**2-3 Assignment: Vector Sorting**

Cristiano Miranda

Southern New Hampshire University

CS-300: Analysis and Design

Professor Muthukudage

14 September 2025

**2-3 Assignment: Vector Sorting**

Code Reflection:

            The Vector Sorting project's purpose is to load data from a file and to allow the user to sort the data based on a string. More specifically, bids from a CSV file are loaded by the program and stored in a vector. From here, the value associated with each bid's title property is used to sort bids within the vector. The program utilizes two different sorting methods, selection sort and quicksort, to highlight the difference in execution time between the two methods. The code shows that while there are multiple solutions to a problem, some are better than others, and in this case, the quicksort sorts the entire vector faster on average. Still, both methods employ different techniques. Selection sort compares the current index value to all other values in the array, finding the smallest value that is placed higher than it in the vector, and swaps it, continuing this pattern until the second-to-last array element is reached, upon which the algorithm finishes sorting the vector. For quick sorting, this method utilizes a pivot value that is then compared to multiple elements in the vector. If the element is less than the pivot value, it is moved to the left of the pivot, and if it is greater than the pivot, it is transferred to the right. Through this, once the pivot is compared against the other elements, its position is found since all elements to the left are less and all elements to the right are greater. Then, this pattern is repeated by splitting the vector into two partitions, one being a lower partition involving the elements before the pivot point and the pivot point index as well. The other is a higher partition involving the elements above the pivot point. The process continues until every element is utilized as a pivot point and moved to its proper position, since now the vector will be sorted as all elements within the vector are less than the values after it and greater than the values before it. While the quicksort method is faster, it is also more complicated to implement.

Some of the issues I ran into involved mispositioning specific lines of code and improperly incrementing values. In terms of mispositioning specific code, this issue arose when I accidentally placed the swap function outside of the if statement. Although this was a simple mistake, it resulted in a significant amount of time spent debugging. Similarly, for the improper incrementing problem, this also occurred when I accidentally applied the incorrect logic to the highIndex variable, which caused me to experience out-of-range errors since I incorrectly caused the program to add to the highIndex instead of subtracting. These problems showed me the importance of paying attention when using patterns, as making even a simple mistake while coding them can result in hard-to-notice and solve issues that could have been easily avoided had I been more attentive when implementing the code. Still, these problems helped reinforce my debugging skills, as I successfully found and solved them. In my case, I started by isolating the problem and identifying exactly where it was happening. Once I isolated the issue to the pivot function, debugging the code became easier, as I knew where to focus my attention. Finally, I took my time and walked through the code step by step, following my pseudocode process and the code pattern found online to discover where the error in my code occurred. By doing this, I was able to identify the issues and correct them, ensuring my program functioned correctly. Still, in other instances where the code is not a pattern, learning to walk through the code and the process I am trying to perform is a helpful and valuable skill to practice, allowing me to solve any problems I encounter more easily in the future.

Pseudo Code:

* displayBid(bid)
  + START
  + DISPLAY bid.bidId + “: ” + bid.title + “ | ” + bid.amount + “ | ” + bid.fund
  + RETURN
  + END
* getBid()
  + START
  + INIT bid
  + DISPLAY “Enter Id: “
  + GET bid.bidId
  + DISPLAY “Enter title: “
  + GET bid.title
  + DISPLAY “Enter fund: “
  + GET bid.fund
  + DISPLAY “Enter amount: “
  + SET bid.amount
  + RETURN bid
  + END
* loadBids(csvPath)
  + START
  + DISPLAY “Loading CSV file “
  + INIT bids: VECTOR
  + INIT file

// read file associated with file path

* + READ from file at csvPath and SET file to result
  + TRY
    - FOR index from 0 to number of rows in file
      * INIT bid

// Set all associated properties via their respective row and column location in the CSV file

* + - * SET bid.id = file[index][1]
      * SET bid.title = file[index][0]
      * SET bid.fund = file[index][8]
      * SET bid.amount = file[index][4]
      * APPEND bid to bids VECTOR
    - ENDFOR
  + CATCH (Error)
    - DISPLAY Error message
  + ENDTRY
  + RETURN bids VECTOR
  + END
* partition(bids, begin, end)
  + START
  + INIT lowIndex
  + INIT highIndex
  + SET lowIndex = begin
  + SET highIndex = end
  + INIT midPoint

