Technical Specification Report on Quicksort

Charles Calvert

March 15, 2019

Descriptive Abstract

This report describes the implementation of the divide and conquer algorithm quicksort on an unsorted array of integers through the use of comparisons of elements, swapping elements, partitioning the array, and recursion. The implementation of the partition algorithm within quicksort will describe how partitioning the array and recursion is accomplished. This report also examines the running time of quicksort and compares that running time to other divide and conquer algorithms, such as binary search and merge sort. An experienced Computer Science programmer will recognize and understand the advanced aspects of this algorithm.

Introduction

The aim of this report is to expand on the Technical Description of quicksort for a more experienced Computer Science major with a focus on programming. Students who have been studying Computer Science for a few years will have an easy time understanding how quicksort is executed through simple “subproblems [that] may be resolved to produce yet simpler problems,” along with how well quicksort performs (Hoare 10).

In this Technical Specification Report, I will go over the implementation of the quicksort algorithm and the implementation of the partition algorithm, examine the running time of quicksort, conclude this report, and list the references I have cited. First, examining quicksort’s implementation will set the groundwork for the main algorithm. Then, looking at how the partition algorithm plays a part in quicksort will help explain exactly how arrays are split into two subarrays and how recursion of quicksort will follow. After that, analyzing the running time of quicksort will help put the algorithm into perspective amongst other similar algorithms. Finally, I will summarize what I have written in this report for the conclusion. Following the main part of my report will be a list of sources I used in writing it.

Implementation of Quicksort

Quicksort can be implemented with numerous data structures, but it is most often used with arrays. Specifically, unsorted arrays of integers can be sorted with quicksort easily. Let’s say we have an array named X. What quicksort does is split X into two subarrays: X1 and X2. X1 will consist of elements less than or equal to the elements in X2. Through repeatedly splitting these subarrays further and further, the original unsorted array will eventually become sorted. Following two recursive calls, quicksort’s execution will be complete (Dasgupta 56).

Implementation of Partition

The partition algorithm does the main work in quicksort. Again, let’s say we have an unsorted array named X. We will also use the two subarrays of X again: X1 and X2. X1 and X2 will each be the same size as X. Our main goal is to sort X. To do this, we will have to compare the numbers in X with some other number. In this implementation of partition in quicksort, let’s simply have that number be the last element of array X and call it i. Starting from the beginning of X, we will look at every element except for the last one, since that is the element named i we have chosen to compare the other elements with. As we scan through each element of X, we will compare that element with i. If the element is less than or equal to i, we will sort that element into X1. If the element is greater than i, we will sort that element into X2. After we have sorted each element into their appropriate subarray, we copy the elements in X1 back into X. Then, we insert i into X after each of those elements. After i, we copy the elements in X2 into X. i is inserted into X in between the elements from X1 and X2 because that is i’s correct position in the array.

Now, it is still possible that each of the elements in X aside from i are still less than or equal to i, meaning that the array X remains unsorted. But, this is where recursion on the elements copied from each of the subarrays will come into play. A recursive call will be made back in the main quicksort algorithm, repeating the same implementation of partition on each of the elements in X up until i, and each of the elements in X after i. Through these recursive calls, X, will eventually end up as a sorted array.

Running Time of Quicksort

Looking at quicksort’s running time, it is “significantly faster than any of the common sorting algorithms” (“Quicksort Algorithm”). The comparisons made within the partition algorithm are the main operations quicksort makes. Specifically, these comparisons are where we examined the elements in X against i. Examining these comparisons and how often they are made indicates that quicksort’s running time, in a best case where the partition algorithm picks an appropriate middle element for pivot each time it is called, is asymptotically equal to nLogn. n here represents the total number of elements in any given data structure quicksort is run on. Furthermore, quicksort’s running time, in a worst case where a data structure is already sorted or is reverse sorted, is asymptotically equal to n2.

Looking at other sorting algorithms, merge sort, for example, may have a better running time than quicksort, but quicksort has the advantage of not using any additional storage data structures for temporarily placing elements that need to be sorted. Quicksort also does not copy elements back and forth in an array.

Conclusion

In summary, quicksort is an excellent algorithm to use when sorting a data structure. In the use of an array, quicksort can easily compare elements, split these elements based on comparisons, perform recursion on these elements in subgroups, and result with a combined and fully sorted array. The design of quicksort is important to know so that Computer Science students can implement the algorithm themselves in their future courses. The analysis of quicksort will also familiarize students with looking at an algorithm’s running time, giving them an idea of how the effectiveness of algorithms is determined.

References List

Hoare, C. A. R. “Quicksort.” *The Computer Journal*, vol. 5, no. 1, 1 Jan. 1962, pp. 10–16.,

doi:10.1093/comjnl/5.1.10.

Dasgupta, Sanjoy, et al. *Algorithms*. McGraw-Hill, 2008.

“Quicksort Algorithm.” *InterviewBit*, www.interviewbit.com/tutorial/quicksort-algorithm/.