Easy 1

```
#include <iostream>
#include <string>

using namespace std;

int lengthOfLastWord(const string& s) {
    int length = 0;
    int i = s.size() - 1;

    // Skip trailing spaces
    while (i >= 0 && s[i] == ' ') {
        i--;
    }

    while (i >= 0 && s[i] != ' ') {
        length++;
        i--;
    }

    return length;
}

int main() {
    string input = "Hello World";
    int result = lengthOfLastWord(input);

    cout << "Input: " << input << endl;
    cout << "Output: " << result << endl;

    return 0;
}
```

Explanation:

The function lengthOfLastWord takes a string s as input and returns the length of the last word.
We start from the end of the string and skip any trailing spaces.
Then, we count the characters until a space or the beginning of the string is encountered.

Algorithm:

Initialize a variable length to 0.
Start iterating from the end of the string.
Skip any trailing spaces.
Count the characters until a space or the beginning of the string is encountered.

Return the length.

Medium 2:

```
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;

vector<int> majorityElement(const vector<int>& nums) {
    unordered_map<int, int> countMap;
    vector<int> result;

    for (int num : nums) {
        countMap[num]++;
    }


    for (const auto& entry : countMap) {
        if (entry.second > nums.size() / 3) {

            result.push_back(entry.first);
        }
    }

    return result;
}

int main() {

    vector<int> nums = {3, 2, 3};
    vector<int> result = majorityElement(nums);


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

    cout << "Output: [";
    for (int i = 0; i < result.size(); ++i) {
```

```
        cout << result[i];
        if (i < result.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;

    return 0;
}
```

Algorithm:

Counting Occurrences:

Traverse the given array nums and use a hash map (countMap) to store the count of each element.
Checking Majority Elements:

Iterate through the hash map entries and check if the count of any element is more than ⌊ n/3 ⌋, where n is the size of the array.
Building Result:

If the count of an element is more than ⌊ n/3 ⌋, add that element to the result vector.

Explanation:

The algorithm uses a hash map to count the occurrences of each element in the array.
Then, it iterates through the hash map to identify elements that occur more than ⌊ n/3 ⌋ times.
The result vector is used to store the elements that satisfy the condition.
In the given example [3, 2, 3], the output is [3], as 3 is the only element that appears more than ⌊ n/3 ⌋ times.

HARD 2:

```
#include <iostream>
#include <string>

using namespace std;

string shortestPalindrome(string s) {

    string rev_s = s;
    reverse(rev_s.begin(), rev_s.end());
```

```
    int n = s.length();


    for (int i = n; i >= 0; --i) {
        if (s.substr(0, i) == rev_s.substr(n - i)) {

            return rev_s.substr(0, n - i) + s;
        }
    }


    return "";
}

int main() {

    string s = "aacecaaa";
    string result = shortestPalindrome(s);

    cout << "Input: " << s << endl;
    cout << "Output: " << result << endl;

    return 0;
}
```

Explanation:

Reversing the String:

rev_s is created by reversing the original string s using the reverse function.
Finding the Longest Palindrome Prefix:

The for loop iterates from the length of the string (n) to 0.
Inside the loop, it checks if the substring of s from index 0 to i is equal to the substring of rev_s from the end of rev_s minus i to the end of rev_s.
This is essentially checking if the first i characters of s form a palindrome when compared to the last i characters of the reversed string rev_s.
If a match is found, it means we have found the longest palindrome prefix.
Building the Result:

If a palindrome prefix is found, the algorithm adds the remaining characters of the original string (excluding the palindrome prefix) in reverse order to the front.
The result is the concatenation of the reversed substring of rev_s and the original string s.
Output:

The main function initializes a string s with the value "aacecaaa" and calls the shortestPalindrome function.
It then outputs the input string and the result.