The purpose of sorting algorithms is to, in basic terms, sort lists. There’s many different ways to create a sorting algorithm, but in this example, only a quick sort and selective sort is used. A selective sort compares one value to the next value and it repeats this process until it is complete. While effective, it does take quite a long time to sort a list. Quick sort breaks the list into halves using a pivot value, then breaks those halves into others using another pivot value. This way is more complex, but a lot faster than the selective sort.

There wasn’t many issues while creating the code, except for one. I was able to get everything working aside from the last swap in the partitioning of the quick sort. The quick sort would sort everything until the very last element, then it would give a stack overflow error. I looked through the textbook many times and researched through many articles only to comb through the code to find that I had put the wrong int variable in the quicksort function. The function was calling itself with the whole vector as the range plus the high end of the vector. So, it was effectively creating an infinite loop. After switching to the correct variable, the program was able to do its job correctly.

**Fixme 1 – Selection Sort:**

PARAMETERS: bids

DEFINE Int minimum

FOR each bid i in bids {

SET minimum to bid index

FOR each bid j in bids {

IF compare bid j to minimum

SET minimum to j

IF minimum is NOT equal to i

SWAP bid i and bid minimum

**Fixme 2a – Quick Sort:**

PARAMETERS: bids, begin, end

DECLARE Int middle

SET middle to zero

IF begin is greater or equal to end

RETURN

SET middle to CALL partition function with parameters: bids, begin, end

CALL quicksort with parameters: bids, begin, middle

CALL quicksort with parameters: bids, middle+1, end

**Fixme 2b – Quick Sort Partition:**

PARAMETERS: bids, begin, end

DECLARE Int lowest, highest, mid

SET lowest to begin

SET Highest to end

SET mid to: begin + (end – begin) divided by 2

DECLARE String pivot

SET pivot to middle bid’s title

DECLARE bool finished

SET finished to false

WHILE finished is not true {

WHILE bids lowest title is less than pivot {

INCREMENT lowest

}

WHILE pivot is less then bids highest title {

DEINCREMENT highest

}

IF lowest is great than or equal to highest

SET finished to true

ELSE

SWAP lowest and highest bids

INCREMENT lowest

DEINCREMENT highest

RETURN highest