Data Structures and Algorithm Analysis

Report detailing the results of Quick Sort optimization

Peter Maynard
Aberystwyth University
Department of Computer Science
pem9@aber.ac.uk

ABSTRACT

This report is to find out the best value at which to change down to a alternative sort, as well as which alternative sort to use. As quick sort is slower at sorting small amount of items than other sorts such as insertion and bubble sort. It will also find out at which point is the optimum value to switch sorts. The way in which this is done is by running the sorts on many different data sets, which have a varying numbers of items, and the length of each sort is recorded and used in conjunction with the tests run again but with a varying cutoff value ranging from five up to five hundred in steps of five. These tests are run ten times each and the average value returned.

The results show that the best alternative sort to switch down to is the insertion sort due to it fast sorting of small items. The optimum cutoff value is twenty, this allows for faster sorting than quick sort with out an alternative sort. It was found that if the cutoff value was too high then the length of time for the sort to complete would be increased greatly compared to the standard quick sort. Bubble sort was used as an alternative sort, but was found to take much longer than the alternative insertion sort. It also does not make a difference what size the data sets are the average cutoff point is around 20, even if there are five million items compared to one million.

General Terms

Theory, Computer Science

Keywords

Sorting, Algorithms, QuickSort, BubbleSort, Performance

1. INTRODUCTION

This report discuses the results of optimizing quick sort, by using an alternative sort when it reaches a cutoff value. This is to find out if it is quicker compared to a standard sort. As alternative sorts such as comparison sort are quicker at sorting smaller amount of numbers.

This report will cover different types of alternatives sorts, such as bubble sort and insertion sort, to find out which one of these are faster used in conjunction with quick sort. It will also use different cutoff values to find the optimum value for performance. These tests will be run on a collection of different data sets with varying items to sort along with sorted data sets.

2. METHODOLOGY

To make sure that all the tests are as accurate as possible, they were all run on the same hardware, running in the same conditions for all of the tests.

The first test was timing the elapsed length of time it took for a basic quick sort to complete. The basic quick sort does not use an insertion sort once it gets to a certain value. This was run on all of the data-sets. To make sure that the results were conclusive. The test is to time the length of the sort ten times and return the average length of time. This was done one hundred times, which means that the sort was run a total of one thousand times per data set.

The second lot of tests was on the optimized quick sort. This uses a cutoff value which when reached will switch down to an insertion sort. Insertion sorts are quicker at sorting smaller items. Again to make sure that the results are conclusive each of the data sets where run ten times. This time though there needs to be a way to test how long the sort takes depending on the cutoff value. The way that this is done is by incrementing the cutoff value in steps of five. Starting from five up to five hundred. For each of these values the sort was run ten times and the average value was stored.

The results from these tests can then be used to work out which sort is quicker. They should also be able to show the optimum cutoff value for the sort, weather having such a high cutoff value could decrease performance or increase. By using different sized data sets it will be possible to determine if the cutoff value will change the performance of the sorts. Another test was to sort all ready sorted data sets of the same size as the unsorted.

The two alternative sorts that where used for the cutoff sort, are bubble sort and insertion sort. These are both good at

Table 1: Data sets and their sizes

Data set	*Number of items*
test3	1000
test3a	2000
test3b	5000
test4	10000
test4a	20000
test4b	50000
test5	100000
test5a	200000
test5b	500000
test6	1000000
test6a	2000000
test6b	5000000

sorting a smaller number of items than quick sort is.

The names and number of items used in generation of the results are shown in Table 1.

3. RESULTS

3.1 How long does it take for each type of sort to run

The results for the basic Quick sort show that on average, it takes the same amount of time to complete the data sets. This is expected as there is no change in the sorting algorithm.

For all the results, the length of time that was averaged for data sets 3, 3a and 3b, were returned ether zero or one. Due to these results, test 3, 3a and 3b will not be used in the remaining of this report, as this data is of no use. Figures 1, 2, and 3 show the results for both the basic quick sort and the optimized quick sort. From these results you can see clearly the average length of the sort is consistent.

