Jacob Duncan

COSC 370

2/13/2020

README

1. What is the theoretical time complexity of your sorting algorithms (best and worst case), in terms of the array size?

Best Case Worst Case

Quick sort Ω(N log N) O(N^2)

Merge sort Ω(N log N) O(N log N)

1. How does the absolute timing scale with the number of elements in the array? The size of the elements? Can you use the data collected to rectify this with the theoretical time complexity?

The absolute timing for Quicksort can be seen as a positive correlation, as more

elements are in the array, the time it takes to sort is therefore longer. However, with Merge sort, it is hard to identify if the absolute timing is changed based on the number of elements in the array as it is an extremely fast sort. At large amounts of elements in the array is when you can see the absolute timing begin to have a higher standard deviation. For me, it was hard to identify if the size of the elements would matter, if a bigger integer value would take longer than a smaller integer value, however I am sure that if I did a smaller, more narrow test of the algorithms I could determine if it had a difference.

I can use the data I collected to justify the theoretical time complexity if I analyzed the size of the array along with the time it took to complete the sort.

1. How do the algorithms perform in different cases? What is the best and worst case of your own test results?

As seen in the graphs attached in the zip file, the trend in Quicksort shows that with a

backward sorted array, the absolute timing is extremely higher than that of an array with duplicate elements or a presorted array. In Merge sort, the data is tighter together as the algorithm is more efficient in its best and worst case. Although a random array takes the longest time to sort, it is trailed by duplicates. What is different about Merge sort is that backward sorted arrays and presorted arrays take nearly the same amount of time to sort.

The best and worst cases in my own test results match up pretty closely to those of the theoretical time complexities.

1. Do your observations confirm the difference in the best and worst case for Quicksort? How does Merge sort handle these cases?

As I found that in Merge sort the presorted and backwards sorted array are similar in

time it takes to sort, we can know that the differences in the best and worst case for Quicksort and Merge sort are different, just by looking at the graph. Not only is Merge sort taking 1/100th of the time Quicksort takes, it is very different in handling the different types of arrays I threw at it.

1. How do these algorithms compare to your implementation of Bubble sort from Lab1?

These two sorts are much more efficient than Bubble sort in Lab1. Bubble sort was

extremely inefficient when it came to large size arrays however, Quicksort and Merge sort handle large size array with ease as they take a different approach and have a different time complexity, so it only makes sense that they take a shorter amount of time.

1. How could the code be improved in terms of usability, efficiency, and robustness?

I could have made my code more efficient if I were to have used a different type of

partition function and merge function for each of the sorts to make the pivot in a more efficient place, therefore making the sorts run even faster.