Program 2 - ECS 122B, Spring 2016

David Lin, Jonathan Kim, Mark Fan

1 Introduction

The Problem:
In program 2, we are using Quick Select, Deterministic Select, and a sorting algorithm (arbitrarily selected and implemented insert sort) in order to select the specified k'th largest integer in a set of integers passed to the program that utilizes these algorithms. These programs' run-times were recorded in order to compare how quickly each algorithm returns the k'th order integer from data sets of varying sizes (10^2 integers to 10^6 integers).

Pseudo-code:
Quick Select:

```
partition(int [] array, int lo, int hi, int pivotIndex) {
    pivot = array[pivotIndex]
    swap(array[pivotIndex], array[hi])
    int wall = lo

    for (int i = lo; i < right; i++) {
        if(array[i] < pivot)
            swap(array[wall], array[i])
            wall++
        end if
    end for

    return wall
}

quick_select(int [] array, int lo, int hi, int kOrder) {
    while(TRUE) {
        if (lo == hi) {
            return array[lo]
        }
        int pivotIndex = lo + RANDOM(hi - lo + 1)
        pivotIndex = partition(array, lo, hi, pivot)

        if (kOrder == pivot)
            return list[kOrder]
        else if (kOrder < pivotIndex) {
            right = pivotIndex-1
        }
        else {
            left = pivotIndex+1
        }
    }
}
```

Deterministic Select (Median of Medians):

```
det_select(int [] array, int kOrder) {
    if(array.size() <= 10)
        sort array
        return array[kOrder-1]
    end if

    subset[i] = partition of array in groups of 5

    for(int i = 0; i < array.size()/5; i++)
        medianOfGroups[i] = det_select(subset[i], 2) // The center element
    end for

    medianOfMedians = det_select(medianOfGroups, n/10)

    partition array such that left < medianOfMedians, mid = medianOfMedians, right

    if (kOrder <= length(left))
        return det_select(left, kOrder)
    else if (kOrder > length(left) + length(mid))
        return det_select(right, kOrder-length(left)-length(mid))
    else
        return medianOfMedians
    end if
}
```

Select k'th order using Insert Sort:

```
insert_sort(int [] array, int kOrder) {
    for(int i = 1; i < array.length; i++)
        j = i
        while(j > 0 && array[j-1] > array[j])
            swap(array[j] array[j-1])
            j = j-1
        end while
    end for
    return array[kOrder-1]
}
```

2 Empirical Studies

For this project, we gathered data on the run-time performance of the 3 different types of selection algorithms (Quick Select, Deterministic Select, and Selection using Insert Sort) for random data sets containing 10^2 integers, 10^3 integers, 10^4 integers, 10^5 integers, and 10^6 integers. Each input size was sampled 200 times for each algorithm using our own shell script called gatherData.sh and genAll.sh, which simply runs the programs using these 3 algorithms using the samples of data sets and a randomly generated k'th order as input. This data was all gathered using the UC Davis csif machines.

