**Daniel Schween**

**CS 300 Module 2 ‘VectorSorting’ Documentation**

**07/13/2024**

**‘VectorSorting’ Documentation & Pseudocode**

This assignment utilizes a console program to load and sort two data sets from a municipal government. The two data sets contain bids for property auctions submitted by potential buyers. The program’s main functionality is to sort the bids from smallest to highest. In the best interest of the municipal government, the property should be auctioned off to the highest bidder. Additionally, the program clocks the speed or the amount of time the different methods take to sort or load the data.

This assignment explores a few different sorting algorithms to reach the result. The three main methods used are:

* **void selectionSort(vector& bids)**; This function uses a sort algorithm that loops through the elements in the data set linearly and swaps them from least to greatest.
* **void quickSort(vector& bids, int begin, int end);** quicksort() is a recursive function that sorts through the elements via the partition function.
* **int partition(vector& bids, int begin, int end);** This functions aids the quicksort function. It works by dividing the data in the vector by half and sorting accordingly. Searching through half the data can reduce the amount of time spent sorting, thus reducing ticks.

There are several supporting definitions and structures aiding the primary methods and classes:

* **strToDouble()** converts the CSV data into a data type the program can use.
* **Bid()** is a structure used to hold bid information as a unique identifier.
* **displayBid()** is a static method used to display bid information.
* **getBid()** returns the bid information prompted from the user.
* **loadBids()** loads a CSV file containing the bid data into a vector.

Several variables are also used within the methods to aid with functionality, algorithms, and looping. For example, the partition method uses **int low, int high, int middlePoint, int pivot, and bool finished** to help define a sorting algorithm. “**bid.title**” is used as the sort field in the algorithmic logic.

Disregarding data sets and algorithms, the program initially prompts the user to select from 5 different options, including an exit option, from a menu. The choices are:

1. Load Bids
2. Display All Bids
3. SelectionSort All Bids
4. QuickSort All Bids
5. Exit

Depending on the user input, the data sets can be displayed and/or sorted. After being loaded, displayed, and sorted, the municipal government can select a buyer based on bid value. Sorting the elements in the vector based on value makes it easier for an individual to select a buyer.

My main challenge was implementing the partition function to coincide with the quickSort function. Initially, I did not understand the concept of linking two different functions to produce one outcome. After a bit of research, I found that breaking this particular sorting algorithm into two different methods made the program more simplistic; “StackOverflow” helped me understand the concept more. The partition method handles the sorting technique, while the quickSort method implements the output. The quickSort method alone cannot handle partitioning without being overly complex.

**selectionSort**() gets the vector of auction bids and loops through the elements from smallest to largest. To properly sort the bids, the function compares the value of the current bid to the next bid and swaps indexes accordingly. **quickSort**() gets the vector and calls for it to be sorted. If the lowest element is greater than or equal to the highest element, no sorting is necessary. To sort lowest to highest, I call the **partition()** function to handle the logic and recursively call **quickSort()**  to swap the indexes of elements. The lower element will be first, and the highest element will be displayed at the end of the vector.

**Pseudocode:**

**START**

**SET** functions and methods;

**INITIALIZE** common variables, libraries, and data types

**READ/STORE** CSV files (data sets)

**OUTPUT** console menu for user selection

**WHILE** user input is not equal to ‘9’

**GET** user input (1-4)

**IF** user input is ‘1’

**START** clock and store time in ticks

**CALL** loadBids() and store CSV data into a container/vector

**OUTPUT** number of records in the data set

**STOP** theclock and **OUTPUT**  the time in seconds it takes to read the data set

**IF** user input is ‘2’

**LOOP** through the data set and **DISPLAY** all bids

**CALL** displayBids()

**IF** user input is ‘3’

**START** the clock and **STORE** time in ticks

**CALL** selectionSort() function to sort bids from smallest to largest

**STOP** the clock and **OUTPUT** sorted bids

**OUTPUT** time needed to sort the vector in ticks and seconds

**IF** user input is ‘4’

**START** the clock and store in ticks

**CALL** quickSort() via the partition() function to **SORT** the auction data from smallest to largest

**STOP** the clock and **OUTPUT** sorted bids

**OUTPUT** time needed to sort the vector in ticks and seconds

**IF** user input is ‘9’

**EXIT** the program

**OUTPUT** ‘Good bye’

**END**