 5};
    auto prefixSum = prefixSumArray(arr);
    cout << rangeSum(prefixSum, 1, 3) << endl;  //
Output: 9 (2 + 3 + 4)
    return 0;
}
```

6

Majortiy element (moore Vooting)

```
#include <iostream>
#include <vector>
using namespace std;
int findMajorityElement(vector<int>& arr) {
    int count = 0, candidate = -1;
    for (int num : arr) {
        if (count == 0) {
            candidate = num;
            count = 1;
        } else {
            count += (num == candidate) ? 1 : -1;
        }
    }
    // Verify the candidate
    count = 0;
    for (int num : arr) {
        if (num == candidate) count++;
    }
    return (count > arr.size() / 2) ? candidate : -1;
}
int main() {
    vector<int> arr = {3, 3, 4, 2, 4, 4, 2, 4, 4};
    cout << findMajorityElement(arr) << endl;  // Output: 4
    return 0;
}
```

7

find missing number

```
#include <iostream>
#include <vector>
using namespace std;
int findMissingNumber(vector<int>& arr) {
    int n = arr.size() + 1;  // Size should be n+1 including
the missing number
    int xorFull = 0, xorArr = 0;
    for (int i = 1; i <= n; i++) xorFull ^= i;
    for (int num : arr) xorArr ^= num;
    return xorFull ^ xorArr;
}
int main() {
    vector<int> arr = {1, 2, 4, 5, 6};
    cout << findMissingNumber(arr) << endl;  // Output: 3
    return 0;
}
```

8

find duplicates slow and fast pointer

```
#include <iostream>
#include <vector>
using namespace std;
int findDuplicate(vector<int>& arr) {
    int slow = arr[0], fast = arr[0];
    // Phase 1: Detect cycle
    do {
        slow = arr[slow];
        fast = arr[arr[fast]];
    } while (slow != fast);
    // Phase 2: Find entrance to cycle
    slow = arr[0];
    while (slow != fast) {
        slow = arr[slow];
        fast = arr[fast];
    }
    return slow;
}
int main() {
    vector<int> arr = {3, 1, 3, 4, 2};
    cout << findDuplicate(arr) << endl;  // Output: 3
    return 0;
}
```

9

trapping rain problem

```
#include <iostream>
#include <vector>
using namespace std;
int trapRainwater(vector<int>& height) {
    int n = height.size();
    if (n <= 2) return 0;
    vector<int> leftMax(n), rightMax(n);
    leftMax[0] = height[0];
    rightMax[n - 1] = height[n - 1];
    for (int i = 1; i < n; i++)
        leftMax[i] = max(leftMax[i - 1], height[i]);
    for (int i = n - 2; i >= 0; i--)
        rightMax[i] = max(rightMax[i + 1], height[i]);
    int water = 0;
    for (int i = 0; i < n; i++)
        water += min(leftMax[i], rightMax[i]) - height[i];
    return water;
}
int main() {
    vector<int> height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    cout << trapRainwater(height) << endl;  // Output: 6
    return 0;
}
```

10

Rearrange alternate positive and negative elements

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
void rearrangeArray(vector<int>& arr) {
    int n = arr.size();
    vector<int> result(n);
    int positiveIndex = 0, negativeIndex = 1;
    for (int i = 0; i < n; i++) {
        if (arr[i] >= 0) {
            result[positiveIndex] = arr[i];
            positiveIndex += 2;
        } else {
            result[negativeIndex] = arr[i];
            negativeIndex += 2;
        }
    }
    // Copy the rearranged array back to the original array
    for (int i = 0; i < n; i++) {
        arr[i] = result[i];
    }
}
int main() {
    vector<int> arr = {1, -2, 3, -4, 5, -6};
    rearrangeArray(arr);
    for (int num : arr) cout << num << " ";  // Output: 1 -2 3
-4 5 -6
    return 0;
}
```

