**2-3 Assignment: Vector Sorting – Code Reflection and Pseudocode**

Alex Baires

Southern New Hampshire University

CS-300-H2999 DSA: Analysis and Design

Professor Nathan Lebel

November 5th, 2023

**Code Reflection:**

The purpose of the VectorSorting.cpp program is to load a .csv file of simulated, unsorted bids for an auction of property and to compare the efficiency of sorting this data between the selection sort and quick sort algorithms. The user can load the bids into a vector and display the unsorted bids. When they run one of the algorithms, they can again display the bids, but their titles will be sorted in alphabetical order this time. After running either of the algorithms, the program will provide the number of clock cycles and seconds it took to sort the data to provide the user with a metric for comparison. Sorting the bids vector with the selection sort algorithm is accomplished by starting from the first cell and iterating through each title cell in the bids vector from left to right while keeping track of the vector index of the lowest title string value found. If a value lower than that in the start index is found, it swaps the lowest value index with the initial start index. Then, it increments the start index (to sort the next cell in the vector) and scans the rest of the vector for the next lowest value. These iterations continue until the start index is incremented to the position of the final index in the vector, which means it has been sorted.

The quick sort algorithm calls a partition method that divides the bids vector into two parts at the midpoint of the bids vector. The bid string value inside the cell at the midpoint is called the pivot. The partition method takes the leftmost element of the vector and moves it one cell at a time until its string value is greater than or equal to the title value of the pivot. It also takes the rightmost element of the vector and decrements it until it finds a title value less than or equal to the pivot. When both conditions are satisfied, the elements are swapped. This repeats until all values to the left of the pivot are less than the pivot, and all values to the right of it are greater than the pivot. These values are not necessarily sorted. The partition method returns the highest index of the lower partition for use in quick sort. The quick sort algorithm first calls a partition method that partitions the entire vector, returning a pivot. The quick sort then recursively calls itself further to divide the left-most and right-most vectors into smaller sub-vectors. This continues until there is a sub-vector with zero or one element. At this point, the bids vector is sorted.

Some of my challenges with developing this program were self-inflicted. I am stubbornly adamant about using my Macbook for class, and, unlike my experiences in CS-210, I had trouble getting the VectorSorting program to run in CLion with Clang. I then used Visual Studio 2022 in a Windows 11 arm64 virtual machine in Parallels, which concerned me because all the school projects run in x86 or x64 mode. Luckily, I had no problems after importing the project files and changing a few settings. During development, my main objective was to get the sort algorithms to sort correctly. I am a big fan of prototyping, so I created a few small programs with vectors of random integers (instead of bid objects) to ensure that my selection sort and quick sort algorithms worked correctly before tackling the assignment. The code from the Zybooks Chapter 3 modules helped. Still, I found Jay Wengrow’s book, “A Common-Sense Guide to Data Structures and Algorithms,” invaluable in understanding and implementing the algorithms in the program. Wengrow’s partition method used the rightmost index as the pivot and started comparing elements with the leftmost element and the rightmost element before the pivot at the final index. I almost implemented that version (and made a prototype), but in my research, computer scientist Robert Sedgewick stated that using the median of the array was one of the most efficient versions of quick sort. The coding was not complex, but I hit a major roadblock with the quick sort algorithm’s partition method. I declared the midpoint as an integer and compared the bids at both the low and high indices to the midpoint itself. My bids were somewhat sorted, with minor errors. The only way to fix the problem was to call a third quickSort() function within the parent quickSort() function. While this provided correct output, I knew this was a workaround and not how quick sort should work. Finally, I realized that I was comparing the title strings to the midpoint index and not to the string within the element. Declaring a string “pivot” variable that is the string title value at the midpoint and using it to compare to my bids title at the lower and higher indices fixed my error.

**Pseudocode:**

START the VectorSorting program.

The program retrieves the path to the .csv file and uses it as a command line argument (ready to be loaded).

DEFINE an empty vector called “bids” to store bid information from the .csv file.

DISPLAY the main menu with 5 options.

IF the user selects option 1:

The program loads the bids utilizing the path to the .csv file.

DISPLAY the total number of bids read into the vector as an integer.

CALCULATE and DISPLAY the elapsed time to load the .csv file into the “bids” vector.

RETURN to DISPLAY the main menu.

IF the user selects option 2:

FOR all iterations, up to but not including the total number of items in the “bids” vector:

DISPLAY the bids read, including ID number, title, and quantity.

RETURN to DISPLAY the main menu.

IF the user selects option 3:

INVOKE the selectionSort function:

FOR all iterations from the first element to the last element in the “bids” vector:

The minimum bid is the first element that is checked.

FOR all iterations from one element to the right of the element being checked up to the last element of the “bids” vector:

IF the bids element being checked is less than the minimum bid:

This is now the newest minimum bid.

IF the minimum bid is less than the element being checked:

SWAP the bid at the element being checked with the minimum bid.

// Remember, we are sorting from left to right.

CONTINUE until the vector is sorted from left to right.

DISPLAY the total number of bids sorted and in the “bids” vector.

CALCULATE and DISPLAY the elapsed time to complete the selection sort.

IF the user selects option 4:

INVOKE the quickSort function:

IF the subarray has 0 or 1 elements:

COMPLETE the quick sort.

DECLARE the midpoint variable and SET as the RESULT of the PARTITION() function.

The partition function takes the “bids” vector and the indices of its first and last elements.

DECLARE a midpoint that is the leftmost/left index added to the result of the rightmost/right index subtracted from the leftmost/left index divided in half.

DECLARE a pivot that is the value of the bids located in the midpoint index.

WHILE true // infinite loop

WHILE the bid element at the left index is less than the pivot:

INCREMENT the left index.

WHILE the bid element at the right index is less than the pivot:

DECREMENT the rightmost index.

IF the left index reaches or passes the right index:

BREAK out of the loop.

ELSE:

Both while loops for left and right indices stop because the condition is no longer met.

SWAP the bids at the left index and the bids at the right index.

CONTINUE WHILE TRUE until BREAK statement met.

RETURN the highest index of the left partition.

RUN above quickSort algorithm recursively using the “bids” vector, left element and newly found midpoint element) to sort the half of the vector portion less than the midpoint.

RUN quickSort algorithm recursively using the “bids” vector, midpoint element plus 1, and right element of the vector portion greater than the midpoint.

// This should provide a vector that is completely sorted.

DISPLAY the total number of bids sorted and in the “bids” vector.

CALCULATE and DISPLAY the elapsed time to complete the quick sort.

DEFAULT user action (they press any number besides 1 through 4)

DISPLAY “Goodbye”

END VectorSorting program.