Divide and Conquer Analysis

Cody L. Strange, Jayden L. Albrecht

Utah Valley University

Divide and Conquer Analysis

This paper provides an in-depth analysis of two popular sorting algorithms Merge Sort and Quick Sort by comparing which algorithm is faster and more efficient, we will be using four case studies as evidence. Each of these case studies will be tested using both sorting algorithms, and with three different list sizes of size 210, 215, and 220. Each case study will resemble a real-life scenario and have differing distributions in their lists. We will run each test case ten times per list size and record the time it takes for each sorting algorithm to complete and record the average of the ten run times to determine which sorting algorithm is faster. We will also be recording the total number of basic operations – number of comparisons – required to complete the sorting algorithms and compare those to determine which sorting algorithm is more efficient.

# Elo Simulation

## Summary

The first case study will be sorting a list of players from an arbitrary competitive video game by their ELO. Where ELO represents their skill in said game.

### Distribution

The list will be split into four different categories each with a lower bound and an upper bound, low(1000-2400), mid(2401-2700), high(2701-2900), and pro(2901-3000). Where 50% of the players are in the low category, 30% are in the mid, 15% are in the high, and 5% are in the pro.

#### Findings

Beginning with input size 210 we found that quick sort beat merge sort by an average of approximately 2. 6.5\*10-4 seconds. For input size 215, we found that quick sort once again beat merge sort by an average of approximately 2.34 \*10-2. For input size 220, we found that merge sort beat quick sort by an average of approximately 4.4105 \*101 seconds. What this tells us is that for this level distribution quick sort is negligibly faster at small list sizes while merge sort is significantly faster at larger list sizes.

##### Data.

Merge Sort(210)

**Text

Description automatically generated**

Merge Sort(215)

**Text

Description automatically generated**

Merge Sort(220)

**Text

Description automatically generated**

Quick Sort(210)

**Text

Description automatically generated**

Quick Sort(215)

Text

Description automatically generated

Quick Sort(220)

**Text

Description automatically generated**

# Manufacturer Simulation

## Summary

The second case study will be sorting a list of manufacturer employees from some arbitrary company by their pay. The employee’s pay can vary based on overtime, bonuses, and position.

### Distribution

The list will be split into two different categories each with a lower bound and an upper bound, basic worker($30,000-$40,000), manager($60,000-$80,000). Where 90% of the employees are basic workers and 10% are managers.

#### Findings

Beginning with input size 210 we found that merge sort beat quick sort by an average of approximately 3.02\*10-3 seconds. For input size 215, we found that merge sort beat quick sort by an average of approximately 4.028 \*10-2. For input size 220, we found that quick sort beat merge sort by an average of approximately 2.44844 \*100 seconds. What this tells us is that for this level distribution merge sort is faster at small list sizes while merge sort is faster at larger list sizes; however, the difference is not large enough to be of much significance.

##### Data.

Merge Sort(210)

**Text

Description automatically generated**

Merge Sort(215)

**Text

Description automatically generated**

Merge Sort(220)

**Text

Description automatically generated**

Quick Sort(210)

**Text

Description automatically generated**

Quick Sort(215)

Text

Description automatically generated

Quick Sort(220)

Text

Description automatically generated

# Highschool Simulation

## Summary

The third case study will be sorting a list of high school students from some arbitrary school by their age.

### Distribution

The list will be split into four different categories where each category is an age group. 15yr old, 16yr old, 17yr old, 18yr old. Each category will contain 25% of the students in the list.

#### Findings

Beginning with input size 210 we found that merge sort beat quick sort by an average of approximately 3.5581\*10-1 seconds. For input size 215, we found that merge sort beat quick sort by an average of approximately 1.721 \*101. For input size 220, we were unable to do the comparison as quick sort never completed within a reasonable amount of time. What this tells us is that for this level distribution merge sort is always faster and while this is negligible at very small lists on larger lists the difference is so large that quick sort is not viable as a sorting algorithm.

##### Data.

Merge Sort(210)

**Text

Description automatically generated**

Merge Sort(215)

**Text

Description automatically generated**

Merge Sort(220)

**Text

Description automatically generated**

Quick Sort(210)

**Text

Description automatically generated**

Quick Sort(215)

Text

Description automatically generated with medium confidence

Quick Sort(220)

N/A

# Workorder Employee Simulation

## Summary

The fourth case study will be sorting a list of service work orders by the employee that created the work order.

### Distribution

The list will have an even distribution so that 100% of the work orders will be between the numbers 00000000 – 99999999.

#### Findings

Beginning with input size 210 we found that quick sort beat merge sort by an average of approximately 3.0\*10-4 seconds. For input size 215, we found that quick sort beat merge sort by an average of approximately 5.59 \*10-2. For input size 220, we found that quick sort beat quick sort by an average of approximately 1.9072 \*100 seconds. What this tells us is that for this level distribution quick sort is always faster but not by a significant difference.

##### Data.

Merge Sort(210)

**Text

Description automatically generated**

Merge Sort(215)

**Text

Description automatically generated**

Merge Sort(220)

**Text

Description automatically generated**

Quick Sort(210)

**Text

Description automatically generated**

Quick Sort(215)

Text

Description automatically generated

Quick Sort(220)

Text

Description automatically generated

**Conclusion**

## Summary

Having compared both merge sort and quick sort throughout our four case studies we have gathered information on the speeds of the two sorting algorithms. We have determined that the distribution of data can have a significant impact on the time it takes an algorithm to sort. This is especially relevant to quick sort in that large amounts of data that has relatively low distribution can cause the algorithm to become virtually unviable. In our case studies merge sort has shown to be more consistent between all distributions and when it is slower than quick sort it is by an relatively insignificant amount. This leads us to believe that merge sort is the superior sorting algorithm and should be opted for in most general scenarios where a sorting algorithm is needed for large amounts of data. However, in the instances that every second is invaluable, then quick sort would be preferred if the distribution of data is uniform or at least large enough that the number of duplicates is extremely low.

### Future Research

We acknowledge that how study was done with a limited amount of information and has some inherent flaws. Such as the use of random numbers rather than a sample from the real world. Only comparing list sizes of 210, 215, and 220 as well as only having lists that were not uniform and allowed for duplicates which would inherently favor merge sort. We also acknowledge that while the study began with the intent to investigate the basic operations counts of each sorting algorithm and how that it might contribute to the efficiency of the algorithm, with our limited research we found that quick sort always had more basic operations than merge sort no matter which was faster and there was not any relevant information to be found. However future research on that subject and the topics previously mentioned could prove this to be false and allow for a more accurate conclusion.