Tonny Mutumba

Algorithms CS4050

Assignment 1

ABSTRACT

*In this paper, I experimented about what quicksort variation runs the fastest. We study that quicksort is close to be one the fastest sort methods ever written. However, there are a series of different ways quicksort is implemented. I am going to explore some different methods that seem to be used in most of these quicksort implementations. i.e. Partitioning, Median of 3, Stopping before the bitter end (Insertion sort).*

INTRODUCTION

Quicksort is one of the most efficient sorting algorithms based on the splitting of an array into smaller ones and swapping them into their rightful positions. The worst complexity of Algorithms is *0(N)^2*, which quicksort lies right close to, but it can also be stretched to greater levels of *O(NlogN)*. Complexity of this algorithms defines so much what it can do and how long it will take for any list or array to be sorted.

In this quicksort implementation, I will try to use some of the different ways mentioned in the abstract with an algorithm complexity that lies between O(N)^2 to O(NlogN). In the Partitioning implementation, I do employ two schemes, that is; Lomuto and Hoare. Lomuto scheme uses the median-of-three rule counter, in which a better estimate of the pivot is determined as the true median within the partitions. Hoare, on the other hand, uses two indices that starts at the end of the array being partitioned. Both of these schemes can degrade quicksort to O(N)^2 for the already sorted input but this can be omitted while implementing the code by using a floor function which I implement as recQuicksort methods within the code.

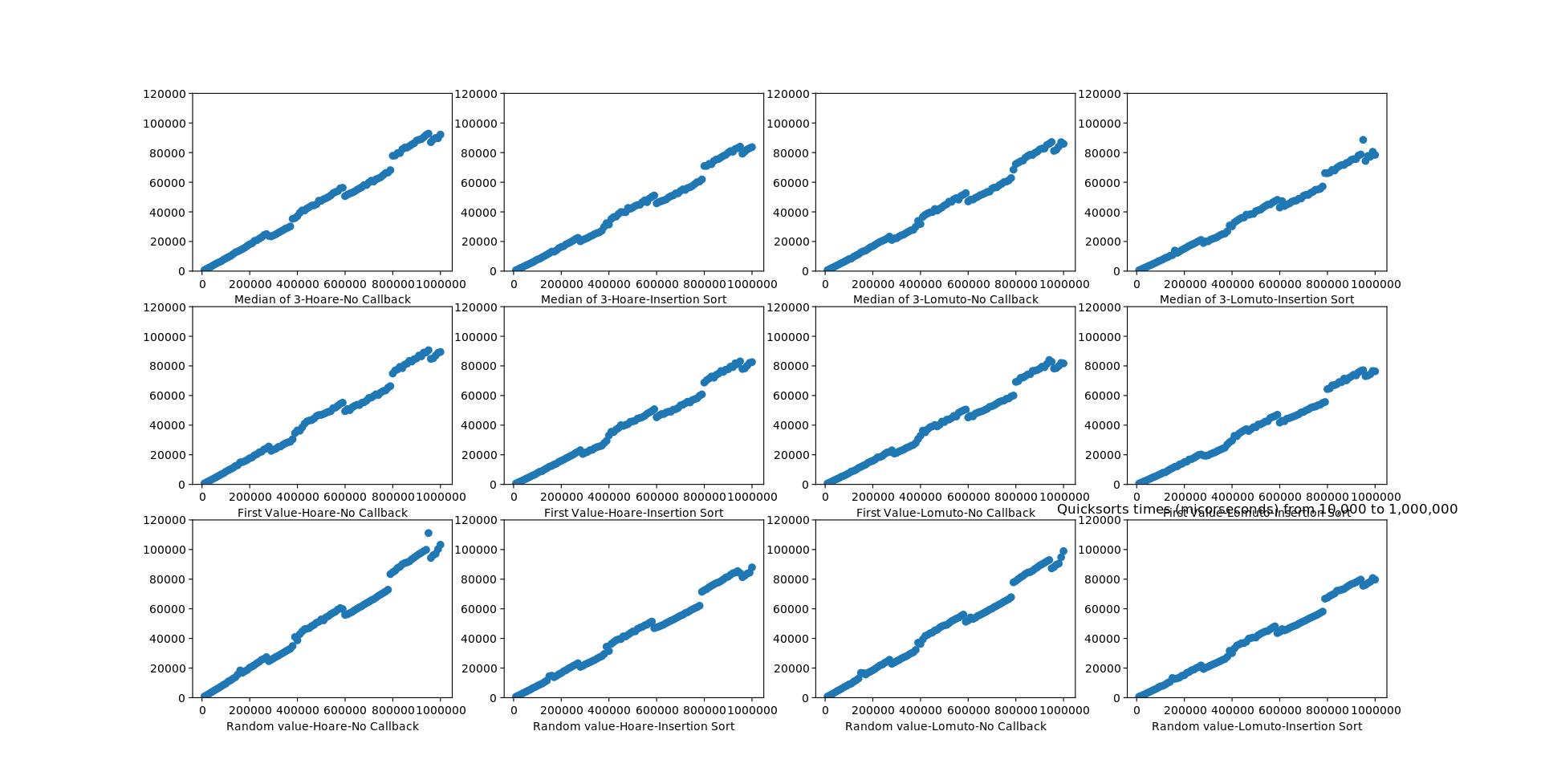
While the insertion sort was used in half the samples. The sort performs few swaps, comparisons and other operations in small arrays and normally takes an algorithms complexity of O(kn). The optimal cut-off threshold switching from quicksort to insertion sort is taken as 10 in my program.

