1. Find the Second Largest Element:

Input:

Array: {12, 35, 1, 10, 34, 1}

Output:

The second largest element is: 34

Code:

```
#include <iostream>
using namespace std;
int findSecondLargest(int arr[], int n) {
   int largest = INT_MIN;
   int secondLargest = INT_MIN;

   for (int i = 0; i < n; i++) {
      if (arr[i] > largest) {
         secondLargest = largest;
         largest = arr[i];
      } else if (arr[i] > secondLargest && arr[i] != largest) {
         secondLargest = arr[i];
      }
   }
   return secondLargest;
}

int main() {
   int arr[] = {12, 35, 1, 10, 34, 1};
   int n = sizeof(arr) / sizeof(arr[0]);

   int result = findSecondLargest(arr, n);
   cout << "The second largest element is: " << result << endl;

   return 0;
}
```

2. Array Rotation:

Input:
Array: {1, 2, 3, 4, 5, 6, 7}
Number of steps to rotate: 3

Output:
Rotated array: 5 6 7 1 2 3 4
Code:

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

void rotateArray(int arr[], int n, int k) {
   k %= n;  // In case k is greater than n
   reverse(arr, arr + n);
   reverse(arr, arr + k);
   reverse(arr + k, arr + n);
}

int main() {
   int arr[] = {1, 2, 3, 4, 5, 6, 7};
   int n = sizeof(arr) / sizeof(arr[0]);
   int k = 3;  // Number of steps to rotate

   rotateArray(arr, n, k);

   cout << "Rotated array: ";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }

   return 0;
}
```

3. Intersection of Two Arrays:

Array 1: {1, 2, 2, 1}
Array 2: {2, 2}
Output:

Intersection: {2}

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

vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
    unordered_set<int> set1(nums1.begin(), nums1.end());
    unordered_set<int> set2;
    vector<int> result;

    for (int num : nums2) {
        if (set1.count(num) && !set2.count(num)) {
            result.push_back(num);
            set2.insert(num);
        }
    }
    return result;
}

int main() {
    vector<int> nums1 = {1, 2, 2, 1};
    vector<int> nums2 = {2, 2};

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

    cout << "Intersection: {";
    for (int i = 0; i < result.size(); i++) {
        cout << result[i];
        if (i < result.size() - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;

    return 0;
}
```

4. Sum of Subarrays:

Input:

Array: {1, -2, 3, 10, -4, 7, 2, -5}
Output:
Maximum subarray sum: 18

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

int maxSubArraySum(vector<int>& nums) {
   int maxSum = INT_MIN;
   int currentSum = 0;

   for (int num : nums) {
      currentSum = max(num, currentSum + num);
      maxSum = max(maxSum, currentSum);
   }
   return maxSum;
}

int main() {
   vector<int> nums = {1, -2, 3, 10, -4, 7, 2, -5};
   int result = maxSubArraySum(nums);

   cout << "Maximum subarray sum: " << result << endl;

   return 0;
}
```

Merge Two Sorted Arrays:

Input:
Array 1: {1, 3, 5, 7}
Array 2: {2, 4, 6, 8}
Output:
Merged Array: {1, 2, 3, 4, 5, 6, 7, 8}

```
#include <iostream>
using namespace std;
void mergeSortedArrays(int arr1[], int n1, int arr2[], int n2, int result[]) {
   int i = 0, j = 0, k = 0;

   while (i < n1 && j < n2) {
      if (arr1[i] < arr2[j]) {
         result[k++] = arr1[i++];
      } else {
         result[k++] = arr2[j++];
      }
   }

   while (i < n1) {
      result[k++] = arr1[i++];
   }

   while (j < n2) {
      result[k++] = arr2[j++];
   }
}

int main() {
   int arr1[] = {1, 3, 5, 7};
   int n1 = sizeof(arr1) / sizeof(arr1[0]);

   int arr2[] = {2, 4, 6, 8};
   int n2 = sizeof(arr2) / sizeof(arr2[0]);

   int mergedArray[n1 + n2];

   mergeSortedArrays(arr1, n1, arr2, n2, mergedArray);

   cout << "Merged Array: {";
   for (int i = 0; i < n1 + n2; i++) {
      cout << mergedArray[i];
```

