Austin Frey

Professor Ling

CS-300

November 6, 2022

Sorting Algorithms Assignment Reflection

The VectorSorting program is intended to organize a collection of bids on auction items. Developing the code did not seem like it was going to be difficult, especially because of how well designed the template for the assignment was. Implementing selection sort did not have any issues. However, quick sort developed a bug that is extremely difficult to track down due to the recursive nature of the algorithm. For some reason, quick sort does not completely sort the vector on the first call. I attempted to solve this problem multiple ways.

I coded the algorithm for quick sort presented in Zybooks nearly identically, and yet I continued to encounter the bug. I tried various comparison methods since sometimes comparing strings with relational operators can cause weird bugs, but that did not help either. I tried changing the indexes to see if maybe there was a one-off error I was not spotting, but alas, the program still failed to complete the sort on the first call. A secondary call to quick sort completes the process. Since quick sort has such a fast completion time compared to selection sort, I determined that calling the function twice was negligible in terms of time and memory usage. While calling the function twice does introduce redundancy, I am past my deadline and needed to get something submitted. The program functions as it is expected to, even if a look “under the hood” shows a little bit of a “hacked” solution.

**Pseudocode / Flowchart**

Selection Sort:

“procedure selection sort (bids)

bids : array of items

n : size of list

for i = 1 to n - 1

/\* set current element as minimum\*/

min = i

/\* check the element to be minimum \*/

for j = i+1 to n

if bids[j] < bids[min] then

min = j;

end if

end for

/\* swap the minimum element with the current element\*/

if indexMin != i then

swap bids[min] and bids[i]

end if

end for

end procedure” (Data Structure and Algorithms Selection Sort, n.d.)

Quick Sort:

procedure quickSort(bids, begin, end)

mid: middle index

if right-left <= 0

return

else

pivot = bids[mid]

partition = partitionFunc(bids, left, right)

quickSort(bids, left, lastLowIndex)

quickSort(bids, lastLowIndex+1, right)

end if

end procedure” (Data Structure and Algorithms Selection Sort, n.d)

Quick Sort Partition:

function partitionFunc(bids, left, right)

leftPointer = left

rightPointer = right - 1

pivot = bids[left + (left – right) / 2]

while True do

while bids[++leftPointer] < pivot do

//do-nothing

end while

while rightPointer > 0 && bids[--rightPointer] > pivot do

//do-nothing

end while

if leftPointer >= rightPointer

break

else

swap leftPointer,rightPointer

end if

end while

swap leftPointer,right

return leftPointer

end function (Data Structure and Algorithms Selection Sort, n.d)

References

*Data Structure and Algorithms Selection Sort*. (n.d.). https://www.tutorialspoint.com/data\_structures\_algorithms/selection\_sort\_algorithm.htm