**Module Two – Vector Sorting Assignment Reflection**

The purpose of the code is to implement and compare Selection Sort and QuickSort algorithms for sorting a collection of bids by their bid.title. Selection Sort works by repeatedly finding the smallest title in the unsorted portion and swapping it into the last element of the sorted portion. The complexity for selection sort is O(n^2). QuickSort, on the other hand, uses a divide-and-conquer approach, partitioning the vector around a pivot and recursively sorting the subarrays. Its time complexity is O(nlogn) and it’s significantly faster especially when there is a large dataset.

The main challenge for me was implementing the partitioning logic correctly. Ensuring the pivot was chosen appropriately and that elements were correctly divided into "less than" and "greater than" partitions required careful thinking. Additionally, managing recursion posed a challenge, as improper base cases or recursive calls could cause stack overflows. These issues were resolved by reading zybook example carefully, following the project walkthrough video, and ensuring the base case (begin >= end) was correctly implemented.

**Pseudocode for selectionSort:**

//define min as int (index of the current minimum bid)

// check size of bids vector

// set size\_t platform-neutral result equal to bid.size()

// pos is the position within bids that divides sorted/unsorted

// for size\_t pos = 0 and less than size -1

// set min = pos

// loop over remaining elements to the right of position

// if this element's title is less than minimum title

// this element becomes the minimum

// swap the current minimum with smaller one found

// swap is a built in vector method

**Pseudocode for partition:**

//set low and high equal to begin and end

// Calculate the middle element as middlePoint (int)

// Set Pivot as middlePoint element title to compare (string)

// while not done

// keep incrementing low index while bids[low].title < Pivot

// keep decrementing high index while Pivot < bids[high].title

/\* If there are zero or one elements remaining,

all bids are partitioned. Return high \*/

// else swap the low and high bids (built in vector method)

// move low and high closer ++low, --high

//return high;

**Pseudocode for quicksort:**

//set mid equal to 0

/\* Base case: If there are 1 or zero bids to sort,

partition is already sorted otherwise if begin is greater

than or equal to end then return\*/

/\* Partition bids into low and high such that

midpoint is location of last element in low \*/

// recursively sort low partition (begin to mid)

// recursively sort high partition (mid+1 to end)

**Pseudocode for running case 3 (selection sort):**

// Initialize a timer variable before sorting bids

// Call selectionSort to sort the bids

// Calculate elapsed time and display result

**Pseudocode for running case 4 (quick sort):**

// Initialize a timer variable before sorting bids

// Call quickSort to sort the bids

// Calculate elapsed time and display result