Sean Brandon
spbrando@ucsc.edu
11/12/20

CSE13s Fall 2020
Assignment 5: Sorting
Writeup Document

In this assignment I was tasked with implementing four different types of sorting algorithms and gathering statistics on each algorithm's performance such as number of comparisons required and swaps required to sort an array of a certain length. The four different types of sorting algorithms implemented are bubble sort, shell sort, quick sort, and binary insertion sort. Testing these sorting functions with a test harness, I was able to gather the necessary statistics on how they all compare to each other. Below is a description of the time complexity for each sort, what I have learned, and how I experimented with the various sorts.

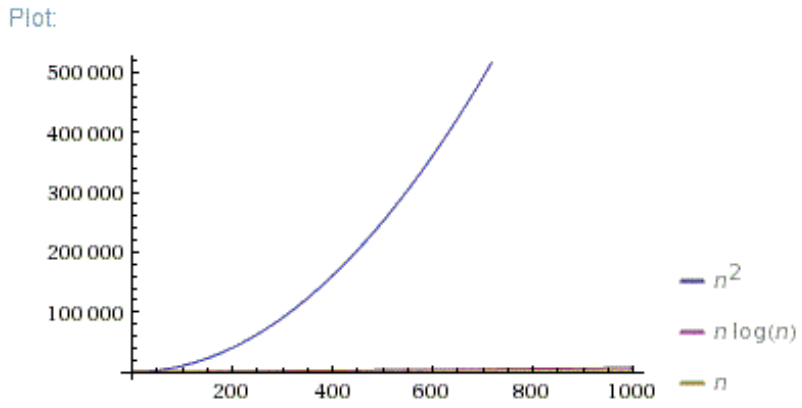
Bubble:
The time complexity of bubble sort is as follows: Best case = O(n), average case = $O(n^2)$, worst case = $O(n^2)$. The fact that the best case of this algorithm is O(n) means that it is able to quickly detect if an array is already sorted and this feature is its advantage over its competitors. However, the fact that its average and worst cases are $O(n^2)$ means that as the length of the unsorted array grows linearly, the time complexity grows exponentially which is much worse than its competitors.

Shell:
The time complexity of shell sort is as follows: Best case = n, average case = depends on gap size but is around O(nlogn), worst case = also depends greatly on gap sequence. Different gap sequences give different results. However, it is known that the worst case is always less than or equal to $O(n^2)$. In our pseudocode we used n * 5/11 but we could also have used a sequence such as n * ½. Deciding which gap sequence to use is a difficult question to answer because too few gaps slows down the speed at which the values are swapped but too many gaps produces an overhead.

Quick:
The time complexity of quick sort is as follows: Best case = O(nlogn), average case = O(nlogn), worst case = $O(n^2)$. While its worst case of $O(n^2)$ does seem troublesome, this can mostly be avoided by choosing a partition correctly. Quick sort is a very popular sorting algorithm because of how reliably it can produce results in nlog(n) time which is very quick (hence the name).

Binary Insertion:
Time complexity for binary insertion sort is as follows: Best case = O(nlogn), average case $O(n^2)$, worst case $O(n^2)$. The speed of binary insertion sort is greatly improved by the binary search algorithm. Without it, it would simply be an insertion sort but the binary search algorithm allows us to find the correct placement of an element with significantly less comparisons.

Below shows a graph of time complexities in relation to n, showing visually just how much of a difference there is between n^2 and nlog(n).



I took this graph (and only this graph) from this site: https://www.google.com/url?sa=i&url=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F23329234%2Fwhich-is-better-on-log-n-or-on2&psig=AOvVaw39v_9TClk7P2wCfhGTIDn8&ust=1605524501798000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCMiWwP-yhO0CFQAAAAAdAAAAABAP

To see the differences between time complexity of each sorting algorithm I did a few things:

- Had each algorithm sort an array of 1,000,000 randomly arranged elements to see the average case differences
- Had each algorithm “sort” an already sorted array of 1,000,000 elements to see the differences in time for best cases
- Had each algorithm sort a reverse order sorted array of 1,000,000 elements to see the difference between each algorithm’s worst case.