Shoshana Abrass
abrasss@onid.oregonstate.edu
CS162, Lab #7
Feb 22, 2015

CS162 Lab 7

Last week in Lab6 I implemented Selection Sort. This week I added BubbleSort to the menu; the bubble sort code is very simple so it didn't take long, but I did refer back to a sample program I wrote last summer.

Measurements

The clock() function didn't have enough granularity to measure the difference in fast algorithms like linear search. These measurements use the chrono::high_resolution_clock which reports in microseconds of CPU time.

Conclusions

Linear search, as expected, is O(n): the cpu consumed scales linearly with the number of comparisons performed.

Selection sort and Bubble sort are O(n^2), as described in the book. Selection sort is more efficient; it performs $(n - 1) + (n - 2) + \ldots + 2 + 1 = n(n - 1) / 2$ comparisons. Bubble sort, the way I've implemented it, is less efficient. It can be optimized by skipping the >n elements on the nth pass. Without the optimization the worst-case performance is n^2. I would expect Bubble sort to take twice as long as selection sort on the same random data; in fact the tests show it takes 2.6 times as long. [1]

Binary search performs exactly as predicted. In these tests I'm always measuring the worst case: 0, the target of the search, is at the very beginning of the (sorted) file - or missing. Binary search for missing target will take$[\log_2(N)+1]$ steps, and this is exactly what I see in the program output.

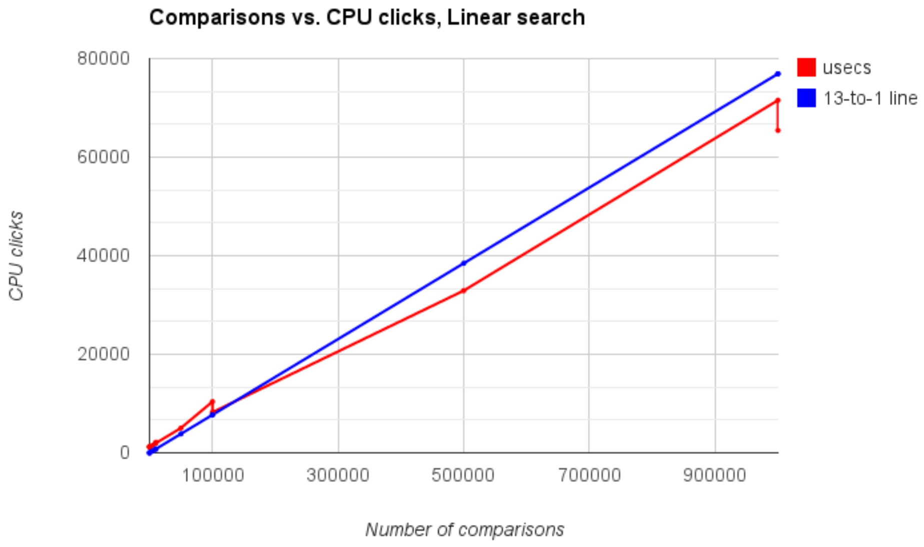
Data: Linear search

	target at beginning		target in the middle		target at end		target not there	
search	steps	usecs	steps	usecs	steps	usecs	steps	usecs
Linear,10k	1	1282	5001	1529	10,000	2126	10,000	1958
Linear,100k	1	1122	50001	5039	100,000	10,374	100,000	8,212
Linear,1M	1	1221	500,001	32,919	1,000,000	71,551	1,000,000	65,429

[1] For a very small data set (1000 numbers) there's no measurable difference between the two sorts. My guess is that the time taken by the algorithm is trivial compared to the time it takes to open, read, and write the files -- so the difference is invisible at this scale.