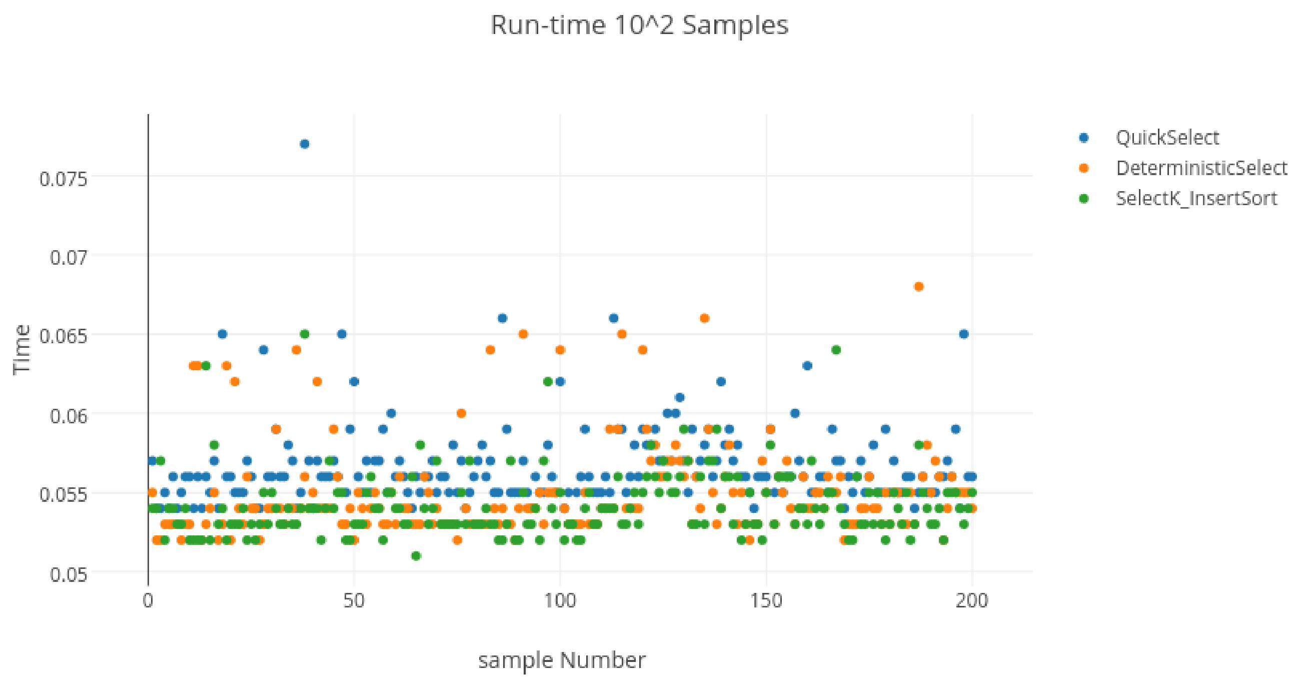
3 Conclusions

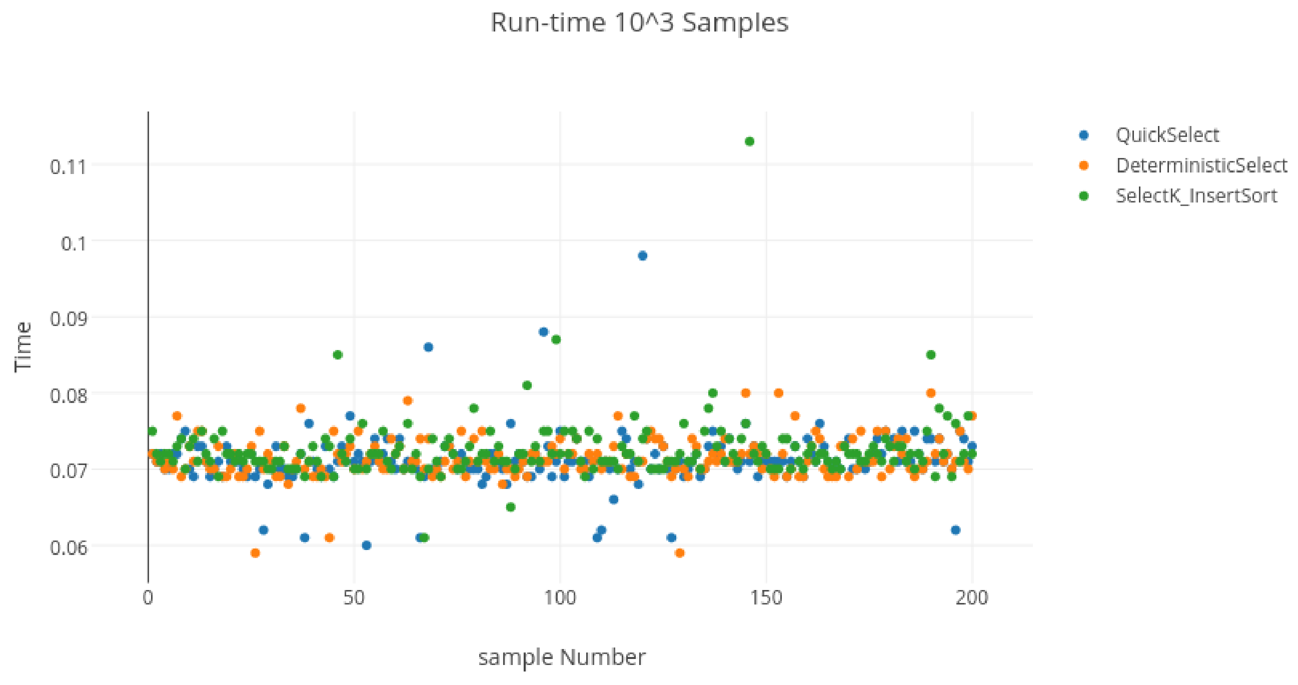
The quick select algorithm has a worst case running time of $O(n^2)$ but its average running time is $O(n)$ by partitioning the array in two parts - integers that are less than or equal to the pivot, and integers that are greater than the pivot. The algorithm then determines which partition the k'th order is in (or if it was found already, it is returned) and searches in that partition while ignoring the other partition where the k'th order is not in.

