Southern
New Hampshire
University

Chris Trimmer

CS-300-T1159 DSA: Analysis and Design

Assignment 2-3: Vector Sorting Reflection

22EW1 – 09/10/2022

Southern
New Hampshire
University

The purpose of this document is to provide a code reflection of assignment 2-3. The code reflection describes the selection sort and quick sort functions I used for the assignment.

Code Reflection

The selection sort function takes a vector of bid objects as an argument. It proceeds to sort the vector of objects using selection sort algorithm based on the **title** of the bid. We use a nested for loop to execute the sort. The outer loop iterates over each object in the vector and sets the current object as the **min** index. The inner loop tests the **min** index against the remaining unsorted objects, setting any object that has a **title** less than the current **min** index title as the new min index. After the unsorted objects have been tested, it swaps the object set as the current min index with the current object under test from the inner loop. Effectively, the list of objects become sorted one at a time, as each object is tested against the other objects in the vector, and then swapped to the next position in the vector with sorted objects.

The quick sort function has two parts: the quick sort function itself, and a helper function to partition the vector. The partition function takes as arguments a vector of bid objects, the starting index, and the ending index. It determines the midpoint of the vector to be used as the pivot based the current starting and ending index, and divides it in half. It uses a nested while loop to determine the low and high index until the low index is greater than or equal to the high index. During each iteration, if the low index is not greater than or equal to the high index, then it performs a swap of the elements, and increments the low index and decrements the high index. It returns the high index to the quick sort function.

The quick sort function is a recursive function that takes as arguments the vector of bid objects, the starting index, and ending index. It first tests to ensure the size of the vector of bids is greater than zero. As a base case, if the begin index is greater than the end index, then the function is completed with sorting and returns to the caller. The function calls the partition function, passing a reference to the vector of bid objects, the begin index, and the end index. The resulting index that is returned from the partition function is the last element in the low partition. We then recursively call quick sort on the low partition using the begin index and the index returned from partitioning. We then call quick sort recursively on the high partition using the returned index + 1, and the end index. The recursive calls continue until the base case is met. The vector will be sorted after the recursive functions have completed.

I didn’t encounter any problems or issues when completing the exercise. I followed steps outlined in the assignment performing each of the tasks one at a time. I also tested and compiled my code after each step. I feel that catching bugs early in the coding process makes the remaining development quicker and easier. Note that I used the large .csv file that has 12000+ records for testing. The columns in that file are not in correct alignment, so I copied the file and create a new version that has the columns in correct alignment.

Southern New Hampshire University

Chris Trimmer

CS-300-T1159 DSA: Analysis and Design

Assignment 2-3: Vector Sorting Pseudocode

22EW1 – 09/10/2022

Southern
New Hampshire
University

The purpose of this document is to provide a code pseudocode of assignment 2-3. This includes the main functions used in the code. I will also provide screenshots of the results from my code executions.

Pseudocode

Menu Loop

The menu loop is contained within main. The loop enables the user to continue making menu selections until they exit the program by choosing option 9.

```
Get user input
  While user input is not equal to 9
      Display menu options
          Switch (user input)
              Case 1:
                  Start timer
                  Load the bids from the .csv file
                  Store results bids as objects in vector
                  End timer and calculate elapsed time
                  Display results
                  Break
              Case 2:
                  Loop through bids vector and display the results
                  Break
              Case 3:
                  Start timer
                  Call selection sort function and pass vector of bids
                  End timer and calculate elapsed time
                  Display results
                  Break
              Case 4:
                  Start timer
                  Call quick sort function and pass vector of bids
                  End timer and calculate elapsed time
                  Display results
                  Break
```

Selection Sort

```
Void selectionSort(vector<Bid>& bids) {
  Define variable min to be used as variable to store index of current minimum bid title
  Set variable result to the size of the vector of bid objects

  If result is less than 1
    then return, as there is nothing to sort

  // perform selection sort using nested for loop
  For (size_t position = 0; position < result; ++position) {
    Set min to position

    // use inner loop to compare current position to test against the unsorted objects
    For (size_t j = position + 1; j < result; ++j) {

      // use control to test current position is less than next unsorted object
      If current min.title is less than next bid.title
        Set min to next bid index

    } End inner loop

    Swap current position and min position

  } End outer loop

} End selection sort function
```

