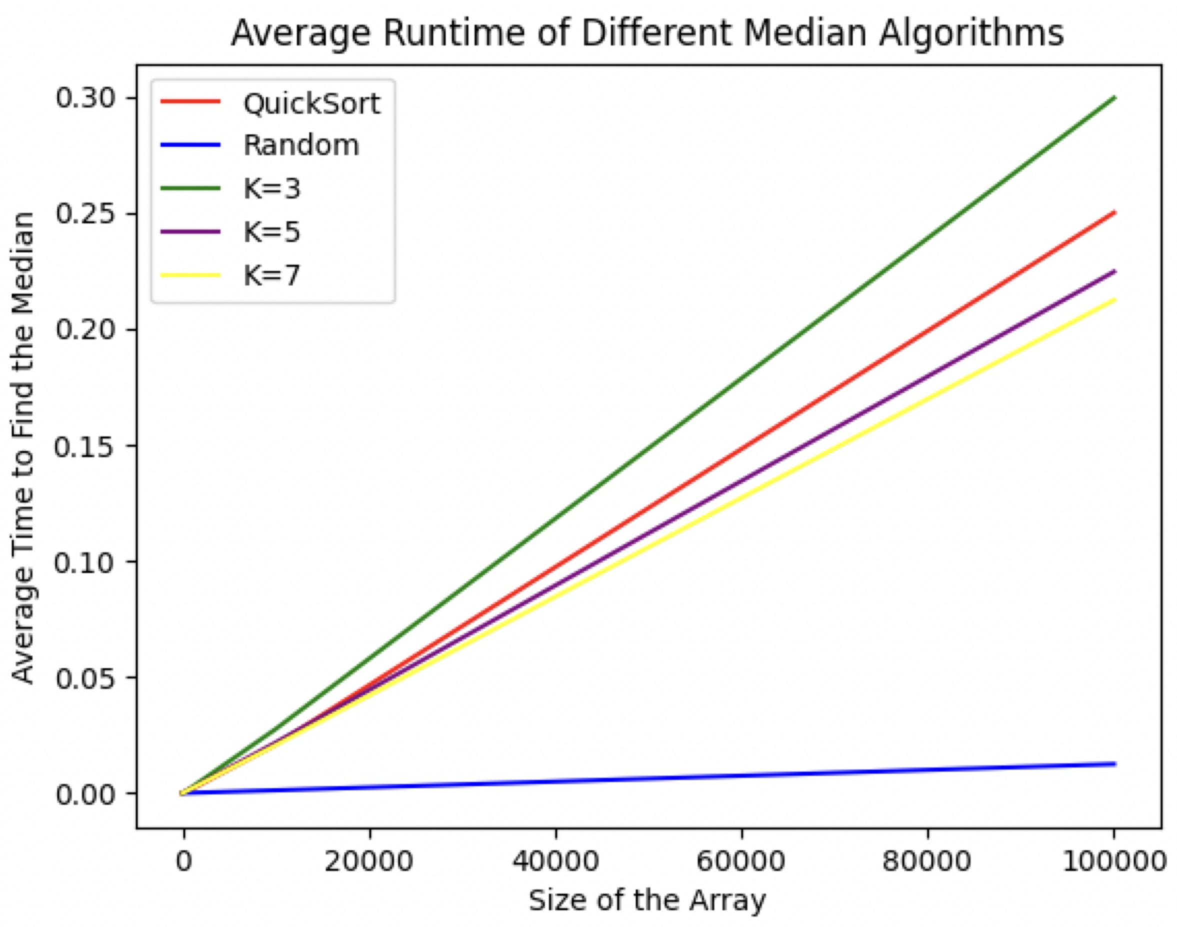
Empirical analysis of Selection Algorithms
CSC 349-01
Lab 3
Andrew Okerlund (apokerlu@calpoly.edu)
Noa Kehle (nkhele@calpoly.edu)

