IT6033 Data Structures and Algorithms
Victoria Livingstone
92018976

Practical Task 1:

1.4 Analyze the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of array list to complete the task?

Answer:
An arraylist was the most appropriate data structure. It allowed for the removal of data, and in future the addition of other patient files. Another option for this would have been an Array, however that would have required the specification of the number of items within the Array, and if any items were removed or added the number would have to have been changed.

2.7 Analyze the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of stack to complete the task?

Answer:
The stack structure was most appropriate for the task as it was only required the program to keep track of how many instances of the opening bracket occurred. Another data structure that could have been used would be an arraylist. This could have been done by incrementing and decrementing the index pointer to keep track of how many instances of brackets had occurred. However this is more complex, and in this case unnecessary.

3.5 Analyze the data structure used in the task. Was the data structure fit for the task? Explain your answer. Is there another data structure that could be used in place of queue to complete the task?

Answer:
Another data structure that could have been used instead of a queue for this task would have been an arraylist. This would have involved removing each item from the front of the arraylist as it was processed. However this would have continued to increase the length of the Arraylist indefinitely. Therefore, a queue was most appropriate, as it did not extend beyond the index length currently in use.

Practical Task 2:

2.6 Use your counter or timer output to explain the differences between the performance of the first and second sort operations Hint: Depending on what data you choose, you will be using either a best case or worst case data set. Make sure that you comment on this.

Answer:
According to the implemented timer the first sort took 2 Milliseconds. The second sort, which utilized the best-case scenario, as all the data was already sorted only took 1 millisecond. This indicated that the quicksort program is at least twice as quick to sort data in the best-case scenario. I theorize that if the data had been in the reverse order (The worst-case scenario) that the time taken would have been more comparable, if not longer than the initial sort.

3.5 include some text that explains which algorithm is more efficient, the Java built in sort, or the bubble sort provided in the previous task. Your response should be at least a paragraph long and should include technical terms found in both the course, and those taken from your own research online. These might include references to algorithm names, big O notation, best case, average case and worst-case complexity.

Answer:
The built in Java sort is by far the more efficient and effective sorting method. Both sorting methods have big O notation of O(n^2) , however bubble sort requires, on average, n/4 swaps per entry , while Selection sort requires only 1. This means that for small data number the difference in number of swaps, and computational time is less noticeable, but with larger datasets, and the worst-case scenario, this can mean over 100x more comparisons for bubble sort vs. java sort.

Practical Task 3: Algorithm Analysis

1.1: Analyze problem and identify two possible algorithmic solutions, one using a sequential search and one using a binary search. Describe each solution in words and pseudo-code and discuss their appropriateness to solve the problem.

Answer:
Sequential search solution:
One solution would be to use a sequential search of the data to find the result. This would involve taking the name of the movie that you wanted to search for and comparing it to each movie title within the provided dataset. This has both advantages and disadvantages. If the data is unsorted then is has an advantage over other searching methods that is doesn't require sorting first. However as it has Big O notation of O(n) it becomes far more cumbersome and impractical when he datasets become larger.
A sequential search would involve the following pseudo code:

1) Read array side, get target from user
2) i= 0
3) While i< size of Array
4) If target == array[i]
5) Print , then break loop
6) Else i++

In this specific scenario, a linear search would not be the most efficient searching method. As the dataset is liable to be already sorted it would be more efficient to use a Binary search as it involves less computational time and comparisons.

Binary search solution:
A Binary search is usually viewed as the more appropriate searching method, especially with larger datasets. Although it does require the dataset to be sorted prior to a search, it handles large datasets much more efficiently than a sequential search.
A Binary search would involve the following pseudo code:

1) Let min = 0 and max = n-1.
2) Guess = average of max and min, rounded down (so that it is an integer).
3) If array[guess] == target, Break. Return target.
4) If array[guess] < target, then set min = guess + 1.
5) Otherwise, the guess was too high. Set max = guess - 1
6) Go back to step 2.

In this specific scenario, a Binary search would be the best solution. The dataset would already be sorted and therefore a binary search would be able to find the target with the least amount of comparisons and computational time.

1.2: Analyze the identified algorithms and predict their performance, using asymptotic analysis and Big O notation. Explain your reasoning.

Answer:
While both algorithms are not entirely considered impractical and cumbersome, there is definitely a more efficient method when comparing them. A linear search has big O notation of O(n), which is considered a fair complexity level regarding calculating computational time and space. It has a linear relationship, meaning that the more data that needs to be processed, the more computation time and space will be needed. This makes it practical for small datasets, however once the datasets begin to get bigger, it becomes too cumbersome to function effectively.

A Binary search has big O notation of O(log(n)). This means that the relationship between the number of operations completed and computational time/space is represented by a log graph. This shows that as the quantity of data becomes significantly larger, a binary search will process it with much more efficiency than a linear search.

