Jackie Diep

Professor Bo Li

CSC 413

6 April 2021

Programming Assignment: Inversion Count

Abstract: Tracking the number of inversions remaining in an array is a method to view how close an array is to being sorted. Our project is to sort the numbers given in a .txt file from 1 to 100000 in ascending order, and we are to track the number of inversions present in the array. This paper will go into how I implemented this algorithm and the struggles I had with it.

Problem Description:

The course programming assignment of our class, Inversion Count, involves sorting through an array and counting the number of inversions within the array as it is sorted. This algorithm uses a given .txt file consisting of 100000 numbers. This file has two properties: it holds all integers inclusively from 1 to 100000 and no integers are repeated. In the words of the project description,"... two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j." The project also suggests using a Divide-and-Conquer method to handle the large size of the array, and in this case, merge-sort is the algorithm that I implemented.

Pseudocode:

Pseudocode for “merge-sort with inversion counting” referencing “Counting Inversions in an array” in my references

Function to divide an array into two and recursively call itself until there is only one

```
{
Initialize variables to divide the array and track inversions
Create a structure to test for recursion conditions (Until one element remains){
    Recursively call itself until the test is failed
    Call merge to combine the divided arrays and count inversions
    }
Return inversion count to main
}
```

Function to merge the arrays and count inversions

```
{
Initialize variables to hold locations for the left, middle, and temporary left indexes of the
subarrays
    as well as a variable to hold inversion count values
Create a structure to loop until the left index reaches the middle or the middle index reaches the
    right{
    If the value of the left index is less than or equal to the value of the middle index of the
        array{
            Set the value of the temp array at the current temporary left index equal to
            the value of the array being sorted at the left index and increment both.
            }
    Else{
        Set the value of the temp array at the current temporary left index equal to
        the value of the array being sorted at the middle index and increment both.
        Calculate the amount of inversions and add to the inversion count
        }
    }
Iterate through the rest of the array, copying the elements to temp starting from the left index and
    ending at the right index
Copy the completed temp array to the original array
Return inversion count
}
```

Implementation:

My code utilizes four functions in addition to main. The four functions are mergeSort, merge, readFile, and writeFile.

Starting with main, I initialize two blank arrays of size 100000: toCount and tempArray, along with a pointer to toCount named *tC. I then initialize two strings to hold the file names for input and output. Next, I call the readFile function passing in the pointer to toCount and the input file name.

From here, the readFile function simply initializes a temp variable and opens an ifstream. The values of the .txt file are then read into the toCount array through the *tC pointer until either end-of-file or the index exceeds 99999, and then the file is closed.

MergeSort is then called from main, passing in toCount, tempArray, 0, and 99999. After initializations, an if structure is used to maintain the recursion until the end with the condition that the right index is greater than the left index. This should hold true until there is only one value in the array. Inside of the loop, the middle of the array is calculated and the three functions are called in sequence. The first call is a mergeSort recursive call that uses the midArray as the right index. The second call is a mergeSort recursive call that uses the midArray + 1 as the left index. The final call is to the merge function to combine the divided arrays and count inversions.

Inside of merge, three indexes are used to iterate through the array: the left index of the array, the middle index of the array, and the left index of the temporary array. Until the left index reaches the middle of the array or the middle index reaches the right of the array, values of the array are iterated through until the condition is reached. Inversions are calculated in this section when the value of the left index is greater than the value of the middle index with "inversionCount = inversionCount + (midArray - left)". Afterwards, the rest of the array is iterated through with the opposite condition of the one that ended the previous loop, and the sorted values of the temporary loop are set as the values of the original loop and inversion count is returned to mergeSort.

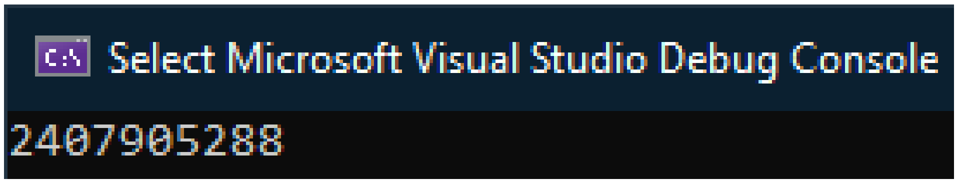
After all recursions have been merged, the array has been sorted and the inversion count has been returned to main. From here, main simply outputs the inversion count and prints the finished array to a file.

Problems:

In total, I only truly had two important issues in this project. My first issue was wrapping my mind around recursions and how to properly use them to count inversions. I was not properly grasping the divide-and-conquer aspect of the assignment, and I was not setting the end condition of the recursion properly. This required a lot of brainstorming and research until I understood when to stop and how to utilize the pieces that I was left with after the end of recursion. The more infuriating problem that I had was also the most simple problem I have run into in a long time. The value that I was using to hold inversion count was an integer, and the value that I was getting for the inversion count was too large. This resulted in overflow and I was left with a nonsensical negative number that took me several hours to solve. Eventually, I realized the issue and changed the data type of inversion count and the two merge functions to "long long" instead of "int".

Running Results Outputted to .txt file "SortedIntegerArray.txt"

My final results of the project are a completely sorted array from 1-100000 inclusive in ascending order, and an inversion count of 2,407,905,288. Because of the size of the array and answer it is difficult to be entirely sure of its correctness; however, all test arrays I used to count inversions completed with correct answers. Due to my testing and my answer being within expected bounds of 0 < x < 4,999,950,000, I am led to believe that my algorithm is correct.



Select Microsoft Visual Studio Debug Console

```
2407905288
```

Conclusions:

The main thing I feel that I have learned from this project is the understanding of how to begin comprehending recursions. Working from beginning towards the end of the recursion felt impossible, and I was not making progress by trying to program this way. Working starting from the beginning and end conditions and formulating how to get from point A to point B was much more helpful, and I understand the importance of loop invariants much better. Previously, I considered loop invariants as a way to prove things rather than create things, but it is pretty clear that they are necessary in all things recursion related. While the overflow issue with data types was a big issue for me, I definitely learned more about loop invariants and recursions than I did fixing that issue.

References:

1. Count inversions in an array: Set 1 (Using merge sort). (2021, March 22). Retrieved April 07, 2021, from https://www.geeksforgeeks.org/counting-inversions/