```
// 01)Given an integer array nums, find the
// subarray with the largest sum, and return its sum.
// Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
// Output: 6
// Explanation: The subarray [4,-1,2,1] has the largest sum 6.

#include <iostream>
using namespace std;

int maxSubArray(int nums[], int size) {
    int max_sum = nums[0];
    int current_sum = nums[0];

    for (int i = 1; i < size; i++) {
        current_sum = max(nums[i], current_sum + nums[i]);
        max_sum = max(max_sum, current_sum);
    }
    return max_sum;
}

int main() {
    int nums[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int size = sizeof(nums) / sizeof(nums[0]);
    cout << "Maximum subarray sum: " << maxSubArray(nums, size) << endl;
    return 0;
}

// 02)Given an integer array arr[] of size n, the task is to find the count
inversions of the given array.
// Two array elements arr[i] and arr[j] form an inversion if arr[i] > arr[j] and i
< j.
// Input: arr[] = {7, 2, 6, 3}
// Output: 4
// Explanation: Given array has 4 inversions: (7, 2), (7, 6), (7, 3), (6, 3)

#include <iostream>
using namespace std;

int countInversions(int arr[], int n) {
    int count = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                count++; // Inversion found
            }
        }
    }
    return count;
}

int main() {
    int arr[] = {7, 2, 6, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of inversions: " << countInversions(arr, n) << endl;
    return 0;
}

// Merge Sort Approach
```

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

int mergeAndCount(int arr[], int temp[], int left, int mid, int right) {
    int i = left;  // Starting index for left subarray
    int j = mid + 1;  // Starting index for right subarray
    int k = left;  // Starting index to be sorted
    int inv_count = 0;

    while ((i <= mid) && (j <= right)) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
            inv_count += (mid - i + 1);  // Number of inversions
        }
    }

    while (i <= mid)
        temp[k++] = arr[i++];

    while (j <= right)
        temp[k++] = arr[j++];

    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}

int mergeSortAndCount(int arr[], int temp[], int left, int right) {
    int inv_count = 0;
    if (left < right) {
        int mid = (left + right) / 2;

        inv_count += mergeSortAndCount(arr, temp, left, mid);
        inv_count += mergeSortAndCount(arr, temp, mid + 1, right);
        inv_count += mergeAndCount(arr, temp, left, mid, right);
    }
    return inv_count;
}

int countInversions(int arr[], int n) {
    int temp[n];
    return mergeSortAndCount(arr, temp, 0, n - 1);
}

int main() {
    int arr[] = {7, 2, 6, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Number of inversions: " << countInversions(arr, n) << endl;
    return 0;
}


// 3)Given an integer array nums and an integer k, return the kth largest element
in the array.
// Note that it is the kth largest element in the sorted order, not the kth
distinct element.
```

```
// Solve it without sorting.
// Example:
// Input: nums = [3,2,1,5,6,4], k = 2
// Output: 5

#include <iostream>
using namespace std;

const int MIN_VALUE = -1000000;

int findKthLargest(int arr[], int n, int k) {
    int maxElement;

    for (int i = 0; i < k; i++) {
        maxElement = MIN_VALUE;

        for (int j = 0; j < n; j++) {
            if (arr[j] > maxElement) {
                maxElement = arr[j];
            }
        }

        // After finding the max element, mark it as a very small number
        // so it is ignored in the next iteration
        for (int j = 0; j < n; j++) {
            if (arr[j] == maxElement) {
                arr[j] = MIN_VALUE;
                break;
            }
        }
    }

    return maxElement;
}

int main() {
    int arr[] = {3, 2, 1, 5, 6, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;

    int result = findKthLargest(arr, n, k);
    cout << "The " << k << "th largest element is: " << result << endl;

    return 0;
}

// 8)Given an array of integers of size ‘n’, Our aim is to calculate the maximum
sum of ‘k’
// consecutive elements in the array(Using Sliding Window Technique)
// Input : arr[] = {100, 200, 300, 400}, k = 2
// Output : 700
// Input : arr[] = {1, 4, 2, 10, 23, 3, 1, 0, 20}, k = 4
// Output : 39
// We get maximum sum by adding subarray {4, 2, 10, 23} of size 4.
// Input : arr[] = {2, 3}, k = 3
// Output : Invalid
// There is no subarray of size 3 as size of whole array is 2.

#include <iostream>
```

```
using namespace std;

int maxSumOfKConsecutive(int arr[], int n, int k) {
    if (k > n) {
        cout << "Invalid" << endl;
        return -1;
    }


    int maxSum = 0;
    for (int i = 0; i < k; i++) {
        maxSum += arr[i];
    }

    int windowSum = maxSum;


    for (int i = k; i < n; i++) {
        windowSum += arr[i] - arr[i - k];
        maxSum = max(maxSum, windowSum);
    }

    return maxSum;
}

int main() {
    int arr1[] = {100, 200, 300, 400};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int k1 = 2;
    cout << "Max sum of " << k1 << " consecutive elements: " <<
maxSumOfKConsecutive(arr1, n1, k1) << endl;

    int arr2[] = {1, 4, 2, 10, 23, 3, 1, 0, 20};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    int k2 = 4;
    cout << "Max sum of " << k2 << " consecutive elements: " <<
maxSumOfKConsecutive(arr2, n2, k2) << endl;

    int arr3[] = {2, 3};
    int n3 = sizeof(arr3) / sizeof(arr3[0]);
    int k3 = 3;
    maxSumOfKConsecutive(arr3, n3, k3);

    return 0;
}

// 9)You are given an array of prices where prices[i] is the price of a given stock
on an ith day. You
// want to maximize your profit by choosing a single day to buy one stock and
choosing a
// different day in the future to sell that stock. Return the maximum profit you
can achieve from
// this transaction. If you cannot achieve any profit, return 0
// Input: prices = [7,1,5,3,6,4]
// Output: 5
// Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-
1 = 5

#include <iostream>
```

```
using namespace std;

int maxProfit(int prices[], int n) {
    if (n == 0) return 0;
    int minPrice = prices[0];
    int maxProfit = 0;


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

        if (prices[i] < minPrice) {
            minPrice = prices[i];
        }


        int profit = prices[i] - minPrice;


        if (profit > maxProfit) {
            maxProfit = profit;
        }
    }

    return maxProfit;
}

int main() {
    int prices[] = {7, 1, 5, 3, 6, 4};
    int n = sizeof(prices) / sizeof(prices[0]);

    int result = maxProfit(prices, n);
    cout << "Maximum profit: " << result << endl;

    return 0;
}
```