The optimized sort, uses a cutoff point which it then switched down to an insertion sort, see below. This is designed to allow for improved performance, over a standard quick sort. Which in theory should slow down when attempting to sort a small number of items. Figures 1, 2, and 3 show the speeds at which the optimized sort runs at.

```
for(int i = low; i < high; i++) {
  int j = i;
  Comparable tmp = this.items[i];
  while((j>0)&&(this.items[j-1].compareTo(tmp)>0)){
    this.items[j] = this.items[j-1];
    j--;
  }
  this.items[j] = tmp;
}
```

These results were all based on data sets which were unsorted data. Which means that quick sort should be able to complete using O(n log n). In some implementations of quick sort when sorting sorted data it would cause worst case performance O(n2), if using the leftmost element of the partition as the pivot. But by choosing the median of the items

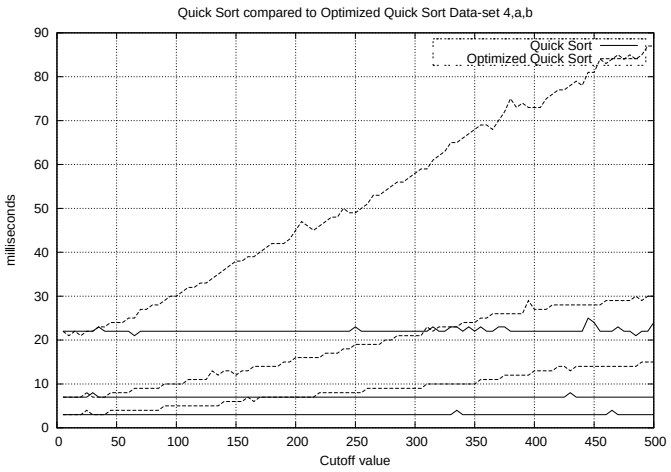

Figure 1: Quick Sort and Optimized Quick Sort results for data set 4, 4a and 4b

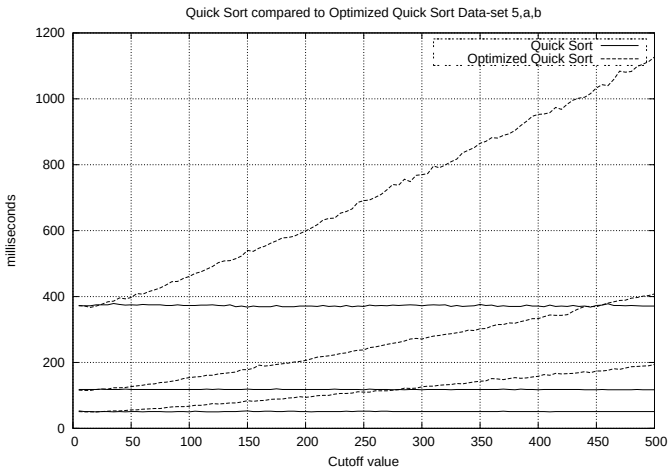

Figure 2: Quick Sort and Optimized Quick Sort results for data set 5, 5a and 5b

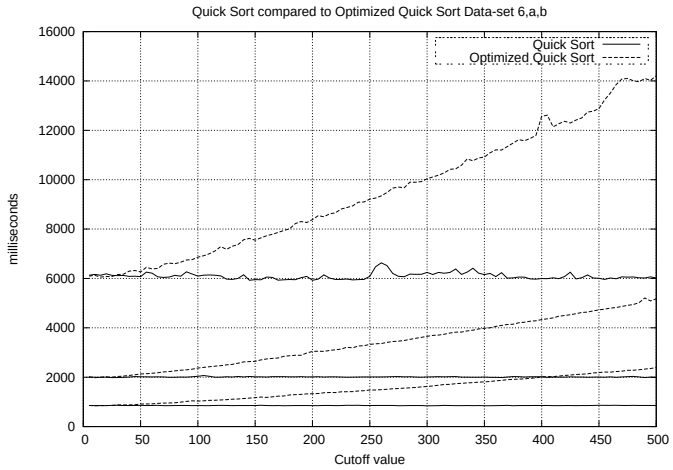

Figure 3: Quick Sort and Optimized Quick Sort results for data set 6, 6a and 6b

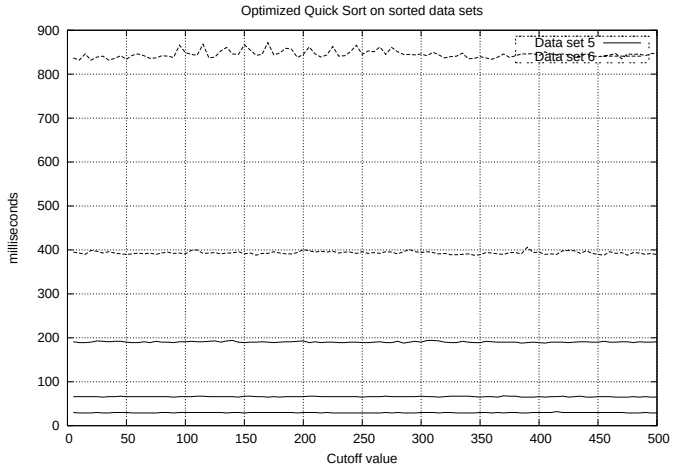

Figure 4: Data-sets 5, 5a, 5b and 6 with sorted data

as the pivot should prevent this. The results of a sorted data sets are shown in figure 4. This shows that the basic quick sort is able to complete sorted data much quicker than unsorted data.

3.2 What are the differences in sorting times between the different variations

