UC Davis. ECS122b June 6, 2016
Guang Xu Chris Ng

Program 2 Report

1 Group Memebers

Student 1: Guang Xu ID:998309428
Student 2: Christopher Ng ID:998311403

2 Introduction

For our experiment we compared the run times for three algorithms, each essentially building upon the previous for a theoretically more optimal performance. First off we have Quicksort-Select, which is essentially quicksort, and afterwards finding the kth element. To do this we:

1. Pick a random element in our array data to be the pivot.
2. Data to the left and right of the pivot will then be swapped so that the resulting array will have all elements less than the pivot to its left and all elements greater than the pivot to its right, with the pivot in its final, sorted position.
3. The left and right partitions of the array will then have quicksort done upon them as well, and it will be recursively called until the entire array is sorted.

Because our goal is only to find the kth element, it is unnecessary to recursively preform quicksort on both the left and right partitions after they have been sorted relative to the pivot, hence the use for quickselect, which only recursively calls itself on the partition that the kth element is in. Additionally, if the kth element happens to be a pivot, there is no longer a need to continue sorting the array, as the kth element has already been found, and thats another way in which quickselect saves time over quicksort. So for this we:

1. Pick a random element in our array data to be the pivot.

2. Data to the left and right of the pivot will then be swapped so that the resulting array will have all elements less than the pivot to its left and all elements greater than the pivot to its right, with the pivot in its final, sorted position.

3. If the index of the pivot is less than k, then the right partition will have quickselect done on it, else if the index of the pivot is greater than k, the left partition will have quickselect done on it. Otherwise, if the index of the pivot is the kth element, then we are done.

A problem with quickselect is that its run time is extremely sensitive to the pivots chosen. As they are random, they can range from the pivot being the median of the array and thus partitioning the array in two equally-sized partitions which would be a best case, or essentially only reducing an arrays size by one if it is the largest or smallest element in the array. Deterministic-Select in theory improves upon quickselect by utilizing an algorithm to find a number with an index close to the median to use as the pivot instead. For the implementation of this we:

1. Partition the array into groups of five

2. Get the median of each partition

3. Get the median of the medians and use as pivot

4. Data to the left and right of the pivot will then be swapped so that the resulting array will have all elements less than the pivot to its left and all elements greater than the pivot to its right, with the pivot in its final, sorted position.

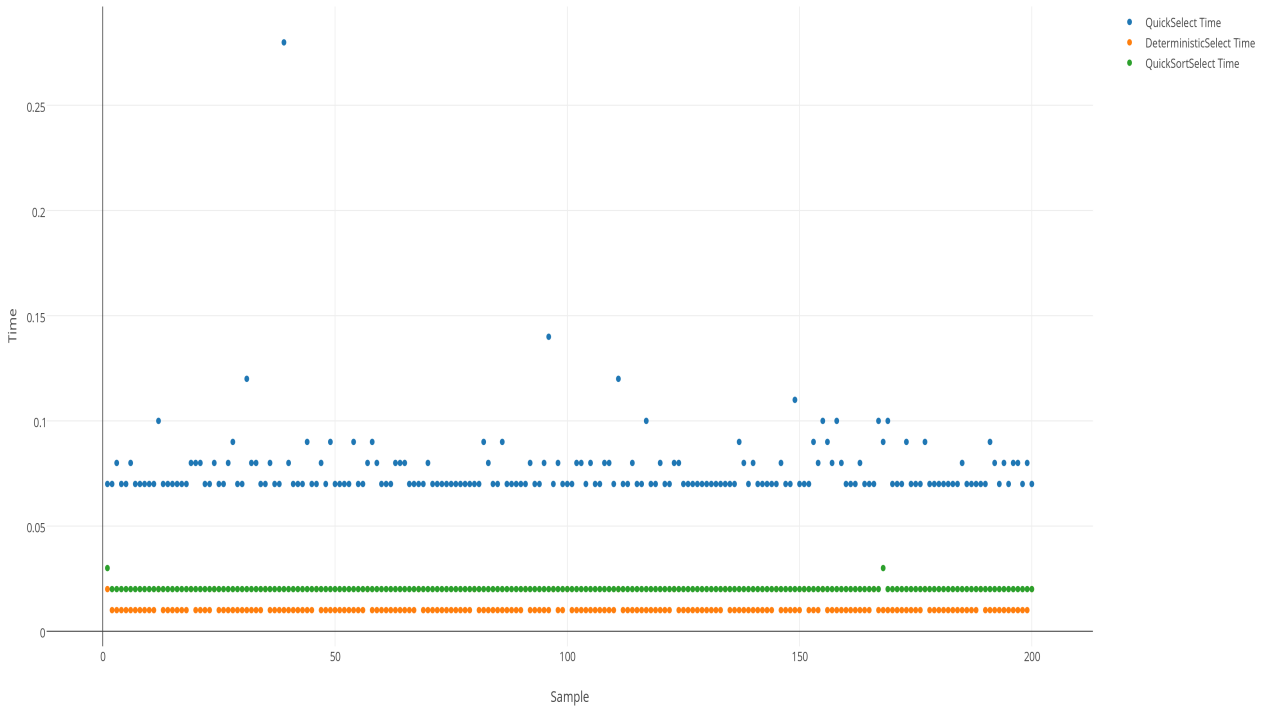
5. If the index of the pivot is less than k, then the right partition will have deterministic select done on it, else if the index of the pivot is greater than k, the left partition will have deterministic select done on it. Otherwise, if the index of the pivot is the kth element, then we are done.

