Gaston Aganza, Jacob Oliver

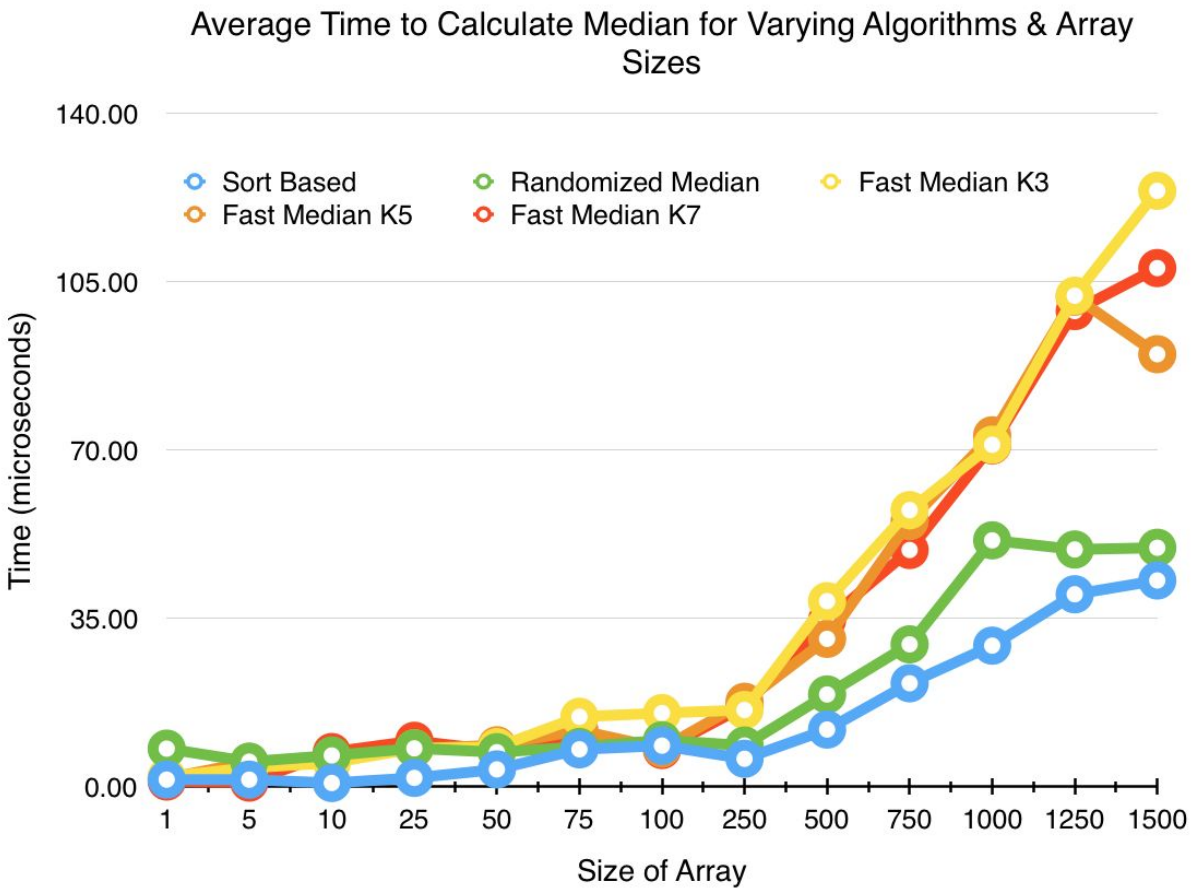
CSC 349 Lab 2 Report

This lab involved calculating the median of a given array using different algorithms. We implemented five different algorithms. Our first algorithm was a sort-based algorithm, where the arrays were sorted and the median was then either the middle element of the average of the two middle elements. Our second algorithm was a randomized median algorithm, where a random pivot position was picked, and the array was recursively divided into two arrays by comparing elements to the pivot element. Our last three algorithms (fast median K) split the array into chunks of size 3, 5, and 7, respectively, and uses the median of the medians of the chunks as the pivot element.