There are two variations of the optimized quick sort. The version that uses the insertion sort and one that uses bubble sort. See figure 5 for a comparison of the two implementations. Due to the length of time that the bubble sort implementation took, the data sets shown there are 3, 3a and 3b for bubble sort and 4, 4a and 4b for insertion sort. It is clear that the insertion sort is quicker than the bubble sort. When it is run on data set 3 and 3a the average result is 10ms and for 3b bubble sort took on average 300ms, compared to the 0ms the insertion sort took. The length of time required does not increases along with the cutoff value,

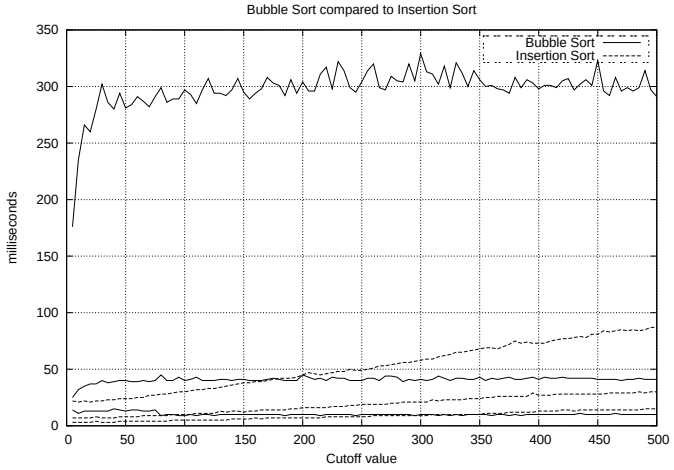

Figure 5: Optimized Quick Sort BubbleSort compared to InsertionSort

like with insertion sort. The maximum length of time that insertion sort took was around 90ms, and this was for data sets 4, 4a and 4b which are much larger than 3 that bubble sort was using.

Figures 1, 2, and 3 show the variations in the speed of the two different sorts. From this data you can see that in almost all cases the length it takes to complete to sort is doubled, by the time the cutoff reaches 500, compared to the basic quick sort. In some cases the optimized quick sort is faster, this depends of the value of the cutoff. More on this in the next section.

3.3 Where do the break points occur

The break points seam to be occurring just after the cutoff value reaches the eighty mark. From there on the length of time that it takes to complete the sort is increased dramatically. This seams to be the break point for the insertion sort. This is due to the fact that when it starts to reach its worst case space, O(n) total, it starts to slow down. When it is given an optimum value of around 20, more on this later. You can see this clearly in all the results figure 6.