11

Longest Common Subsequence

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;
int longestConsecutive(vector<int>& nums) {
    unordered_set<int> numSet(nums.begin(),
nums.end());
    int longest = 0;
    for (int num : nums) {
        if (numSet.find(num - 1) == numSet.end()) {  // Only
check if it's the start of a sequence
            int currentNum = num;
            int currentStreak = 1;
            while (numSet.find(currentNum + 1) !=
numSet.end()) {
                currentNum++;
                currentStreak++;
            }
            longest = max(longest, currentStreak);
        }
    }
    return longest;
}
int main() {
    vector<int> nums = {100, 4, 200, 1, 3, 2};
    cout << longestConsecutive(nums) << endl;  //
Output: 4
    return 0;
}
```

12

subarray with given sum

```
#include <iostream>
#include <vector>
using namespace std;
bool subarraySum(vector<int>& arr, int target) {
    int sum = 0;
    unordered_map<int, int> prefixSum;
    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];
        if (sum == target) return true;
        if (prefixSum.find(sum - target) != prefixSum.end())
return true;
        prefixSum[sum] = i;
    }
    return false;
}
int main() {
    vector<int> arr = {1, 4, 20, 3, 10, 5};
    int target = 33;
    cout << subarraySum(arr, target) << endl;  // Output:
1 (True)
    return 0;
}
```

13

intersection of 2 array

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;
vector<int> intersectionOfArrays(vector<int>& arr1,
vector<int>& arr2) {
    unordered_set<int> set1(arr1.begin(), arr1.end());
    unordered_set<int> result;
    for (int num : arr2) {
        if (set1.find(num) != set1.end()) {
            result.insert(num);
        }
    }
    return vector<int>(result.begin(), result.end());
}
int main() {
    vector<int> arr1 = {1, 2, 2, 1};
    vector<int> arr2 = {2, 2};
    vector<int> result = intersectionOfArrays(arr1, arr2);
    for (int num : result) cout << num << " ";  // Output: 2
    return 0;
}
```

14

find pairs of sum

```
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;
vector<pair<int, int>> findPairsWithSum(vector<int>&
arr, int target) {
    unordered_map<int, int> map;
    vector<pair<int, int>> result;
    for (int num : arr) {
        int complement = target - num;
        if (map[complement] > 0) {
            result.push_back({complement, num});
            map[complement]--;
        } else {
            map[num]++;
        }
    }
    return result;
}
int main() {
    vector<int> arr = {1, 2, 3, 4, 3, 2, 1};
    int target = 4;
    vector<pair<int, int>> result = findPairsWithSum(arr,
target);
    for (auto& p : result)
        cout << "(" << p.first << ", " << p.second << ") ";  //
Output: (1, 3) (2, 2) (3, 1)
    return 0;
}
```

15

count number of inversions

```
#include <iostream>
#include <vector>
using namespace std;
int mergeAndCount(vector<int>& arr, int left, int right) {
    if (left >= right) return 0;
    int mid = left + (right - left) / 2;
    int invCount = mergeAndCount(arr, left, mid);
    invCount += mergeAndCount(arr, mid + 1, right);
    // Merge step and count inversions
    invCount += merge(arr, left, mid, right);
    return invCount;
}
int merge(vector<int>& arr, int left, int mid, int right) {
    int invCount = 0;
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> leftArr(n1), rightArr(n2);
    for (int i = 0; i < n1; i++) leftArr[i] = arr[left + i];
    for (int i = 0; i < n2; i++) rightArr[i] = arr[mid + 1 + i];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k++] = leftArr[i++];
        } else {
            arr[k++] = rightArr[j++];
            invCount += (n1 - i);  // Count inversions
        }
    }
    while (i < n1) arr[k++] = leftArr[i++];
    while (j < n2) arr[k++] = rightArr[j++];
    return invCount;
}
int countInversions(vector<int>& arr) {
    return mergeAndCount(arr, 0, arr.size() - 1);
}
int main() {
    vector<int> arr = {1, 20, 6, 4, 5};
    cout << countInversions(arr) << endl;  // Output: 5
    return 0;
}
```

16

find kth largest element

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int findKthLargest(vector<int>& nums, int k) {
    priority_queue<int, vector<int>, greater<int>>
minHeap;
    for (int num : nums) {
        minHeap.push(num);
        if (minHeap.size() > k) {
            minHeap.pop();
        }
    }
    return minHeap.top();
}
int main() {
    vector<int> arr = {3, 2, 1, 5, 6, 4};
    int k = 2;
    cout << findKthLargest(arr, k) << endl;  // Output: 5
    return 0;
}
```