The deterministic select algorithm, median of medians, has a worst case running time of $O(n)$, and best case performance of $O(n)$. And insert sort has a worst and average case running time of $O(n^2)$ comparisons and swaps.

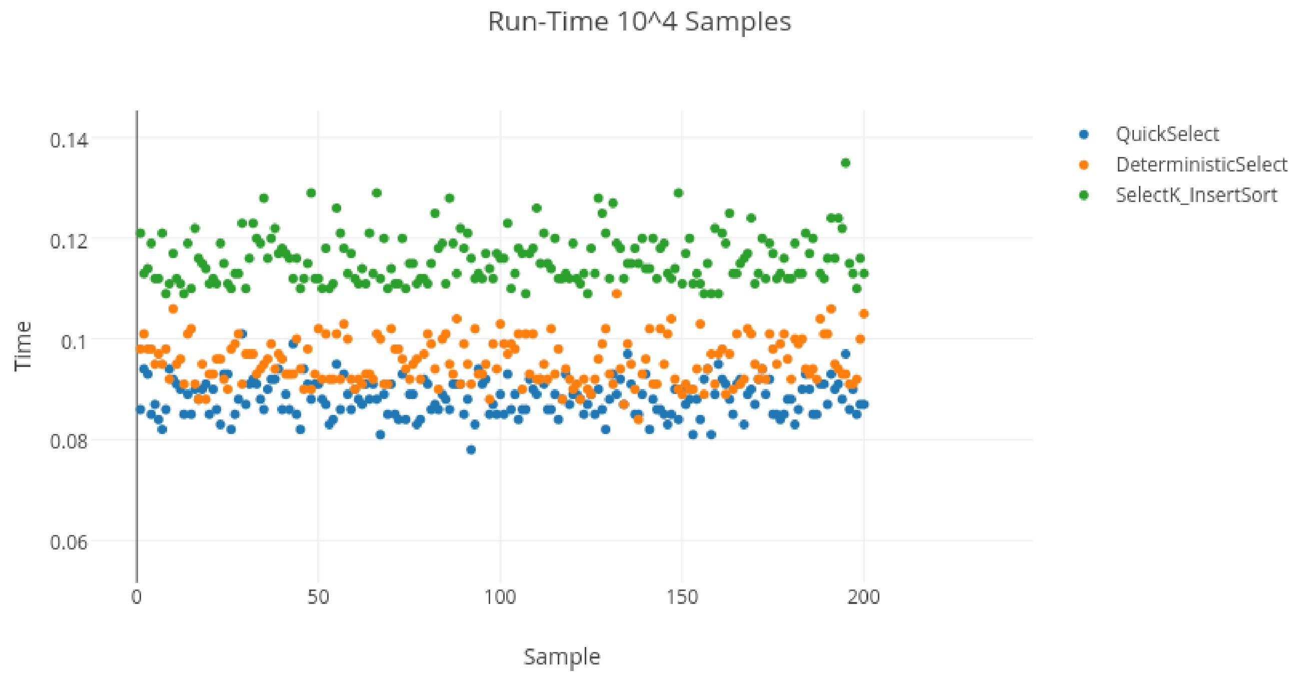
And as expected, generally the insert sort algorithm performed the worst because it always needed to work with the entire array while being an inefficient sorting algorithm for such large data sets. Quick select generally performed better than the sorting algorithm, and worse than the median of medians deterministic select algorithm since it randomly determined a pivot, which is not guaranteed to partition the array effectively. And the deterministic select algorithm that calculates an estimated median of medians to get as close to splitting the array in half as close as possible at each iteration without spending too much time on calculations performed the best overall.

Refer to the graphs in this section. We can see from the graphs showing the run-time of the 3 algorithms that, up until 10^4 samples, they ran at approximately the same speed.

