Stephen Townsend
7/27/16

Assignment 7 (Comparison)

When I started looking at my teammates codes, I decided to first figure out how they decided to structure their code. This was a way for me quickly and visually see how they went about testing for each element in the array. Through this assignment I will have my group members' code structure and a snippet of code I will be talking about.

1. Sort Array
2. Frequency
3. Test for mode / Push to vector

To the left is my structure and how I achieved my results:

Comparison 1

Sylvan's Code

1. Frequency
2. Test for mode
3. Check duplicates
4. Push to vector
5. Sort – Vector

First off I am going to compare Sylvan's code. When I was looking at his code, I noticed that there were many nested for loops. This caused a lot of looping to occur in the program.

Below is a snippet of code:

```
// First iteration through the array is to compute the highest occurrence of any one
or more integer values
   for (int i = 0; i < size; i++) {
      value = array[i];
      occurrence = 0;

      // Nested loop iterates through the entire array again and counts occurrences of
this particular value
      for (int j = 0; j < size; j++)
         if (array[j] == value)
            occurrence++;

   // Update highestOccurrence if this value occurs more than the previous
highestOccurrence
      if (occurrence > highestOccurrence)
         highestOccurrence = occurrence;
   }
```

This code is used to check for mode occurrences, this iterates through each array element twice. I find that this is the biggest down fall of the code. In regard to my code, I only used one for loop with if else statements to achieve the same outcome. Thus, my code has better performance than Sylvan's code.

Later on finding the mode, he managed to use a flag. Even though this just added to the complex looping, this is something that stood out specifically. This allowed him create a special check for duplicates before pushing an element into the vector. I feel it was unnecessary though and that he should have incorporated the check into one of his many loops.

The one thing I liked on Sylvan's code was the documentation. There were plenty of comments that were very descriptive. You can see this in the snippet of code above. I was able to understand the purpose of sections of code quickly.

Comparison 2

Shannon's Code

1. Sort – Array
2. Frequency
3. Test for mode / Push to vector
4. Sort – Vector

Shannon's code was structured almost like mine. The only difference was the sort at the end. The second sort is not needed because the array was already sorted. The elements will be pushed into the vector in order they occur in array.

Below is a snippet of code:

```
//loop to add numbers that meat previous num dups to vector
for (int i = 0; i < size; i++)
{
	if (array[i] == num)
	{
		if (count == modeCount)
		{
			final.push_back(array[i]);
			num = array[i + 1];
			count = 0;
		}
	}
	else
	{
		num = array[i];
		count = 1;
```

Shannon had the code that the collective has chosen as the best. There is not much that I can find wrong with Shannon's code. It was simple, clean, and easy to follow. The only aspect that I did not like is the aesthetics of not having many comments.

Overall, Shannon's code was superior due to its readability and being easier to follow. Compared to the three separate functions that I made. My code was superior where it did not have any extras such as her sort.

Comparison 3

Jonathan's Code

1. Sort – Array
2. Frequency / Test for mode / Clear vector / Push to vector

Jonathan's code was by far the highest performing out of the group. He was able to use one for loop to correctly find the modes and iterate through the array once.

Below is a snippet of code:

```
// do we have a new mode?
    if (frequencyCounter >= maxFrequency)
    {
      // does the current value have a higher frequency then previously identified
modes?
      if (frequencyCounter > maxFrequency)
      {
        // if so, clear all the previously identified modes, since they aren't really
modes after all
        result.clear();
        maxFrequency = frequencyCounter; // then set the maxFrequency to the current
value of frequencyCounter
      }
```

I found It interesting how he used the result.clear(); command. The reason being, the code does not separate out to find max frequency then compare that to find the mode. Instead he went with a way to take the current max frequency and push that element into the vector. If later a higher frequency was found, then the clear command cleared the vectors contents and pushed the new element as a mode. For how the code was structured, this is a nice solution.

From looking at the portion of the code, above, with the two if statements, at first sight it feels like it is duplicating code because frequencyCounter is being compared twice with maxFrequency. However, I do understand the purpose and once I drilled in, I found why he did it.

I think for that even though his code is performance based, there is a little too much happening under that one for loop. It would be better to separate the work with more loops. I did this in my code by using functions to separate each new calculation or manipulation of the array. By using functions, readability should be improved.

The biggest fault I found was that he code was missing the #include <algorithm>. This is an issue because you should not assume that someone else's main function will include this header. As a result, the sort that we needed to include will not work without it. I confirmed this by running the code and needed to add the header to compile.

Overall, Jonathan had a strong code, that would only perform better than mine. The downside, where mine is superior, is in upgradability and modification. It is easier to focus on one section of code to debug then the code as a whole due to integration of multiple calculations.

Learning/Improvements

In regards to future assignments, I stated previously that "by using functions, readability should be improved." However, my group did not agree. It seemed that my code was hindered by this and harder to follow. However, in the future it may be best to continue to use functions, but I may want to make my comments more detailed. I can do this by stating transitions from function calls. I would like to keep functions so that there is a separation of duties with the calculations. When I was debugging my code, it was a lot easier to look at the frequency and then go to the overloaded findMode (three parameter)

function. Besides the detailed commenting, I may want to incorporate a more optimized solution in my code. Similar to Jonathan's code, but still separated.