// Calculate the middle position of the partition

* + SET midpoint = lowIndex + (highIndex - lowIndex) / 2
  + INIT done
  + SET done = FALSE
  + WHILE done IS NOT EQUAL TO TRUE
    - WHILE bids[lowIndex].title < pivotValue
      * INCREMENT lowIndex
    - ENDWHILE
    - WHILE bids[highIndex].title > pivotValue
      * DECREMENT highIndex
    - ENDWHILE
    - IF lowIndex IS GREATER THAN OR EQUAL TO highIndex THEN
      * SET done = TRUE
    - ELSE
      * SWAP elements at lowIndex and highIndex
      * INCREMENT lowIndex
      * DECREMENT highIndex
    - ENDIF
  + ENDWHILE
  + RETURN highIndex
  + END
* quickSort(bids, begin, end)
  + START
  + IF bids VECTOR size IS NOT EQUAL TO 0 THEN
    - INIT midIndex
    - SET midIndex = 0
    - IF begin IS GREATER THAN OR EQUAL TO end THEN
      * RETURN
    - ENDIF
    - SET midIndex = partition(bids, begin, end)

// Recursively call quickSort to sort the lower partition

* + - CALL quicksort(bids, begin, midIndex)

// Recursively call quickSort to sort the higher partition

* + - CALL quicksort(bids, midIndex + 1, end)
  + ELSE
    - DISPLAY “Load bids before sorting!”
  + ENDIF
  + END
* selectionSort(bids)
  + START
  + IF bids VECTOR size IS NOT EQUAL TO 0 THEN
    - INIT bidsSize
    - SET bidsSize = bids VECTOR size
    - FOR index from 0 to bidsSize - 1
      * INIT minIndex
      * SET minIndex = index
      * FOR nestedIndex from index + 1 to bidsSize
        + IF bids[nestedIndex].title IS LESS THAN bids[index].title THEN

SET minIndex = nestedIndex

* + - * + ENDIF
      * ENDFOR
      * SWAP elements at index and minIndex
    - ENDFOR
  + ELSE
    - DISPLAY “Load bids before sorting!”
  + ENDIF
  + END
* strToDouble(str, ch)
  + START
  + ITERATE through string and ERASE all instances of ch character variable and SET str to result
  + RETURN str
  + END
* main(argc, argv)
  + START
  + INIT csvPath
  + IF argc is EQUAL TO 2 THEN
    - Set csvPath = element 1 of argv VECTOR
  + ELSE
    - Set csvPath = “eBid\_Monthly\_sales.csv”
  + ENDIF
  + INIT vector bids
  + INIT ticks
  + INIT choice
  + SET choice = 0
  + WHILE choice IS NOT EQUAL TO 9
    - DISPLAY “MENU:”
    - DISPLAY “ 1. Load Bids”
    - DISPLAY “ 2. Display All Bids”
    - DISPLAY “ 3. Selection Sort All Bids”
    - DISPLAY “ 4. Quick Sort All Bids”
    - DISPLAY “ 9. Exit”
    - DISPLAY “ Enter choice: ”
    - GET choice
    - IF choice is EQUAL TO 1 THEN

// get the current clock time

* + - * SET ticks = clock()
      * SET bids = loadBids(csvPath)
      * DISPLAY bids VECTOR size + “ bids read”
      * SET ticks = clock() - ticks
      * DISPLAY “time: ” + ticks + “ clock ticks”

// convert the clock time to seconds using a predefined constant variable

* + - * DISPLAY “time: ” + ticks \* 1.0 / constant variable CLOCKS\_PER\_TICK + “ seconds”
    - ELSE IF choice is EQUAL TO 2 THEN
      * FOR index from 0 to bids VECTOR size
        + DISPLAY displayBids(bids[index])
      * ENDFOR
      * PRINT return line
    - ELSE IF choice is EQUAL TO 3 THEN

// get the current clock time

* + - * SET ticks = clock()
      * CALL selectionSort(bids)
      * DISPLAY “time: ” + ticks + “ clock ticks”
      * SET ticks = clock() - ticks

// convert the clock time to seconds using a predefined constant variable

* + - * DISPLAY “time: ” + ticks \* 1.0 / constant variable CLOCKS\_PER\_TICK + “ seconds”
    - ELSE IF choice is EQUAL TO 4 THEN

// get the current clock time

* + - * SET ticks = clock()

// Pass in the bids vector, its starting index of 0, and its end index

* + - * CALL quickSort(bids, 0, bids VECTOR size - 1)
      * DISPLAY “time: ” + ticks + “ clock ticks”
      * SET ticks = clock() - ticks

// convert the clock time to seconds using a predefined constant variable

* + - * DISPLAY “time: ” + ticks \* 1.0 / constant variable CLOCKS\_PER\_TICK + “ seconds”
    - ENDIF
  + ENDWHILE
  + DISPLAY “Good bye.”
  + RETURN
  + END