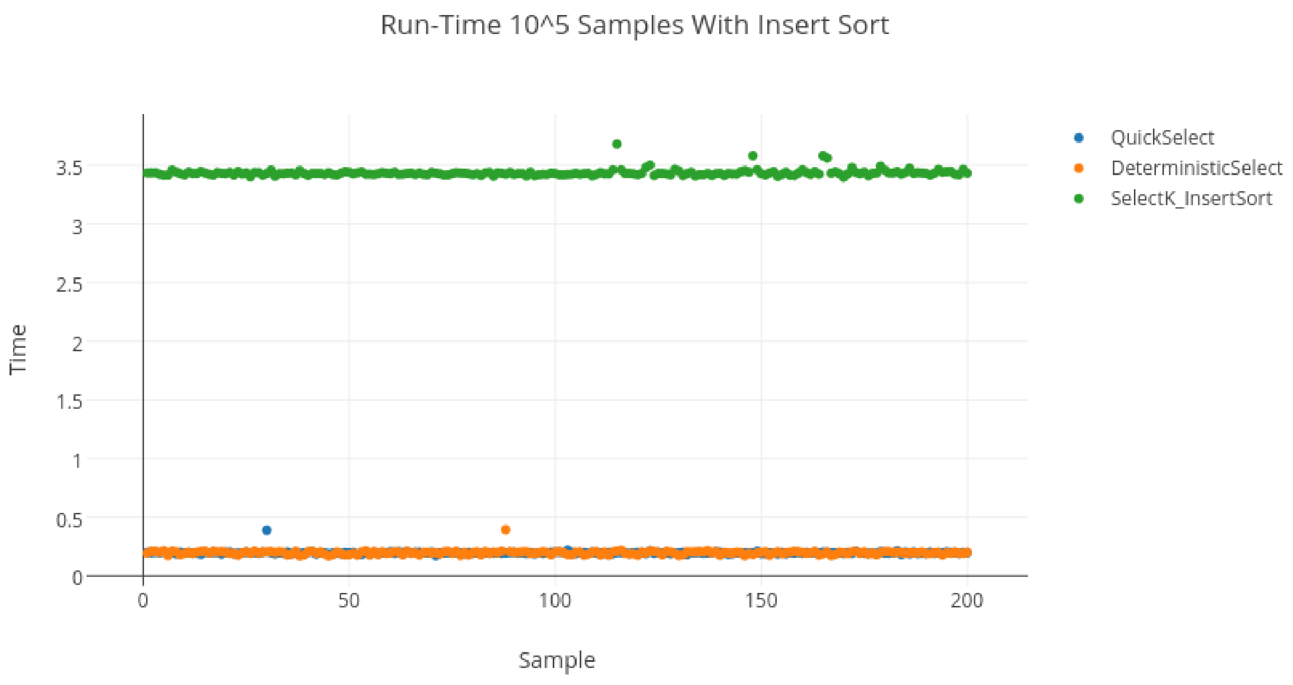


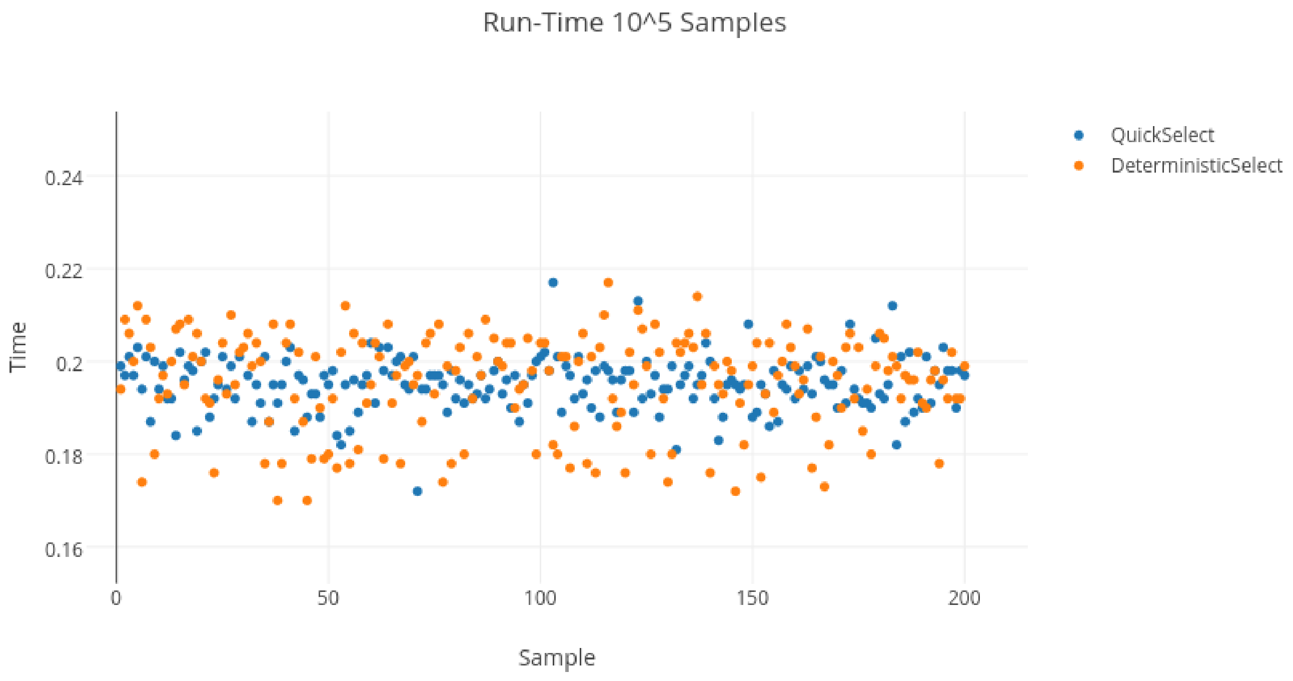


Once we start collecting the run-time using data sets containing 10^4 integers, we notice that generally, the insert sort algorithm is much slower than both the deterministic select algorithm, and the quick select algorithm. And for the 10^4 integer data set size, quick select is generally faster than the deterministic select algorithm.

