```
laurenschmidt@esihp5 csci1112_sp22_hw03 % javac ProfileSorts.java
laurenschmidt@esihp5 csci1112_sp22_hw03 % java ProfileSorts
100 Classes
profileSelection::allocs:3, compares:4950, swaps:99
profileBubble::allocs:3, compares:4950, swaps:2343
profileInsertion::allocs:3, compares:2343, swaps:2343
profileQuicksort::allocs:554, compares:783, swaps:286

1000 Classes
profileSelection::allocs:3, compares:499500, swaps:999
profileBubble::allocs:3, compares:499500, swaps:248567
profileInsertion::allocs:3, compares:248567, swaps:248567
profileQuicksort::allocs:10567, compares:13984, swaps:7827

10000 Classes
profileSelection::allocs:3, compares:49995000, swaps:9999
profileBubble::allocs:3, compares:49995000, swaps:24929215
profileInsertion::allocs:3, compares:24929215, swaps:24929215
profileQuicksort::allocs:111036, compares:201583, swaps:75984
```

Selection Sort

The selectionSort algorithm has three allocations because it allocates space for the String[] largest, the integer pos (position), and the integer j that is in the for loop. These are three variables that require three allocations of space in memory. The comparisons execute whenever the computer checks the elements in the array to see if one is less than the other. Finally, the swaps occur whenever a value is actually less than the value directly to its right. The selectionSort algorithm has the least amount of swaps, but in terms of allocations and comparisons, it is similar to the bubbleSort, and insertionSort methods. It is also relatively slow because it checks every element in the array one at a time.

Bubble Sort

Similarly to the selectionSort algorithm, the bubbleSort algorithm has three allocations and similar swaps and comparisons. It takes up about the same amount of space in memory as the insertionSort and selectionSort methods, but is the slowest and most time consuming, with its swaps far exceeding that of the selectionSort methods.

Insertion Sort

insertionSort takes up a similar amount of memory, and has less comparisons than bubbleSort and selectionSort, but the same amount of swaps as bubbleSort. It is just as slow as selectionSort and bubbleSort, but still takes up the same amount of space in memory.

Quick Sort

quickSort takes the least amount of time to compute, making it optimal for large datasets. However, because it uses recursive methods, it takes up large amounts of memory, with its allocations far exceeding that of insertionSort, bubbleSort, and selectionSort. Because of this, it

is important for the user to be aware of the space in memory that is being used so as not to cause a stack overflow error in which the memory is used up completely.

Conclusion and Comparisons

InsertionSort, bubbleSort, and selectionSort are slower methods, but they take up less space in memory than quickSort. quickSort is the quickest method, but because it uses recursive methods (methods that call themselves) it takes up large amounts of memory, so a stack overflow error is a concern when implementing the quickSort algorithm. The sorting algorithm that should be used depends heavily on the size of the dataset, the memory of the machine that the program is being ran on, and the speed at which the data needs to be sorted.