ANALYSIS

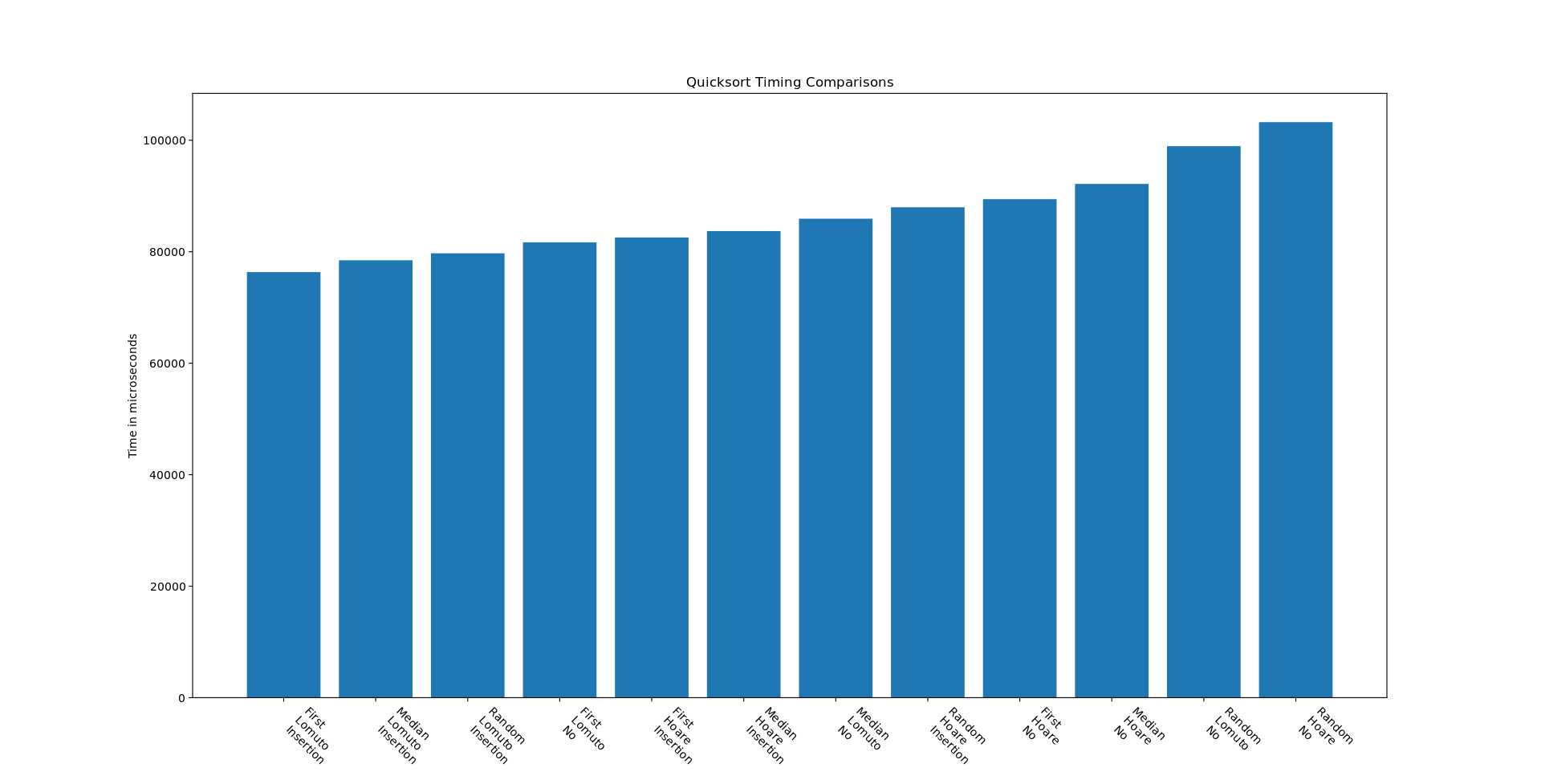
Quicksort’s worst-case and average case running times differ significantly, due to the different way the partition is implemented. From our data, the variations were significant when it came to the size of the sample and this data sizes. The running times also differed when it came to the size and kind of implementation.

Data;

For the different sorting schemes, I used a sample data of 10,000 to 1,000,000 elements.



The times are considered as the averages of the system clock for different arrays.



CONCLUSION

In this assignment, I was supposed to examine what the best efficient and effective way to implement quicksort is. The results obtained from the experiment don’t differ some much for the same size elements. But the small run times differences are significant in determining what implementation what fastest amongst the implemented on and with all the data obtained the median-of-three seemed to be having the better run times. Besides the fact that also first Lomuto insertion had fast run times, the differences came by when the sample data was somewhat sorted elements, the algorithms complexity dropped down to O(N)^2.

Reference:

1. <https://www.interviewbit.com/tutorial/quicksort-algorithm/>
2. <https://en.wikipedia.org/wiki/Quicksort#Lomuto_partition_scheme>
3. Got some help from Luke S