Complete Analysis of the task what algorithm did you use? Why?

The problem we had to solve in this task was to remove any duplicate characters from an array. The algorithm I opted to use was a built-in function called unique(). The function checks for any consecutive duplicate elements. If we find two characters that are the same, the first iteration of the character will be moved forward, therefore overwriting the duplicate character. Then we repeat this process until we have no more duplicates. In order for the unique() function to properly work we have to sort the array first in alphabetical order. The function will then return a new size of the array that I can store in a variable to navigate only unique duplicate characters. But so far we are only printing out non-duplicate characters but not actually removing them from the array. So we store the new size in a vector since we cannot just save it in a new array since we cannot use a non-static variable to create a new array. However, a vector will allow us to change its size during runtime and therefore we can use a non-static variable. So now we successfully created a new array that does not include any non-duplicate characters. The reason why I chose this specific algorithm is because of its simplicity and my familiarity with it. Also, the time complexity of the unique() function is only O(n). The only downside of this approach is that if the order of the original character array matters we are better off using another algorithm that does not require us to sort the array.

What other algorithms apply to this task?

We could make a custom-made algorithm that will loop through the entire array to check for duplicate values. For example, we could save the first iteration of a character in a new array, and then, for all of the characters in the array, we will constantly check if this specific character is in the second list. If it is not we add it to the list otherwise we will just continue. Or we could simply use any other existing library that would remove duplicate characters.

In what ways does the algorithm you used differ from these other algorithms?

As mentioned earlier one key aspect that sets the approach I have used here from other algorithms is that it does not preserve the original order of the array. If we used the custom-made algorithm like mentioned above we would still preserve the original order of the array.

How does the code work?

The code is fully commented to explain how each function works

Complexity analysis of the algorithm

While removing duplicates using the unique() function only takes a linear time of O(n), the overall time complexity is dominated by when we sort our character array which is O(n log n).

Code:

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

int main() {
   // Duplicate character array
   char arr[] = { 'X', 'Y', 'Y', 'X', 'Z', 'X' };
   int Size = sizeof(arr) / sizeof(arr[0]);

   // Sorting the array before applying the unique function
   sort(arr, arr + Size);

   // Using unique to remove duplicate characters.
   auto End = unique(arr, arr + Size);

   // Calculating the new size of the array.
   int Size2 = End - arr;

   // Check if there were duplicate values.
   bool duplicates = Size2 < Size;

   // Printing the array before removing duplicates
   cout << "Character array before removing duplicates: ";
   for (int i = 0; i < Size; ++i) {
      cout << arr[i] << " ";
   }
   cout << endl;
```

```
    // If there were duplicates
    if (duplicates == true) {
        cout << "There were duplicate values in the array." << endl;

        // Creating a new vector to hold the unique elements (We have to use vectors since we cannot use a non static variable like size 2. Vectors allow us to change the size in runtime)
        vector<char> newArr(arr, arr + Size2);

        // Printing the results after removing duplicate characters
        cout << "Character array after removing duplicate characters: ";
        for (char c : newArr) {
            cout << c << " ";
        }
        cout << endl;
    }
    // In case there were no duplicate characters
    else {
        cout << "There were no duplicate characters in the array." << endl;
    }

    return 0;
}
```