**Pseudocode**

**OPEN** program

**SET** integer *choice* = 0

**CREATE** vector *bids* to BID objects

**CREATE** time\_t *ticks*

**WHILE** *choice* not equal to 9

**DISPLAY** menu

**INPUT** *choice*

**IF***choice* equals 1

**SET** *ticks* to system time

**SET** *bids* equal to the loadbid function, with the csvPath argument

**OUTPUT** *bids* vector size + “bids read”

**SET** *ticks* equal to current system time – *ticks*

**OUTPUT** “time: “ + *ticks* + “ clock ticks”

**OUTPUT** “time: “ + *ticks* \* 1.0 / CLOCKS\_PER\_SEC + “ seconds”

**ELSE IF** choice equals 2

**LOOP** through *bids*

**CALL** displayBid function giving *bids* at the current index as the argument

**ELSE IF** choice equals 3

**SET** *ticks* to current system time

**CALL** selectionSort function, sending *bids* as the argument

**SET** *ticks* to current system time –*ticks*

**OUTPUT** “time: “ + *ticks* + “clock ticks”

**OUTPUT** “time: “ + (*ticks* \* 1.0 / CLOCKS\_PER\_SEC) + “ seconds”

**ELSE IF** *choice* equals 4

**SET** *ticks to* current system time

**CALL** quickSort function, sending *bids*, 0, (size of *bids* – 1) as the arguments

**SET** *ticks* to current system time – current ticks value

**OUTPUT** “time: “ + *ticks* + “clock ticks”

**OUTPUT** “time: “ + (*ticks* \* 1.0 / CLOCKS\_PER\_SEC) + “ seconds”

**ELSE IF** *choice* equals 9

**EXIT** loop

**ELSE**

**CONTINUE** loop

**OUTPUT** “Goodbye”

**END** program

**Reflection**

Looking at the different algorithms of sorting, I see each kind has its own advantages. Each one has its own time and place to be used. Some will be used purely for speed, while others will be used for accuracy. The speed and accuracy are measured based on the complexity of the algorithm. There are also some notations that are used to help describe how an algorithm may behave. One example of this is the Big-O notation which shows the upper bound growth rate of the algorithm.

When looking for accuracy I find the algorithm for selection sort to be a safe option. It compares each item with the others of a vector or an array without repeating the same comparison. It essentially splits the sorted and unsorted items to avoid redundancy but still has a lot of comparisons. Due to the amount of comparisons the runtime is considered to be O(). If speed is the goal, then quick sort is a good option, but it is not the most accurate of options. This sort will group elements together and then sort just what is in the small groups. Since not everything is being compared and moved around this algorithm is rather quick for large lists. This algorithm is known to have a runtime of O(N log N).

Vahid, F. (2019). CS300: Data Structure and Algorithms. ZyBooks. Retrieved July 2, 2023 from <https://learn.zybooks.com/zybook/CS-300-X6110-OL-TRAD-UG.23EW6>