Blaise Johnson

Isaac Griffith

CS 3308

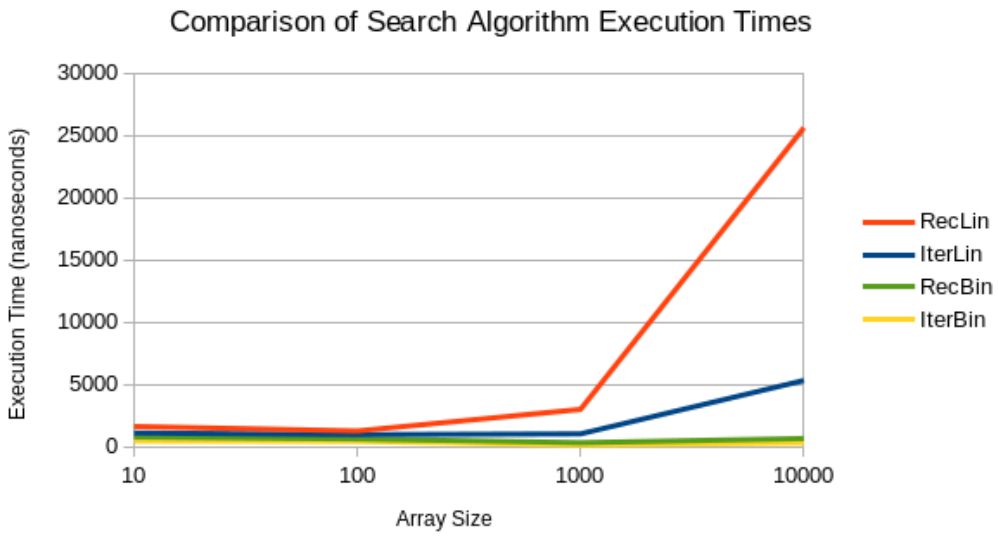
15 February, 2019

An Analysis of Four Sort Algorithms in Java

An experimental analysis was performed on the iterative and recursive implementations of both linear search and binary search. In general, linear search is in O(n), and binary search is in O(log n). Experimental data seems to back up this claim.

A randomly generated array of a certain length and a randomly generated search term which had some chance of being in the array were both created. These were then passed into the algorithm and the execution time was recorded in nanoseconds using the nanoTime function in Java's "time" library. This process was repeated 2000 times with array lengths of 10, 100, 1000, and 10,000 for each of the four algorithms. The table and graph below illustrate the results of the experiment.

N	IterLin	RecLin	IterBin	RecBin
10	319	520	464	321
100	339	303	423	209
1000	708	1972	143	183
10000	4663	20283	314	352



From these it can be seen that the data seems to back up the algorithms' big-O categories. Some observations from the graph and table are that binary search is generally faster than linear search, and that for small arrays, recursive binary is faster than iterative binary, but the case switches on big lists (considering, however, that they are so close regardless of array size, this could be an anomaly).

A surprising observation is how much slower recursive linear search is than iterative linear search. If one were to estimate coefficients for the time complexity, iterative linear search would probably be about 0.5n and recursive would be about 2.5n. It is hard to make any claims about what could be causing such a discrepancy. The only hard and fast claim that can be made from these results is that binary search overwhelmingly beats out linear search.

Answers To Problems in README:

Part 1:

Section 1:

1.2n^3 - 7n^2 + 100n - 36 is in O(n^3)
2n^3 - 7n^2 + 100n - 36 < (2 - 7 + 100 - 36)n^3
59n^3 *IS* in O(n^3) for c > 59, n > 1

2.10n + 3log(n) is in O(n)
10n + 3log(n) < (10 + 3)n
13n *IS* in O(n) for c > 13, n > 2 (since log(1) = 0)

3.(1/1000)n is in O(1)
(1/1000)n < n
n is in O(n) for c > 1, n > 1
n *IS NOT* in O(1)

4.log(n)^2 + log(n)/30 is in O(log(n)^2)
log(n)^2 = 2log(n)
log(n)/30 = (1/30)log(n)
2log(n) + (1/30)log(n) = (61/30)log(n)
log(n)^2 + log(n)/30 < (61/30)log(n)
(61/30)log(n) is in O(log(n)) for c > (61/30), n > 2
log(n)^2 + log(n)/30 *IS NOT* in O(log(n)^2)

5. (n^2)/log(n) + 3n is in O(n^2)
(n^2)/log(n) + 3n < (1 + 3)n^2
4n^2 *IS* in O(n^2) for c > 4 , n > 1

Section 2:

1. O(n)
2. O(n^2)
3. O(n^2)
4. O(n)
5. O(c)

Section 3:

1. O(n)
2. O(n^2)
3. O(nm), (where n is the number of books and m is the number of stars.
   If stars is a ranking then it's doubtful that the star count
   will ever be very large, so the big-O is probably more likely
   to be O(n).)