Introduction

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 theses case studies will be tested using both sorting algorithms, and with three different list sizes of size 2^{10}, 2^{15}, and 2^{20}. 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.

Results

Data

Merge Sort(2^{10})

Merge Sort(2^{15})

Merge Sort(2^{20})

Quick Sort(2^{10})

Quick Sort(2^{15})

Quick Sort(2^{20})

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.

Results

Data

Merge Sort(2^{10})

Merge Sort(2^{15})

Merge Sort(2^{20})

Quick Sort(2^{10})

Quick Sort(2^{15})

Quick Sort(2^{20})

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.

Results

Data

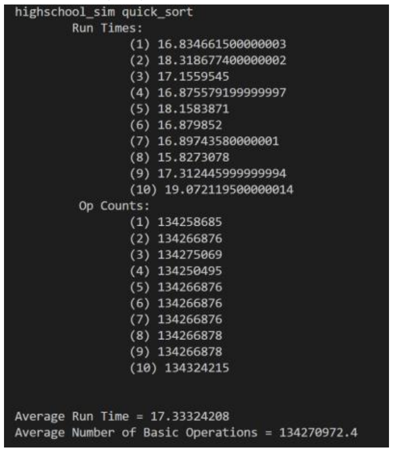
Merge Sort(2^{10})

Merge Sort(2^{15})

Merge Sort(2^{20})

Quick Sort(2^{10})

Quick Sort(2^{15})



```
highschool_sim quick_sort
        Run Times:
                (1) 16.834661500000003
                (2) 18.318677400000002
                (3) 17.1559545
                (4) 16.875579199999997
                (5) 18.1583871
                (6) 16.879852
                (7) 16.897435800000001
                (8) 15.8273078
                (9) 17.312445999999994
                (10) 19.072119500000014
         Op Counts:
                (1) 134258685
                (2) 134266876
                (3) 134275069
                (4) 134250495
                (5) 134266876
                (6) 134266876
                (7) 134266876
                (8) 134266878
                (9) 134266878
                (10) 134324215


Average Run Time = 17.33324208
Average Number of Basic Operations = 134270972.4
```

Quick Sort(2^{20})

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.

Results

Data

Merge Sort(2^{10})

Merge Sort(2^{15})

Merge Sort(2^{20})

Quick Sort(2^{10})

Quick Sort(2^{15})

Quick Sort(2^{20})

Conclusion

Reiterate some introduction information
Reiterate some results information
Explain what further research may need to be done