**Module 2 Reflection and Pseudocode**

Jermaine Wiggins

Southern New Hampshire University

CS300 Analysis and Design

Prof. Sathish Gopalakrishnan

Jan 19, 2025

**Module 2 Reflection and Pseudocode**

Before starting the ZyBooks assignment, I glanced at the instructions, and it seemed very challenging. However, after completing the ZyBooks material, this assignment turned out to be the perfect way to reinforce what I learned.

The purpose of this program is to sort a collection of bids stored in a vector. The first fix-me was a function called partition, which is later used in the quicksort function. This function works by searching for an element greater than the pivot at the beginning and an element smaller than the pivot at the end. When it finds such elements, it swaps their positions, helping to partially organize the list before recursion takes over.

**Partition Pseudocode**

**SET** low to beginning

**SET** high to end

**CALCULATE** mid point and set to pivot

**WHILE NOT Done**

**WHILE** bids low < pivot

**INCREMENT** low

**WHILE** bids high > pivot

**INCREMENT** high

**IF** low > high

Done = **TRUE**

**ELSE**

**SWAP** low and high

**INCREMENT** high and low

**RETURN** high

The quick sort function calls the partition function to help sort the higher and lower partitions of the vector by moving all elements less than the pivot to the left and all elements greater than the pivot to the right. Then the quicksort function calls itself twice to sort both partitions.

**Quicksort Pseudocode**

**SET** mid to 0

**IF** begin >= end

**RETURN**

**SET** mid to partition function

**CALL** quicksort function beginning to mid

**CALL** quicksort function mid to end

The selection sort function finds the smallest element in the unsorted part of the list and moves it to its correct position. It compares each element with the remaining elements to find the smallest one. If a smaller element is found, it remembers its position and swaps it with the current element at the end of the pass. This process repeats until the entire list is sorted.

**Selection Sort Pseudocode**

**SET** min to current index

**SET** bidsize to vector size

**SET** pos to 0

**FOR** pos from 0 to bidsize – 1

**SET** min to pos

**FOR** j from pos + 1 to bidsize + 1

**IF** bidsize title at index j < bidsize title at index min

**SET** min to j

**IF** min doesn’t equal pos

**SWAP** bids at index pos with bids at index min

The last part of the assignment involved invoking the sorting functions within a switch-case block. When the user selects option 3, the program calls the selection sort function, and when the user selects option 4, it calls the quicksort function. Each case also includes a timer to measure and display the sorting speed. As expected, the quicksort function was more efficient than selection sort.

**Case 3 pseudocode**

**CHECK** if bids loaded

**IF** not load bids

**START** timer

**CALL** selection sort function

**PRINT** total bids sorted

**PRINT** total time elapsed

**Case 4 pseudocode**

**CHECK** if bids loaded

**IF** not load bids

**START** timer

**CALL** quick sort function

**PRINT** total bids sorted

**PRINT** total time elapsed