```
        if (i < n1 + n2 - 1) {
            cout << ", ";
        }
    }
    cout << "}" << endl;

    return 0;
}
```

Find the Missing Number:

Input:

Array: {0, 1, 2, 3, 5, 6, 7}
Output:

Missing Number: 4

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

int findMissingNumber(int arr[], int n) {
    int total = (n + 1) * (n + 2) / 2;
    for (int i = 0; i < n; i++) {
        total -= arr[i];
    }
    return total;
}

int main() {
    int arr[] = {0, 1, 2, 3, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    int missingNumber = findMissingNumber(arr, n);

    cout << "Missing Number: " << missingNumber << endl;

    return 0;
}
```

Largest Subarray with Equal Number of 0s and 1s:

Input:
Binary Array: {0, 1, 0, 1, 1, 1, 0, 0}

Output:

Largest Subarray: [0, 1, 0, 1, 1, 1]

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

void findLargestSubarrayWithEqualZerosAndOnes(int arr[], int n) {
   unordered_map<int, int> prefixSum; // Map to store prefix sums
   int maxLength = 0; // Length of the largest subarray
   int endIndex = -1; // Ending index of the largest subarray
   int sum = 0; // Current sum

   for (int i = 0; i < n; i++) {
      if (arr[i] == 0) {
         sum -= 1;
      } else {
         sum += 1;
      }

      if (sum == 0) {
         maxLength = i + 1;
         endIndex = i;
      }

      if (prefixSum.find(sum) != prefixSum.end()) {
         if (i - prefixSum[sum] > maxLength) {
            maxLength = i - prefixSum[sum];
            endIndex = i;
         }
      } else {
         prefixSum[sum] = i;
      }
   }

   if (maxLength > 0) {
      cout << "Largest Subarray: [";
      for (int i = endIndex - maxLength + 1; i <= endIndex; i++) {
```

```
            cout << arr[i];
            if (i < endIndex) {
                cout << ", ";
            }
        }
        cout << "]" << endl;
    } else {
        cout << "No subarray with equal 0s and 1s found." << endl;
    }
}

int main() {
    int arr[] = {0, 1, 0, 1, 1, 1, 0, 0};
    int n = sizeof(arr) / sizeof(arr[0]);

    findLargestSubarrayWithEqualZerosAndOnes(arr, n);

    return 0;
}
```

Find Pairs with Given Sum:

Input:
Array: {1, 2, 3, 4, 5, 6, 7}
Target Sum: 9

Output:

Pairs with Sum 9: (2, 7), (3, 6), (4, 5)

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

void findPairsWithSum(int arr[], int n, int targetSum) {
    unordered_set<int> seen;

    cout << "Pairs with Sum " << targetSum << ": ";
    for (int i = 0; i < n; i++) {
        int complement = targetSum - arr[i];
        if (seen.find(complement) != seen.end()) {
            cout << "(" << arr[i] << ", " << complement << ") ";
        }
```

```
        seen.insert(arr[i]);
    }
    cout << endl;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int targetSum = 9;

    findPairsWithSum(arr, n, targetSum);

    return 0;
}
```

Find Duplicate Number in Array:

Input:
Array: {3, 1, 3, 4, 2}

Output:

Duplicate Number: 3

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

int findDuplicateNumber(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int absValue = abs(arr[i]);
        if (arr[absValue] < 0) {
            return absValue;
        }
        arr[absValue] = -arr[absValue];
    }
    return -1; // If no duplicate found
}

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

    int duplicateNumber = findDuplicateNumber(arr, n);
```

```
    cout << "Duplicate Number: " << duplicateNumber << endl;

    return 0;
}
```

Find the Majority Element:

Input:

Array: {2, 2, 1, 1, 1, 2, 2}

Output:

Majority Element: 2

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

int findMajorityElement(int arr[], int n) {
    int candidate = arr[0];
    int count = 1;

    for (int i = 1; i < n; i++) {
        if (arr[i] == candidate) {
            count++;
        } else {
            count--;
            if (count == 0) {
                candidate = arr[i];
                count = 1;
            }
        }
    }

    // Verify if the candidate is the majority element
    count = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == candidate) {
            count++;
        }
    }

    if (count > n / 2) {
        return candidate;
```

```
    } else {
        return -1; // If no majority element found
    }
}

int main() {
    int arr[] = {2, 2, 1, 1, 1, 2, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    int majorityElement = findMajorityElement(arr, n);

    if (majorityElement != -1) {
        cout << "Majority Element: " << majorityElement << endl;
    } else {
        cout << "No majority element found." << endl;
    }

    return 0;
}
```