Punto 1)

```cpp
class Solution {
public:
    std::vector<int> twoSum(std::vector<int>& nums, int target) {
        std::unordered_map<int, int> numToIndex;
        std::vector<int> result;

        for (int i = 0; i < nums.size(); ++i) {
            int complement = target - nums[i];

            if (numToIndex.find(complement) != numToIndex.end()) {
                result.push_back(numToIndex[complement]);
                result.push_back(i);
                break; // Se asume que hay una única solución.
            }

            numToIndex[nums[i]] = i;
        }

        return result;
    }
};
```

Punto2)

```cpp
class Solution {
public:
    int removeDuplicates(std::vector<int>& nums) {
        if (nums.empty()) {
            return 0;
        }

        int uniqueCount = 1; // Inicialmente, al menos hay un elemento único.

        for (int i = 1; i < nums.size(); ++i) {
            if (nums[i] != nums[i - 1]) {
                nums[uniqueCount] = nums[i];
                ++uniqueCount;
            }
        }

        return uniqueCount;
    }
};
```

Punto 3)

```
class Solution {
public:
    int removeElement(std::vector<int>& nums, int val) {
        int k = 0; // Contador de elementos que no son iguales a val.

        for (int i = 0; i < nums.size(); ++i) {
            if (nums[i] != val) {
                nums[k] = nums[i];
                ++k;
            }
        }

        return k;
    }
};
```

Punto 4)

```
class Solution {
public:
    double findMedianSortedArrays(std::vector<int>& nums1, std::vector<int>&
nums2) {
        // Asegurémonos de que nums1 sea siempre la matriz más pequeña.
        if (nums1.size() > nums2.size()) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int m = nums1.size();
        int n = nums2.size();

        int left = 0, right = m;
        int partitionX, partitionY;
        int maxX, maxY, minX, minY;
        double median = 0.0;

        while (left <= right) {
            partitionX = (left + right) / 2;
            partitionY = (m + n + 1) / 2 - partitionX;

            maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
            maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];

            minX = (partitionX == m) ? INT_MAX : nums1[partitionX];
            minY = (partitionY == n) ? INT_MAX : nums2[partitionY];
```

```
            if (maxX <= minY && maxY <= minX) {
                if ((m + n) % 2 == 0) {
                    median = (std::max(maxX, maxY) + std::min(minX, minY)) /
2.0;
                } else {
                    median = std::max(maxX, maxY);
                }
                break;
            } else if (maxX > minY) {
                right = partitionX - 1;
            } else {
                left = partitionX + 1;
            }
        }

        return median;
    }
};
```