We tested the speed of these algorithms, especially how they behaved asymptotically as the size of the arrays were increased. We calculated the medians of the lists of a given size together 100 times for each algorithm, averaging the results.

Time in Microseconds					

Size of Array (N)	Sort-based	Randomized Median	Fast Median K3	Fast Median K5	Fast Median K7
1	Mean: 1.50; St Dev: .453	Mean: 7.94; St Dev: 2.324	Mean: 2.10; St Dev: 1.697	Mean: 1.77; St Dev: 2.655	Mean: 1.10; St Dev: 2.149
5	Mean: 1.49; St Dev: 6.009	Mean: 5.23; St Dev: 2.102	Mean: 3.96; St Dev: 4.283	Mean: 5.16; St Dev: 1.884	Mean: 0.88; St Dev: 1.597
10	Mean: 0.80; St Dev: 1.206	Mean: 6.52; St Dev: 11.608	Mean: 5.0; St Dev: 2.045	Mean: 5.93; St Dev: 0.807	Mean: 7.23; St Dev: 3.309
25	Mean: 1.86; St Dev: 1.045	Mean: 8.00; St Dev: 4.526	Mean: 7.97; St Dev: 5.474	Mean: 7.98; St Dev: 1.676	Mean: 9.57; St Dev: 3.580
50	Mean: 3.62; St Dev: 0.763	Mean: 7.20; St Dev: 6.208	Mean: 8.26; St Dev: 2.841	Mean: 8.64; St Dev: 4.215	Mean: 7.56; St Dev: 3.737
75	Mean: 7.8; St Dev: 2.234	Mean: 8.29; St Dev: 5.366	Mean: 14.56; St Dev: 2.567	Mean: 11.75; St Dev: 1.783	Mean: 11.08; St Dev: 1.727
100	Mean: 8.56; St Dev: 2.298	Mean: 9.68; St Dev: 1.463	Mean: 15.34; St Dev: 2.563;	Mean: 7.96; St Dev: 4.134	Mean: 7.66; St Dev: 5.370;
250	Mean: 5.83; St Dev: 1.422	Mean: 8.59; St Dev: 3.499	Mean: 15.97; St Dev: 1.800	Mean: 17.68; St Dev: 4.404	Mean: 16.74; St Dev: 1.998
500	Mean: 11.91; St Dev: 2.332	Mean: 19.2; St Dev: 4.899	Mean: 38.59; St Dev: 3.452	Mean: 30.77; St Dev: 2.174	Mean: 34.67; St Dev: 4.837
750	Mean: 21.53; St Dev: 3.512	Mean: 29.62; St Dev: 2.569	Mean: 57.53; St Dev: 4.542	Mean: 55.23; St Dev: 3.420	Mean: 49.23 St Dev: 5.002
1000	Mean: 29.35; St Dev: 3.021	Mean: 51.24; St Dev: 9.598	Mean: 71.11; St Dev: 5.400	Mean: 73.07; St Dev: 5.948	Mean: 70.98; St Dev: 5.674
1250	Mean: 40.03; St Dev: 4.932	Mean: 49.34; St Dev: 3.984	Mean: 102.09 St Dev: 6.039	Mean: 101.89; St Dev: 5.898	Mean: 98.90; St Dev: 6.092
1500	Mean: 42.90; St Dev: 5.099	Mean: 49.7; St Dev: 4.683	Mean: 123.90; St Dev: 5.904	Mean: 89.93; St Dev: 4.958	Mean: 107.84; St Dev: 6.348



The observed runtime behaviors of the the fast median K algorithms seem to potentially experience exponential growth, as the time required to perform the calculations increased at a greater rate as the size of the array increased. Generally speaking, the higher the K number, the slower the algorithm was. Both the sort-based algorithm and the randomized median algorithm performed better than the fast median K algorithms. As we know the sort-based algorithm grows at a rate of NlogN, it is possible that the randomized median algorithm grows at a similar rate, as its results were not far different.

It doesn't seem that any of the algorithms are overtaken by any other algorithm as the size of the arrays increases. There naturally were some changes in what algorithm took longer for the fast median K algorithms, but that seems to be simple statistical variation, as the results for those algorithms were very similar throughout testing.