Bi Nguyen
913875445
ECS 36C
Homework #1

Report

Preliminary

In the following report, we will be looking at two searching algorithm, linear and binary search to analyze the pros and cons of each method. We will be searching through a vector of class objects called Books that stores **isbn, language,** and **type** that describes the newly delivered books given the set of requested books**.** Before we get into the analysis, let's discuss about the idea of linear search and binary search. The basis of linear search is to look for the item relative to its current position until its found. On another hand, binary search requires a sorted list in order to perform its operation. This is one of the tradeoff that we will later discuss in the analysis. The idea of binary search is to begin searching at the middle of the list, and depending on the value of the item or object, it then search either on the left or right side of the list by comparing the value of the object to its value on the left or right. This process is repeated after the algorithm decides which direction to take.

Analysis

In the analysis, we will generate multiple small and large datasets to test out the algorithm. We will analyze the time complexity of the algorithm by averaging the runtime of each data. We hope that by doing this analysis will possibly enable us to determine the ratio between the number of new books and the number of requests to search. We will use different data set for each run but use the same data set for both algorithm to compare.

Linear Search

We will start off by generating 5000 new books and 500 requested books

1st Run

```
nguyenbi@ad3.ucdavis.edu@pc23:~/ECS36C/Homework_1$ create_testData 5000 500
 english
nguyenbi@ad3.ucdavis.edu@pc23:~/ECS36C/Homework_1$ ./SearchNewBooks newbooks.dat request.dat output.dat
Choice of search method ([l]inear, [b]inary)?
1
CPU time: 33830.4 microseconds
```

2nd Run (Different dataset; same input)

```
CPU time: 30105.6 microseconds
```

3rd Run
CPU time: 32550.4 microseconds

AVG time: 32162.1 microseconds

Next we will generate 20,000 new books and 2000 requested books

1st Run
CPU time: 375894 microseconds

2nd Run
CPU time: 369869 microseconds

3rd Run
CPU time: 365104 microseconds

AVG time: 370289 microseconds

Lastly, we will generate 80,000 new books and 8,000 request books

1st Run
CPU time: 5.88156e+06

2nd Run
CPU time: 5.8778e+06 microseconds

3rd Run
CPU time: 5.86414e+06 microseconds

AVG time: 5.87783e+06 microseconds

Binary Search

5000 new books and 500 requested books

1st Run
CPU time: 580637 microseconds

2nd Run
CPU time: 583596 microseconds

3rd Run

CPU time: 579412 microseconds

AVG time: 581215 microseconds

20,000 new books and 2000 requested books

1st Run
CPU time: 1.0057e+07 microseconds

2nd Run
CPU time: 1.0003e+07 microseconds

3rd Run
CPU time: 1.00527e+07 microseconds

AVG time: 1.00466e+07 microseconds

80,000 new books and 8,000 request books

1st Run
CPU time: 1.74707e+08 microseconds

2nd Run
CPU time: 1.74784e+08 microseconds

3rd Run
CPU time: 1.74604e+08 microseconds

AVG time: 1.746983e+08 microseconds

I chose a multipler of 4 when increasing the size of my data sets. I expected the result to yield at a slower rate (4x) but rather it was at 8x slower using linear search switching from 5,000 new books to 20,000 new books and 6x slower from switching from 20,000 to 80,000 books. While for binary search it, it was 5x slower switching from the size of the data from 5,000 to 20,000 and 20,000 to 80,000. As it seems that binary search stays at a consistent rate compared to linear search when we increase the sizes of each data set.

Conclusion

From the above average time that linear search is significantly faster than binary search. The reason is because we have to use a search function (quicksort) in order to first sort the unorder vector. This function takes O(nlog(n)) times. Along with the time complexity of the quicksort algorithm, the binary search takes O(logn) times, and in total the entire run time for binary search worst case scenario is nlog(n^2) which is slower than a linear search whose run time is O(n). Note that there are trade off using linear search and binary search. If a collection of items are unorder, then will result in faster computation time while if a collection is ordered, a binary search will be faster. If both data sets are relatively smaller than 100, I think that it's possible for both algorithm to run at similar rate. However, I think it's difficult to find a ratio between newbooks and request books because that would mean we have to permute all possible combination available in order to explain which algorithm is better.