3 Empirical Studies

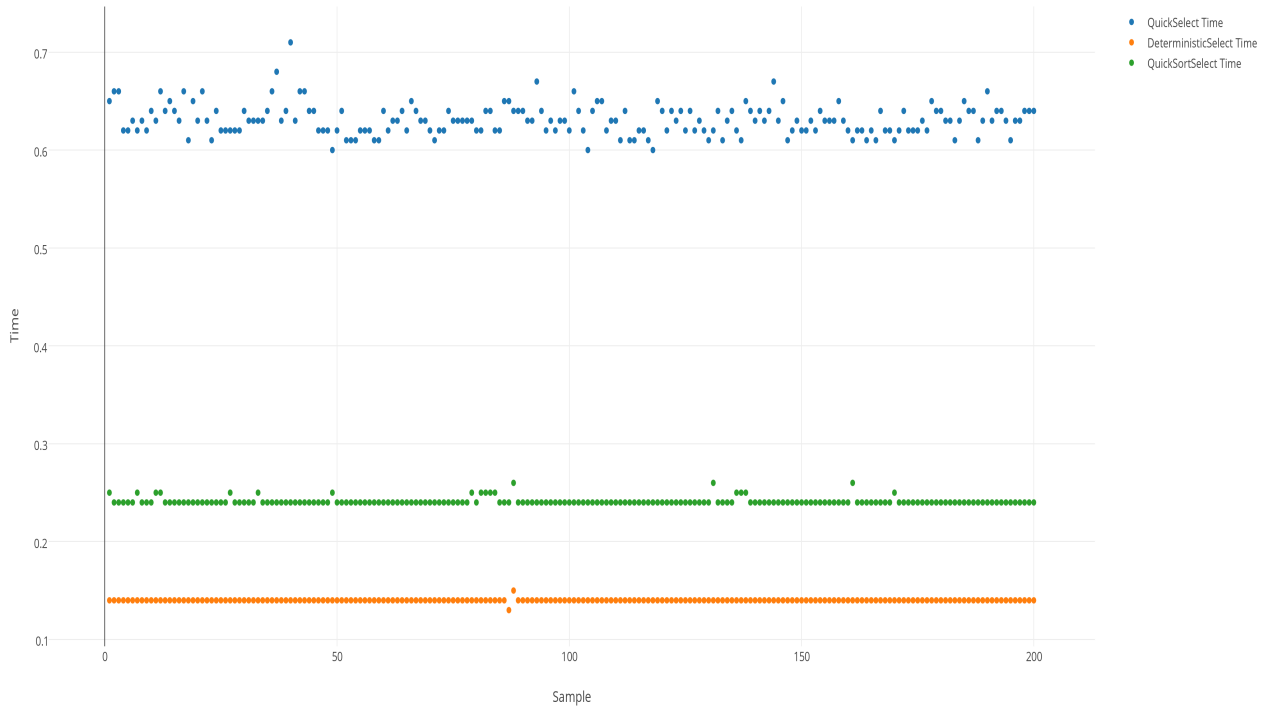
For our experiment, we analyzed run times and the number of partitioning stages on a range of data sizes and data sets. Using a bash script we generated data sets of random numbers ranging from one to one million, with data sizes of $10^2, 10^3, 10^4, 10^5, 10^6$, 200 samples each. Using another bash script, we used the three algorithms on all the data sets, to compare their respective run times for data sets of the same size, and inputted their run times into a csv files, one for each data size, in order to view general trends in their run times. We also included partitioning stages into our csv files in order to get an idea of how their respective number of partitioning stages compare to each other and how they relate to run times.

4 Conclusion

For our data sets with sizes less than 10^5, run times were too fast to compare, as they would finish in almost the same time. For our data sets of size 10^5 and 10^6 however, with the exception of a few outliers, run times for the three algorithms have remained fairly consistent. Determinist select is the fastest algorithm as expected. However, quick select is far slower than quick sort with finding the kth element which should not be the case. Although vexing, we are unable to determine why this is the case. In examining their number of partitioning states, quick select has far fewer which would make us think that it would be faster, as quicksort also sorts the parts we dont need.This goes to show that despite some algorithms being clearly faster than others in theory, other factors such as hardware design and such could result in them being slower in practice.



10^5 Data Plot



10^6 Data Plot

5 Citation

QuickSortSelect:

1. Analysis: https://en.wikipedia.org/wiki/Quicksort
2. Code: Modified from program 1

QuickSelect:

1. Pseudo Code: https://en.wikipedia.org/wiki/Quickselect

DeterministicSelect:

1. Pseudo Code: https://en.wikipedia.org/wiki/Median_of_medians
2. Analysis: https://www.ics.uci.edu/ eppstein/161/960130.html
3. Analysis: http://eshrews.com/select