John Kennedy
Programming Data Structures (H)
Mr. Ritter
4 December 2015

On my honor, I have not given or received any unauthorized aid on this work.

A Look into Various Sorts and Searches

For each algorithm, I wrote a perl script to test the algorithm on various size text files, from 500 to 200,000 words. These perl scripts saved the algorithm output (swaps, comparisons, and time) to a CSV file, which was then parsed and analyzed using Wolfram Mathematica. All data files: Mathematica notebooks (and PDF conversions), Java files, graphs (PDFs) and Perl files, are available in a git repository - github.com/jo-hnkennedy/assignment2.

The key relationship is between word count, or elements to be sorted, and time. To measure time, I used the System.nanoTime() call in Java. However, there have been issues raised by various blog posts (here and here), that make the function potentially untrustworthy. To account for this, I documented the variance of each function's time below, where I ran each function on the same dataset (10000.txt, or 10,000 words), and measured the standard deviation for each function.

Sorting Algorithm	**Standard Deviation**
quick	15.3839392770165
merge	10.7858169607729
bubble	640.918800369421
insertion	68.6107095178016
selection	201.013203706553

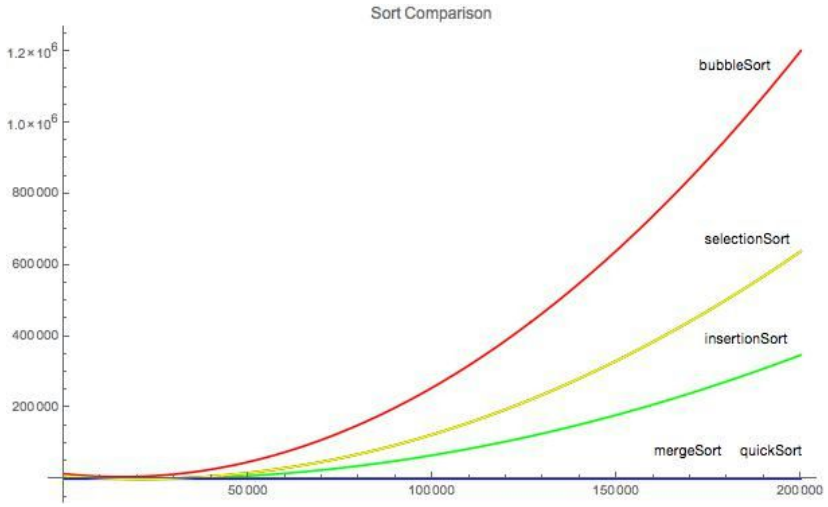
Algorithm	Average Case	Approximated Time/Word function [f(x)]
selection sort	$O(n^2)$	$8728 - 0.84 * x + 0.00002x^2$
quicksort	O(n logn)	$15 + 0.00167x - 2.92 * 10^{-9} x^2$
insertion sort	$O(n^2)$	$4910 - 0.489x + 0.00001x^2$
merge sort	O(n logn)	$12.8 + 0.0008x + 1.205 * 10^{-10}x^2$
bubble sort	$O(n^2)$	$14164 - 1.1004x + 0.0000352x^2$
linear search	O(n)	$0.124765 + 1.876 * 10^{-6}x - 6.104 * 10^{-12}x^2$
binary search	O(log n)	$0.136116 + 1.22210^{-6}x - 3.405 * 10^{-12}x^2$
quadratic	O(log n)	$0.180885 + 0.0002x - 2.43924 * 10^{-10}x^2$