Medians Algorithms

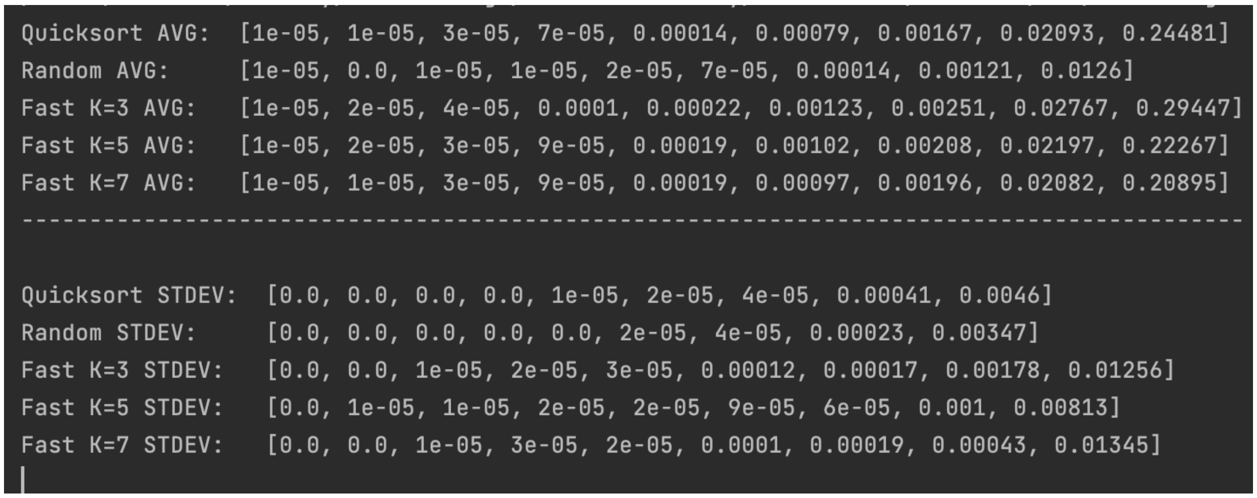
In our experiment we implemented merge sort as our fast sorting algorithm. This was then implemented into our following functions. Our testing trails consisted of arrays of the following sizes: 5, 10, 20, 50, 100, 500, 1000, 100000, 100000, 1000000 with 100 repeats at each size.

Our hypothesis originally was that the QuickSort would be the slowest, followed by random, fast_median k=3, fast_median k=5, and fast_median k=7.

Graphs:



Tables:



```
Quicksort AVG:  [1e-05, 1e-05, 3e-05, 7e-05, 0.00014, 0.00079, 0.00167, 0.02093, 0.24481]
Random AVG:     [1e-05, 0.0, 1e-05, 1e-05, 2e-05, 7e-05, 0.00014, 0.00121, 0.0126]
Fast K=3 AVG:   [1e-05, 2e-05, 4e-05, 0.0001, 0.00022, 0.00123, 0.00251, 0.02767, 0.29447]
Fast K=5 AVG:   [1e-05, 2e-05, 3e-05, 9e-05, 0.00019, 0.00102, 0.00208, 0.02197, 0.22267]
Fast K=7 AVG:   [1e-05, 1e-05, 3e-05, 9e-05, 0.00019, 0.00097, 0.00196, 0.02082, 0.20895]
------------------------------------------------------------------------------------------


Quicksort STDEV:  [0.0, 0.0, 0.0, 0.0, 1e-05, 2e-05, 4e-05, 0.00041, 0.0046]
Random STDEV:     [0.0, 0.0, 0.0, 0.0, 0.0, 2e-05, 4e-05, 0.00023, 0.00347]
Fast K=3 STDEV:   [0.0, 0.0, 1e-05, 2e-05, 3e-05, 0.00012, 0.00017, 0.00178, 0.01256]
Fast K=5 STDEV:   [0.0, 1e-05, 1e-05, 2e-05, 2e-05, 9e-05, 6e-05, 0.001, 0.00813]
Fast K=7 STDEV:   [0.0, 0.0, 1e-05, 3e-05, 2e-05, 0.0001, 0.00019, 0.00043, 0.01345]
```

Analysis:

Fastest → Random
Slowest → Fast_Medain (K=3)

Runtimes for each:
Quicksort → O(n log(n)) but looks like o(n) on the graph
Random → O(n) but far less than any of the others
Fast 3 → O(n)
Fast 5 → O(n)
Fast 7 → O(n)

After the testing trials concluded there was a very clear observable differences in performance between all of the algorithms. For starters, when scrutinizing the *Fast_Median* algorithm it was observed that the larger the subarray (size K) the faster the algorithm was able to produce the median. The observed runtime behavior for the three K values tested of 3, 5, and 7 with an array of 100,000 were: 0.29943 seconds, 0.22459 seconds, and 0.21235 seconds with the most substantial difference between k = 3 and k = 5 and a fastest runtime when K = 7. The reason for this is when K is a larger value the number of recursive steps to get to the basecase is greatly reduced, additionally because the increment in K value is small the time to sort the subarray of size K is negligibly increased. On the contrary when K = 3 the run time is slower. This is because the algorithm will have to recurse more times to reach the base case, while there are less terms in each subarray to sort this change actually hurts its performance because there are more subarrays to sort.

One interesting result is how the speed at which the *Random* function finishes. Our hypothesis for this is that random doesn't execute any sorting steps when calculating the pivot which in turn saves it time. In this instance where we implemented our own version of the Sort inside the base case and for the small_medians function, it is clear that it is causing the fast_median algorithms to be significantly slower than the random. We believe that the fast_medains algorithm with k = 3,5,7 would be the fastest if and only if the sorting was fully optimized, as well as the function that splits it into groups. We can conclude this because the random_median does not perform either of these steps and thus is the fastest, contrary to our original belief. The asymptotic trends for all functions appear linear on the graph, and are written above.