Joshua Brown

Dr. Aaron Gordon

February 8, 2020

Quick-sort Analysis

Quicksort is a well-known algorithm used to optimize the runtime of sorting data sets of various sizes. Often regarded as a divide and conquer sort, it works by selecting a pivot element from an array and partitioning all other elements into two subarrays. The sub-arrays are then sorted continuously until the entire array has been completely sorted. For the purposes of this experiment, runtimes from four various quick sorts were gathered and put into a table for comparison.

The Lomuto partition uses the first element in the array as its pivot while the rest of the array is classified into lower and higher subarrays. The algorithm then moves all the values that are less than the pivot to the left, and the values that are greater than or equal to the pivot to the right. This method is applied to the subarrays until the original array has been completely sorted. Another variation of quicksort is called the Hoare partition scheme, it starts by initializing two indexes at either end of the array. The two indexes move toward each other until an inversion occurs. Once that happens, the two values are swapped, and the process is repeated. The next variation of quicksort is applying a median of 3 process to the Lomuto partition. The median of 3 method is used to pick the pivots of the array which include the first, last, and middle elements. Once those pivots are established, the Lomuto and Hoare partitions move on as usual. The last variation of quicksort that was tested is the “stopping before the bitter end” which uses the partitions named above until the partitioned data becomes less than some chosen value. Once that step is finished, insertion sort is run once over the array to complete the sort.

The experiment begins by filling an array with randomly generated numbers and creating multiple copies of that array. There is a timer set before each sort begins, and it stops when the sort ends. In order to get well tested results, the array was filled with one million, ten million, and fifty million elements. The sorts were run five times and the average was taken from each five and are displayed in Table 1 below. From the results we can conclude that the Hoare algorithm runs faster than the Lomuto algorithm. This is because Hoare’s partition does three times fewer swaps on average, and it creates efficient partitions even when you have many of the same values. Some factors that can cause variance in times is how the arrays are built and the hardware that it used.

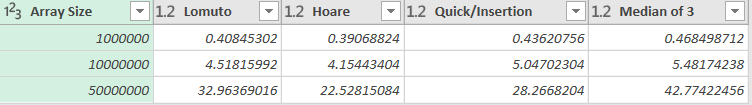
The next experiment was to find the threshold of the insertion sort. The purpose of the threshold is to find the best size of the partitioned array to run insertion sort on. This experiment was run using the same data sizes as before, and the average times can be seen in Table 1 below. The data shows that when the threshold value is relatively low, the time improves dramatically. When the threshold value was between 15 and 20 the best times were recorded. Once that threshold began to grow to a slightly larger number the method became less efficient.

That last experiment run was the median of 3 method to determine the pivot of the quicksort. Since we are choosing the middle element as the pivot, we can say that this is the best-case scenario. However, on average this method proved to be the most inefficient no matter how many elements were tested. It had closer results to the insertion/quicksort than that of the Hoare partition or Lomuto partition. Since we are choosing the middle and end elements in the array, different results were expected. The median of 3 was expected to compete with the Hoare algorithm, however, it performed closer to the insertion/quicksort.

What can be taken away from this experiment is that the Hoare partition had the shortest run time on average. The Lomuto algorithm took slightly longer than the Hoare partition. The insertion sort and median of 3 sorts take the longest on average. Strengths and weaknesses can be seen from looking at the data for the median of 3 and insertion quicksort’s. When the insertion sort threshold is smaller, the sort runs quicker than it would with a larger threshold. The median of 3 sort has a better runtime on average when there are more elements in the array.

There is a clear benefit from using the Hoare partition, as it shows better performance. Although the other quick sort algorithms work effectively, they do not compare to the speed of the Hoare partition. Some internal factors can be held responsible for this performance such as hardware and the way the code was written. This means depending on how the code is written, different results could have occurred. The final conclusion drawn from this experiment is when the need arises to quickly sort large arrays, the Hoare partition is the best algorithm to use.

Table 1:



Works Cited

“Hoare's vs Lomuto Partition Scheme in QuickSort.” *GeeksforGeeks*, [www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/](http://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/).

“Quick Sort Implementation with Median-of-Three Partitioning and Cutoff for Small Arrays.” *Java Tips*, www.java-tips.org/java-se-tips-100019/24-java-lang/1896-quick-sort-implementation-with-median-of-three-partitioning-and-cutoff-for-small-arrays.html.

[Hoare, C. A. R.](https://en.wikipedia.org/wiki/Tony_Hoare) (1 January 1962). ["Quicksort"](http://comjnl.oxfordjournals.org/content/5/1/10). *The Computer Journal*. **5** (1): 10–16. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1093/comjnl/5.1.10](https://doi.org/10.1093%2Fcomjnl%2F5.1.10). [ISSN](https://en.wikipedia.org/wiki/International_Standard_Serial_Number) [0010-4620](https://www.worldcat.org/issn/0010-4620).