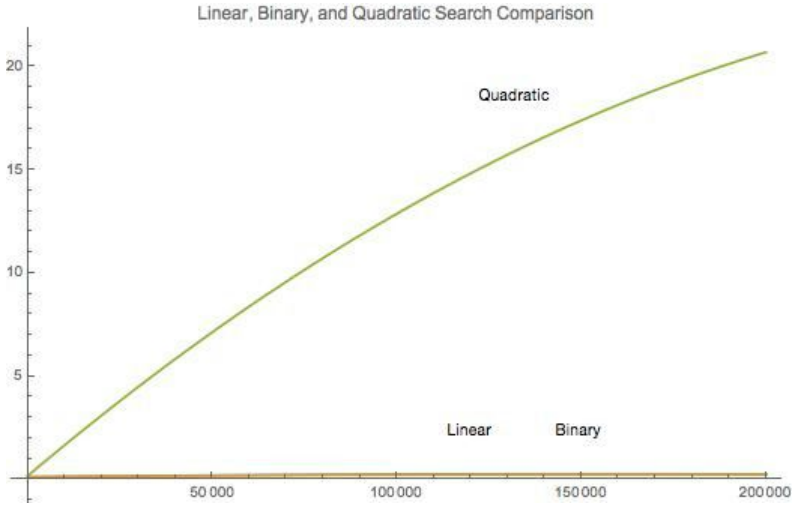
Sorting Test Case: sorted a 50,000 word text file (link).

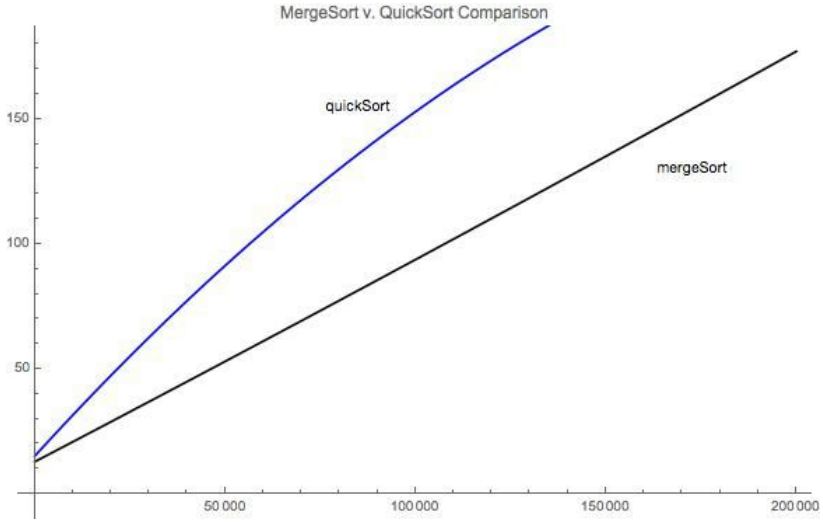
Algorithm	Comparisons	Swaps	Test Time (milliseconds)
selection sort	1,249,975,000	50,000	32,086.7715
quicksort	1,078,086	182,772	129.9129
insertion sort	623,612,400	0	14,872.2482
merge sort	718,243	0	105.0532
bubble sort	1,249,975,000	420,558,487	74,341.5924

Searching Test case: sorted a 200,000 word text file (link) for the word “cowboy”

Algorithm	Comparisons	Test Time (milliseconds)
linear search	0	11.963264
binary search	3,914,910	2.712255
quadratic search	172484	21.733854


Sort Comparison
1.2×10^6
1.0×10^6
800000
600000
400000
200000
bubbleSort
selectionSort
insertionSort
mergeSort
quickSort
50000
100000
150000
200000


Linear, Binary, and Quadratic Search Comparison
20
15
10
5
Quadratic
Linear
Binary
50000
100000
150000
200000


MergeSort v. QuickSort Comparison
150
100
50
quickSort
mergeSort
50000
100000
150000
200000

The graphs demonstrate both the general strengths of the algorithms as well as the flaws of my setup. To truly illustrate the power of binary sort over linear sort, I would have had to search for a word amongst millions of elements. As I only went to 200,000 words and chose a random word that ended up being relatively early in the list, linear search overperforms in my tests. A better test would have been to choose a word not in the list to get a legitimate worst case runtime. My graphs also provide a quick visual guide as to the uses of each algorithm. For example, with an expected element size of about 150,000, one should implement either merge or quicksort from the information in “Sort Comparison”.

Final note: What is in the Github?

The git repository contains full analysis sources (in Mathematica notebooks/pdfs), as well as Time v. Word Count graphs for each algorithm. The graphs also contain the testing file I used, the Perl files I used to batch test, and the CSV files that the Perl scripts outputted to. All of this information is more detailed in the README file.