Quick Sort Partition Helper Function

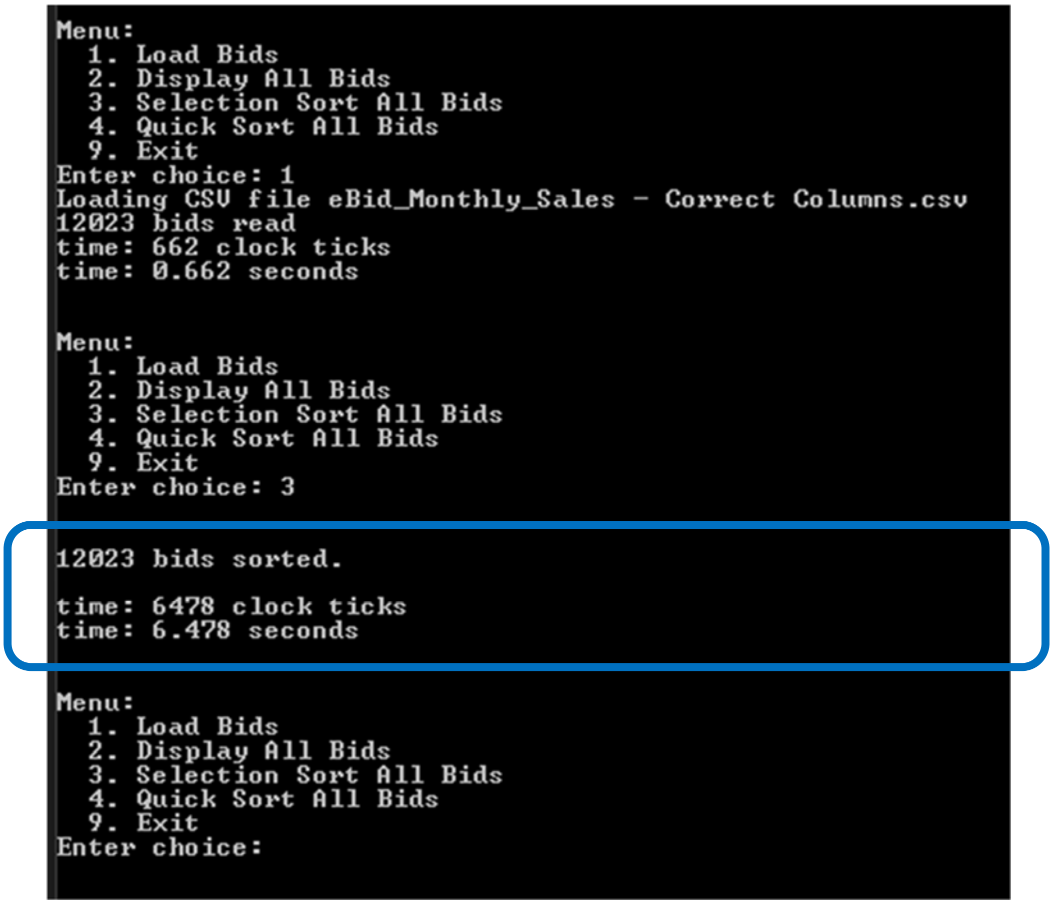
```
Void partition(vector<Bid>& bids, int begin, int end) {

  Set low equal to begin;
  Set high equal to end;

  Set midpoint = low + (high – low) / 2

  // create the pivot object at the midpoint
  Set pivot = bids.at(midpoint)

  Set control variable done = false
  While (not done) {

    While (bids.at(low).title is less than pivot.title) {
      Increment low by 1
    }

    While (pivot.title is less than bids.at(high).title) {
      Decrement high by 1
    }

    If (low is greater than or equal to high)
      then set done to true to exit while loop
    Else {
      Swap bids.at(low) and bids.at(high)

      Increment low
      Increment high
    }

  } end while loop

  Return the new high index back to quicksort function

} End partition function
```

Quick Sort

```
Void quicksort(vector<Bid>& bids, int begin, int end) {

  If bids.size() is less than or equal to 1, then return as the vector is empty

  // base case for recursive function
  If beginning index is greater than ending index, then return as there is no more to sort

  // get the partition index
  Int partitionIndex = partition(bids, begin, end)

  // recursively sort the low partition
  quicksort(bids, begin, partitionIndex)

  // recursively sort the high partition
  quicksort(bids, partitionIndex + 1, end)

} End quicksort
```

