Billy Hegarty

CS300 Assignment 2-3 Vector Sorting

## Code Reflection

The code “VectorSorting.cpp” reads in data from a .csv file and provides several data engagement choices to the user via a numerical menu. The .csv file location can be specified as a command line argument, otherwise it will default to a file named “eBid\_Monthly\_Sales\_Dec\_2016.csv" located in the same local file as VectorSorting.exe. The .csv file contains rows of data. Each row corresponds to an individual bid and auction for an item, such as computer parts or office furniture. The program organizes each bid in a data structure named “Bid” and assigns attributes bidId, title, fund, and amount. These attributes are automatically read in from the .csv file. All of the bids are stored in a single vector (one-dimensional array). After loading the .csv and storing the bids in a vector, the user can display the data in the terminal by selecting Menu option #2.

Menu option #3 is Selection Sort All Bids. Selection Sort is a sorting algorithm that treats the input as two parts, a sorted part and an unsorted part, and repeatedly selects the proper next value to move from the unsorted part to the end of the sorted part. Selection Sort is easy to code and involves a nested loop. It has a runtime of O(N2).

Menu option #4 is Quick Sort All Bids. Quicksort utilizes another method, Partition, to repeatedly split the input about a midpoint pivot into “low” and “high” parts and recursively sort each of those parts. The Partition method splits the input such that none of the elements in the “high” part is less than any element in the “low” part. These now-separated parts are then recursively fed back into Quicksort algorithm, so that those parts are again divided into lower and higher parts. This process continues until each part contains only one element, meaning that each partition is by definition “sorted”. The values are then stacked back together, preserving the sorted order. Quicksort’s runtime is O(N log N).

I used a JetBrains IDE called CLion instead of Microsoft Visual Studio. CLion is user friendly and provides visual indications of errors, warning, and typos throughout the code. The majority of my work on this assignment was focused on making corrections and edits to the provided code, outside the scope of coding the actual Selection sort, Quick sort, and Partition methods. I kept track of these roadblocks and observations in the list below. For most of these challenges, I utilized the ClangTidy solutions suggested by CLion. If I did not understand the ClangTidy messages, or ran into other errors, I entered the error messages into a search engine and looked for discussions on StackOverflow that address the problem. When these attempts failed, I used the CLion debugging tool and kept track of variable values and datasets using pen and paper. Although this method was time consuming, it helped me understand the flow of the program and made future troubleshooting easier.

Problems with the provided code:

1. The instructions have reference to old versions of Microsoft Visual Studio
   1. /src directory is now called “Source Folder”
   2. Project Explorer pane is now called “Solution Explorer”
2. The activity references C++11 features, but the current version of C++ is now 23 or beyond
3. #include <time.h> is deprecated, should be ctime
4. Line 13 of VectorSorting.cpp says #include “CSVparser.hpp”, should be .cpp
   1. Also, line 4 of “CSVparser.cpp” says the same thing, #include “CSVparser.hpp”, should just be #include “CSVparser.h”
   2. Actually I guess that’s fine, just another style choice I haven’t seen before
5. Why are we using “struct” instead of making a public class and instantiating objects?
6. The “Bid” struct member names are different to the CSV header names… this hinders code readability
7. Line 56, void function has a return line
8. Need to #include <vector>
9. Line 99…
   1. csv::Parser file = csv::Parser(csvPath);
   2. initially wasn’t working because there was a typo in the #include “CSVparser.hpp” line of the CSVparser.cpp file
10. vector<Bid> loadBids(string csvPath) {
    1. csvPath should be a const reference 🡪 loadBids(const string& csvPath)
11. CSVparser.cpp
    1. Should use if(!line.empty()) instead of if (line != “”)
    2. Same for if (.size() == 0)….. should be .empty()
    3. quoted = ((quoted) ? (false) : (true)); /// this line can be simplified to:
       1. quoted = !(quoted);
       2. instead of relying on ternary operator (?)
    4. Several instances of for loops that are not range-based
    5. In the Parser::operator[] function, there is redundant “Parser::”
       1. Original 🡪 return Parser::getRow(rowPosition);
       2. but it can just be 🡪 return getRow(rowPosition);
    6. Style preference, but you don’t need to pass “void” in the argument list of a function definition.
       1. Original 🡪 Parser(void)…
       2. just say 🡪 Parser()…
12. The FIXME messages say to sort “over bid.title”… that doesn’t make sense because bid.title is a string. Unless it wants to list them alphabetically? I think it wants us to sort over bid.bidId so they are ordered numerically… but I will go with what the code says and sort by bid.title
    1. Interesting tidbit, the sorting algorithm sorts 0-9 then A-Z then a-z
13. I had to add the following to the CMakeLists.txt in CLion for the .exe to work:
    1. set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -static")

## Pseudocode

* Define a structure “Bid” with attributes that correspond to the column headers in the data .csv file.
* Define a function to display the attributes of a Bid type
* Define getter method to prompt user to enter new Bids, including all required attributes, from the console
  + Return type Bid structure
* Define a function called “loadBids”
  + Input parameter string for the CSV file path
  + Load bids from the CSV into a vector
  + Return the vector containing type Bid
* Define a function that will partition a vector containing Bids
  + Input parameters Bid vector, begin index, end index
  + Partition the vector into a low part and high part, such that no element in the high part is less than any element in the low part
  + Return the index at which the high partition begins
* Define a function that performs recursive quicksort on a vector of bids
  + Utilize the partition method defined above
* Define a function that performs selection sort on a vector of Bids
* Define a function strToDouble that strips an unwanted character from a string and then converts the string to a double
* Create a main function:
  + Process command line arguments, such as the filepath of the CSV file
  + Create a vector that will contain bids
  + Define a timer variable
  + Display a menu and get user choices
  + Perform actions based on user choice
    - Load bids, display all bids, perform selection sort, perform quicksort, or exit
  + Display a goodbye message after exiting