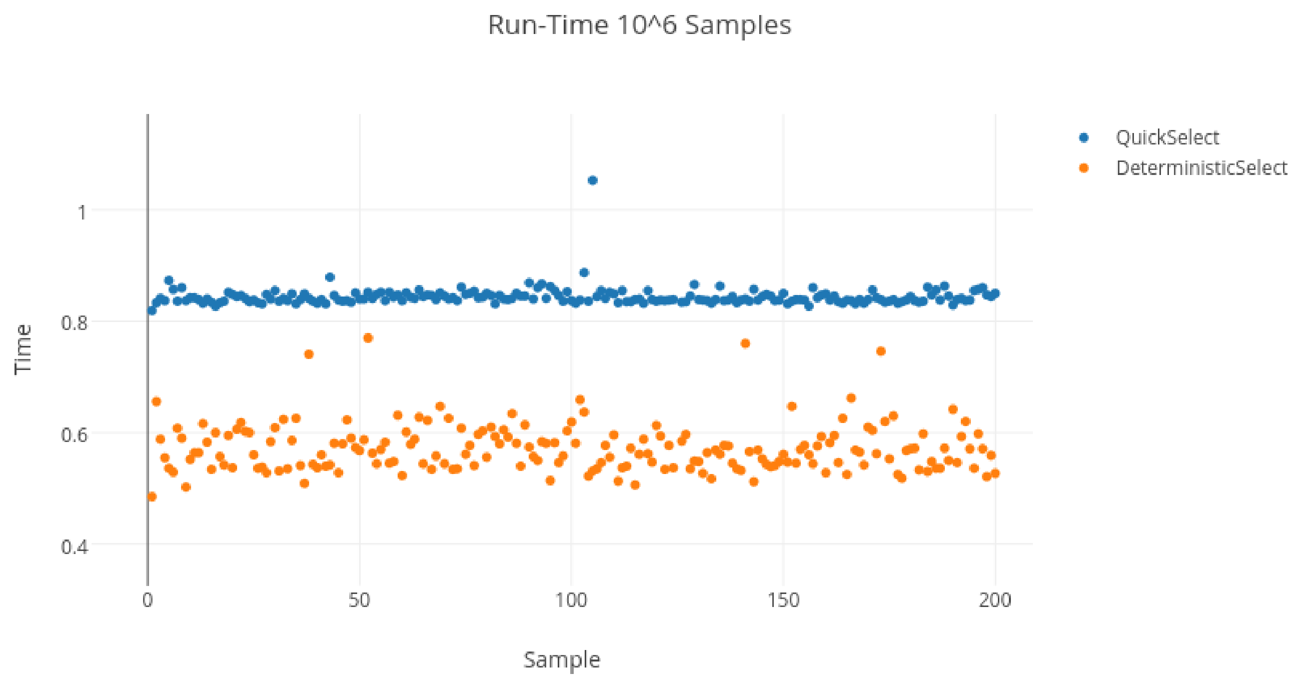


Insert sort became way too inefficient at 10^5 integers and higher as you can see in the graph "Run-Time 10^5 Samples With Insert Sort". Here we can see that quick select has a more consistent run-time than the deterministic select algorithm, while the deterministic select algorithm sometimes runs faster than the quick select algorithm and sometimes runs slower.





Finally at 10^6 integers, quick select takes longer to find the k'th order element than the deterministic select algorithm.



4 Citations

"Insertion Sort." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 21 May 2016. 01 June 2016. <https://en.wikipedia.org/wiki/Insertion_sort>

"Quickselect." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 02 June 2016. 03 June 2016. <https://en.wikipedia.org/wiki/Quickselect>

"Median of medians." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 01 June 2016. 03 June 2016. <https://en.wikipedia.org/wiki/Median_of_medians>

"ICS 161: Design and Analysis of Algorithms Lecture notes for January 30, 1996." 03 June 2016. <https://www.ics.uci.edu/~eppstein/161/960130.html>