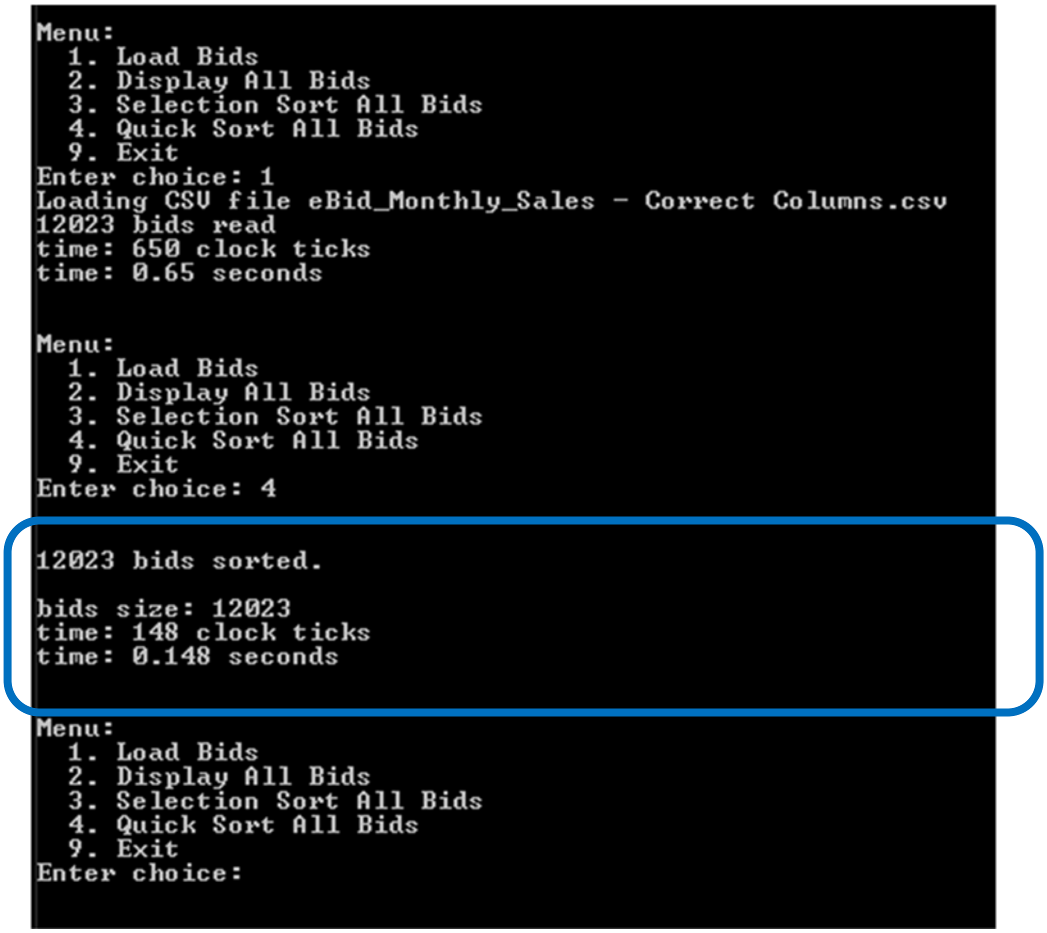
Screenshots

Selection Sort



```
Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice: 1
Loading CSV file eBid_Monthly_Sales - Correct Columns.csv
12023 bids read
time: 662 clock ticks
time: 0.662 seconds


Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice: 3


12023 bids sorted.

time: 6478 clock ticks
time: 6.478 seconds


Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice:
```

Quick Sort



```
Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice: 1
Loading CSV file eBid_Monthly_Sales - Correct Columns.csv
12023 bids read
time: 650 clock ticks
time: 0.65 seconds


Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice: 4


12023 bids sorted.

bids size: 12023
time: 148 clock ticks
time: 0.148 seconds


Menu:
  1. Load Bids
  2. Display All Bids
  3. Selection Sort All Bids
  4. Quick Sort All Bids
  9. Exit
Enter choice:
```