App pseudocode

1. Identify the misplaced selectionSort and quickSort function calls.
2. Determine the correct case blocks where these sorting functions should be executed.
   1. For example, if selectionSort is intended to be executed for a specific sorting option, it should be placed within the corresponding case block.
3. Similarly, move the quickSort(bids, 0, bids.size() - 1); call into its appropriate case block.
4. If quickSort is intended for another specific option, such as option 3, create a new case 3: block and place the quickSort call there.
5. Ensure that each sorting function call is followed by the time measurement and display logic, as well as the loop to display the sorted bids, if this is the desired behavior after sorting.
6. After relocating the sorting function calls, verify that the switch statement is correctly structured with break statements at the end of each case block to prevent fall-through (unless intentional).
7. Test the program to ensure that each sorting option correctly sorts the bids and displays the sorted list along with the sorting time.
8. If any sorting option is meant to display the bids without sorting, ensure that its case block only contains the loop for displaying the bids

Selection sort pseudocode

1. Initialize minIndex: A variable to keep track of the index of the minimum element during each iteration of the sort.
2. Determine the size of the bids vector: The variable bid\_size holds the total number of Bid objects in the vector.
3. Outer loop - Dividing the vector into sorted and unsorted parts:
4. Inner loop - Finding the minimum element in the unsorted part:
5. Swap the found minimum element with the element at the current position pos:
6. After the inner loop, if minIndex is not equal to pos (meaning a smaller element was
7. Repeat until the entire vector is sorted

Quick sort pseudocode

1. Define the function quickSort that takes a vector of Bid objects and two integers, begin and end, as parameters.
2. Initialize the integer mid to 0.
3. Check the base case: if begin is greater than or equal to end, return from the function.
4. Partition the vector into two parts using the partition function, and assign the return value to mid.
5. Recursively call quickSort on the first part of the vector, from begin to mid.
6. Recursively call quickSort on the second part of the vector, from mid + 1 to end.