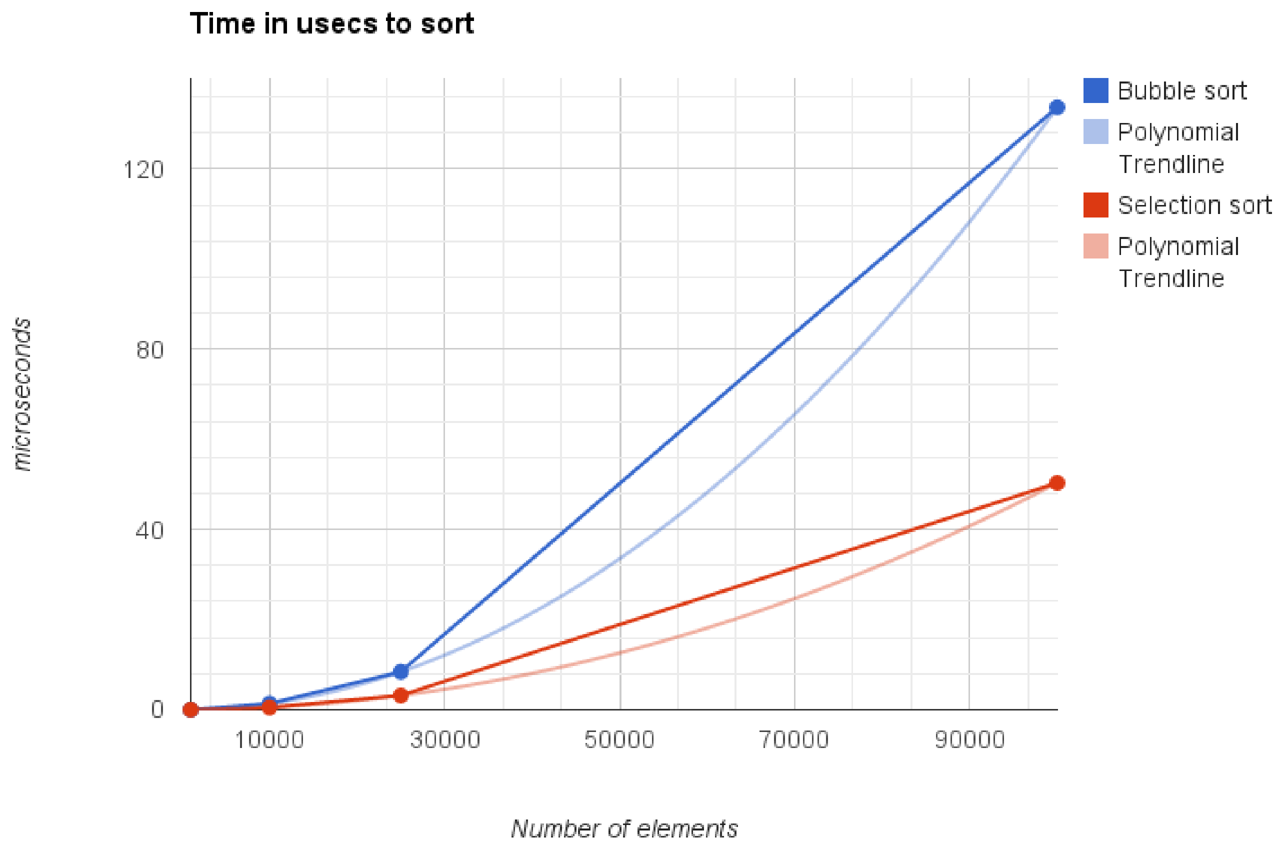
Arranging this table a little differently, we get:

Comparison steps	msecs
1	1282
1	1122
1	1221
5001	1529
10,000	2126
10,000	1958
50001	5039
100,000	10,374
100,000	8,212
500,001	32,919
1,000,000	71,551
1,000,000	65,429



Data: Bubble and Selection sort

Sort	size of file	File 1	File2	File 3	File 4	avg time in seconds
Selection	1000	0.018	0.036	0.025	0.024	0.026
Bubble	1000	0.026	0.021	0.018	0.030	0.024
Selection	10000	0.525	0.523	0.525	0.519	0.523
Bubble	10000	1.349	1.474	1.349	1.362	1.384
Selection	25000	3.178	3.174	3.188	3.186	3.181
Bubble	25000	8.252	8.999	8.226	8.258	8.434
Selection	100000	50.530	50.380	50.230	50.290	50.3575



Data: Binary search

Search	elements	log 2(N)	milliseconds				average
Binary, 1k	1000	10	896	698	695	733	756
Binary, 10k	10000	14	1897	1688	1601	1629	1704
Binary, 50k	50000	16	9828	8583	8477	8486	8844
Binary, 100k	100000	17	17667	12661	10458	10479	12816
Binary, 270k	270000	19	48960	50616	48989	29169	44434
Binary, 1M	1000000	20	91246	87144	87954	87636	88495