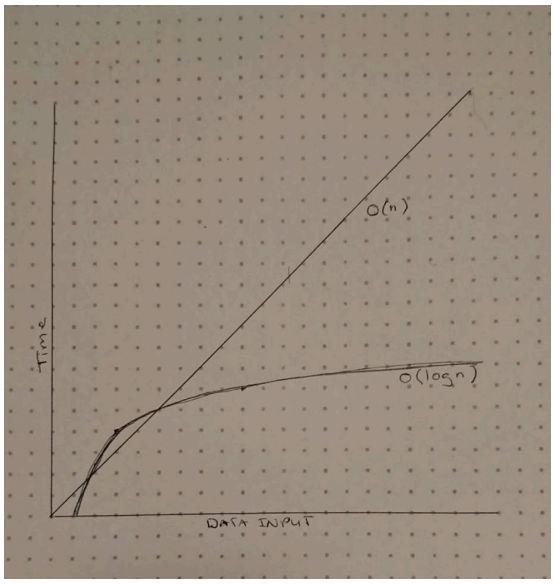
Therefore I predict that with the small and medium datasets, there will not be a huge difference in computational time. However, as the quantity of data to be processed increases the binary search will show to be the far faster and more efficient method of sorting. I also predict that the ratio between the methods, regarding the number of steps taken to complete the search will become greater with the increase of data size. This is due the binary search has a more efficient methods to look through the data for the target.

1.3: Predict the performance of the identified algorithms for small, medium, large and very large data sets (10, 100, 1000, 10 000 or more items in the input set). Present this in a Predictions table that you can use for verification purposes in the question 3.

Note: Assumptions are made for the worst-case scenario	Time Taken		Steps Taken	
	Linear Search	**Binary Search**	**Linear Search**	**Binary Search**
Small	Approximately the same		Target index -1	$Log_2(n) + 1$
Medium	Approximately the same/ Binary may be slightly faster		Target index -1	$Log_2(n) + 1$
Large	Binary will be noticeably faster		Target index -1	$Log_2(n) + 1$
Extra Large	Binary will be significantly faster		Target index -1	$Log_2(n) + 1$

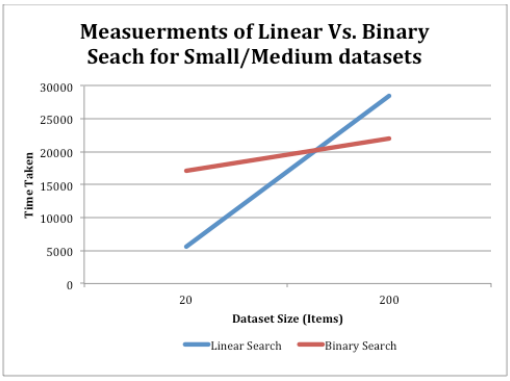
1.4: Provide a simple graph that shows the difference in performance of the two algorithms (a photo/scan of a hand drawn graph is sufficient.)

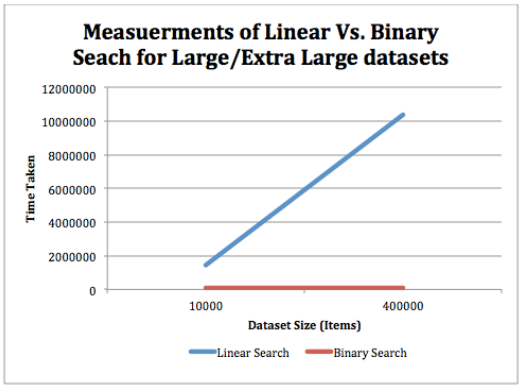
Answer: The difference in performance between the two algorithms is dependent on the Big O notation. In general, for smaller datasets, there wont be a huge difference in computation time. However, as the datasets get larger, the difference in efficiency and performance becomes far more apparent.



3.3 Measurements Data

Note: Assumptions are made for the worst-case scenario	Time Taken (Milliseconds)		Steps Taken	
	Linear Search	Binary Search	Linear Search	Binary Search
Small (20)	5601	17088	20	3
Medium (200)	28507	22076	187	7
Large (10K)	1446405	70156	10546	13
Extra Large (400K)	10385844	62964	390111	18





Note: Putting all data on one Graph skewed the appearance of the trends; therefore data has been put on two appropriate graphs.

3.4 Compare your measurements with your predictions and comment on the outcome, whether the predictions are verified by the measurements or not. Discuss any factors that may have affected the measurements or the predictions themselves.

Answer:
After running the code with different sized data sets, I saw that the predictions I had made previously we mostly accurate. Some of the differences from my predictions were the computational time taken for the small data set. I had predicted that they would both take the same time, however I was wrong as the linear search was only one third of the speed of the Binary.

Once the data sets became larger it was more apparent which algorithm was more efficient, with the linear search taking over 100x more computational time.

With regards to steps taken for computation, the binary search was more efficient throughout all outcomes. As this was the worst scenario, both methods would have only gotten more efficient if the target item was closer to the starts of the dataset. Therefore it is obvious that regardless of the size of the dataset, a Binary search is the most efficient algorithm to use for searching.

It must be noted that these results are only applicable to datasets that are already sorted and if the provided data had been unsorted or had involved duplicates, the results would have changed.