Complexity Timing Experiment

By: Kevin Nolan – CSE 2300

Introduction

Complexity Analysis is a way of determining how fast or slow an algorithm may take, in the case of arrays we can use it to help determine how efficient the sorting algorithm is. In order to show real time speed of these sorting algorithms I will use a java program with the Bubble Sort and Select Sort methods and record the times to show the speed they operate on my PC.

System Used

Processor:	AMD Ryzen 5 – 3.4 ghz
Memory:	16gb RAM
Operating System:	Windows 10, 64 bit operating system.
IDE Used:	Eclipse IDE 4.12

Process

I wrote 2 different programs that prompted for an array size, and then iterate a loop 1000 times to create an array of random integers, recorded the start time in milliseconds, sorted the array, and then recorded the end time. For this project I used arrays of size 500, 2500 and 5000.

- **Bubble Sort:** In general this tends to be considered a slow sorting algorithm as it has to go 1 by 1 through each element n, repeating this process n-1..n-2..etc. iterations and thus giving it a O(n^2) complexity. It can be fast though and have an O(n) complexity if the array is already sorted. I like how simple this sorting algorithm is and how easy it is to trace through, but I knew when I executed my code that it would take some time to process the larger sized arrays. The size 500 array executed near instantly, size 2500 took a bit more at near 7 seconds and then the longest wait by far was the 5000 size array clocking in at near 30 milliseconds. Ideally this sort algorithm would be fine to use with smaller size arrays, but not great once you start getting near 1000.
- **Select Sort:** This sorting algorithm is faster compared to bubble sort, but does tend to be a bit more complex to understand as you have to keep track of the subarrays and a minimum value. It also requires 2 loops with 2 different parts, one part of the loop finds the minimum value while the other swaps the values which gives it a complexity of O(n^2), due to this it will never have a best case of O(n) like Bubble Sort does. It will also never have a worst case of more than O(n^2) as well. After executing the program I could immediately tell differences in terms of processing speed as the size 500 array executed a little faster than bubble sort by a few .01 seconds, but the noticeable difference was the size 2500 array at only 1.4 milliseconds and then the size 5000 array at only a little over 5 milliseconds, a near 24 millisecond difference from bubble sort!

Conclusion

Here are my concluding points:

- Bubble Sort and Selection Sort both have similar complexities (O(n^2)) but as the size of the input increases then Bubble Sort becomes much slower to process.
- Bubble Sort complexity requires it to go from n (starting value) to the max value in terms of comparison amount whereas Selection Sort goes from n -1 x n-2 divided by 2 comparisons.
- Selection Sort essentially makes the array smaller for each iteration so thus it has less values to compare to whereas Bubble Sort always goes through the same amount on each iteration.
- Bubble Sort would be efficient for an array with very few values or a near sorted array. It would not require as many iterations as Select Sort in this instance.
- In terms of programming code speed (times were verified with stopwatch):
    - Size 500 Array: The time to wait was not too noticeable for either sort method in terms of speed.
    - Size 2500 Array: The Bubble Sort program gave a bit more of a pause in processing speed whereas the Select Sort program processed faster and was not as noticeable in wait time.
    - Size 5000 Array: The Bubble Sort program was prominent in wait time for this array size as it took near 30 milliseconds to complete. The Select sort program on the other hand only took near 6 milliseconds of processing time, a vast difference between the two.