The break point for the bubble sort variation are much lower than that of insertion sort, figure 5 shows this. When bubble sort is given a cutoff value of around 10, it starts to take much longer.

3.4 When are simple sorts faster

Figure 6 shows the points where the optimized quick sort becomes faster than the basic quick sort, with the help of simple sorts. This is clear in nearly all the results. When it is given a cutoff value of around 20 the the insertion sort is able to complete the sort quicker than with out. The same can not be said for bubble sort, as it slows the sort down too much even with a low cutoff value.

3.5 Does the size of the data-set make a difference

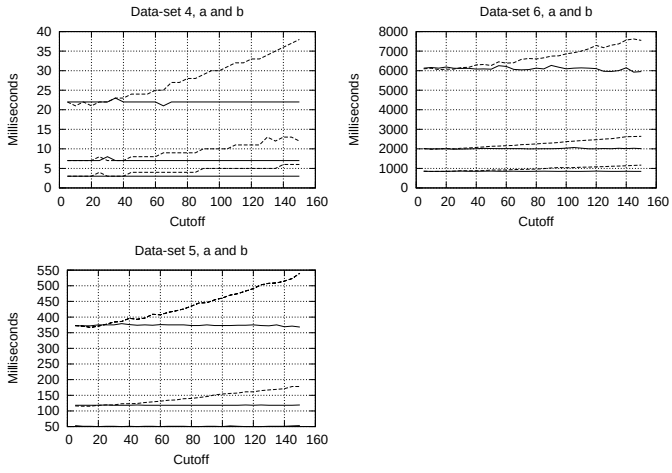

Figure 6: Break points showing with a cutoff value of 150

The results show that it does not make a difference in performance of the sort depending on the size of the data set. You can see that figures 1, 2, 3 and 6 show that the optimized quick sort is faster when the cutoff value reaches the 20 mark. This is true for all of the data sets, which implies that no matter the size of data to be sorted if the cutoff value is around 20 then it will be the optimum value.

3.6 Which is the best general purpose sort

The best general purpose sort it the optimized sort with insertion sort set with a cutoff point to around 20. As this will be able to effectively sort data of any size with the best performance. Where as if it used bubble sort it would not be as quick and will be effected by the size of the data set.

4. CONCLUSION

From the results we can see that the quickest alternative sort is the insertion sort. This is much quicker than bubble sort. Standard quick sort and the optimized sort run at more or less the same speed up until the cutoff value reaches higher than 20, this is where the optimized sort starts to incrementally take longer as the cutoff value is increased. We also found that if the sort is run on already sorted data it takes less time to sort the data then unsorted data.

There is a big difference in time between the insertion sort and bubble sort variation. As the bubble sort took 300ms for data sets which the insertion sort completed in 0ms. This is a clear indication that bubble sort is not a good alternative to switch to. Break points occur when the amount of time that the optimized sort takes to complete is much greater than basic quick sort. In all most all cases this happens around the cutoff mark of eighty. This is with the insertion sort, but with bubble sort this is found to be much lower, five to ten. This makes bubble sort redundant due to the basic quick sort taking less time that is.

The optimum cutoff point to switch down to an alternative sort is around the twenty mark. If the cutoff value is to increase to high, it starts to impact the performance of the sort, due to the fact that the alternative sorts are unable to effectively sort the number of items. This is the same for all data sets, it makes no difference if there are many items or not. The best general purpose sort is quick sort using insertion sort, with a cut of value of twenty.

5. EXECUTIVE SUMMARY

The quickest type of alternative sort was found to be the insertion sort. This worked much quicker than the bubble sort on the same data sets. This is recommend for the optimum quick sort. It was also found that the cutoff value of twenty provides the fastest sort, if the value is incremented then it takes much longer for the sort to complete. The break point happens when there is no performance increase from the cutoff value, this tends to happen around the one hundred mark. The size of the data sets do not make any difference to the cutoff value, it will sort fastest with a cutoff value of twenty irrelevant to the size of the data sets. The best general purpose sort is quick sort with insertion sort set to run when it gets to twenty items.