**Big ‘O’ and Searching**

**Knowledge**

1. **What is Big ’O’ notation, how does it help us compare the efficiency of algorithms?**

Big-O is the relative complexity of an algorithm by reducing the growth rate to a single factor. With respect to its size, Big-O gives you how quickly the runtime grows. Since the exact runtime of an algorithm depends on other factors such as the speed of the processor, Big-O talks about how quickly the runtime grows. Big-O helps us compare the efficiency of algorithms because it essentially answers what happens to the performance of an algorithm as the complexity of data increases.

1. **How does the binary search work?**

Firstly, binary search uses a sorted array. Binary search compares the middle term with the data behind searched for. If the data being searched for comes after the middle term, continue the search with the last half. On the other hand, if the data being searched for comes before the middle term, continue the search with the first half. The loop will continue until the data is found or the entire array has been searched.

1. **Which search is better when considering best-case?**

Linear search is better is better than binary search when considering the best case. The best case for linear is that the item you are looking for is the first item. The best case for binary is that the item is in the middle. In both cases, you only need 1 iteration of the loop. Therefore the decided factor becomes which search is better when running 1 iteration. Since running a linear search once is faster than running a binary search once, linear search is better.

1. **Which search is better when considering worst-case?**

Binary search is better than linear search when considering the worst case. The worst case for linear search of n items is that it takes n iterations to find the data (it is the last item or doesn't exist). The worst case for binary search is that the number of iterations takes essentially log2n. Therefore, binary is more efficient in its worst case than linear is in its worst case.

1. **Give a mathematical analysis that compares the search times between the two techniques.**

Let's say it takes a computer's processor to execute one loop of a linear search algorithm 5 microseconds and one loop of binary search algorithm 6 microseconds.

Best Case (Question 3 Proof):

* The best case for a linear search is O(1) comparisons. Therefore it would take 5 microseconds.
* The best case for a binary search is O(1) comparisons. Therefore it would take 6 microseconds.
* Since linear search takes less time, this proves that linear search is better than binary search when considering the best case

Worst Case (Let's say n=1000) (Question 4 Proof):

* The worst case for a linear search is O(n). Therefore it would take .
* The worst case for binary search is O(log n) comparisons. Therefore it would take .
* Since binary search takes less time, this proves that binary search is better than linear search when considering the best case

**Thinking [/8]**

1. **Predict the average search times for both binary and sequential searches given the following scenarios. Assume that the time taken for the computer’s processor to execute one loop of the sequential search algorithm is 5 microseconds, and the time taken of a loop of the binary search is 6 microseconds. [nb: if you don’t have a log2n function on your calculator, you can use a standard log10n by typing (log10n / log102). To verify, log21024 = 10.]**
   1. **Find a name within a list of 10,000 names**
   2. **Find a name within a list of 10 names**
2. **From the analysis above, when does it make sense to use the sequential search? When does it make sense to use a binary search?**

The analysis above shows that binary search is more effective in both situations as it faster than linear search. One reason to use sequential search is when looking through data that is unsorted because binary search requires the data to be sorted. Although linear search is also better than binary when considering the best case, this situation is quite rare most times in reality. Therefore since it is hard to ensure that this will happen, it becomes impractical in relying on the fact that the best case will occur. Another situation where sequential search makes sense is when the data size is small. In the calculation when n=10, although binary was faster, linear search did still have a low runtime. Therefore sequential search is still a good option to use when the data size is small. Sequential search is also viable if you know the item you are searching for is one of the first few items. However this is not practical as this is not always the case and you cannot assume this. For example it would be effective in times when you are searching for a word that begins with the letter "a" in a dictionary as it is near the start. It makes sense to use binary search in most situations as shown by the calculation above. If you know that the data is already sorted, binary search becomes an effective in finding the target even if the data size is very large. Both results above show that binary search was faster in both situations. This shows whether the data size is small or big, binary search is effective in most situations. Also even though binary search is better than linear search in the worst case, this situation is also rare as well. It is also impractically on relying that this situation will occur because it is rare.

1. **In your own words, describe what searching by hashing is. This should be a detailed paragraph.**

Hashing is transforming a string of characters into a shorter value of specific length or a key representing the original string. In a database, hashing is used to index and retrieve items. Instead of searching for the original value, it becomes faster to find the item with the shorter hashed key. For example, searching for a four digit hash code is faster than searching for text with 26 characters. Hashing is also used in many encryption algorithms such as MD5. A hashing algorithm is called a hash function. The hash function is used for two reasons. One is to index the original value or key. In other words a hash function places keys to array indices. The second is to then later retrieve the data associated with key by using the hash function again. Since hashing is a one-way operation, the most ideal hash function should not be derived by analysis of hashed values. Also an effective hash function avoids collisions. This occurs when you get two different inputs for the same hash value. There are many different kinds of hash functions including division-remainder method, folding method, radix transformation method, and digit rearrangement method. The example below is use of the division remainder method. Suppose that a student's information is stored in a database with their student ID as the key field. Let's say there are a hundred or fewer students with ID numbers ranging from 0 to 9999. Let's take the following example of students: 0, 100, 200,...,9800, 9900. If you made an array called students with a 100 elements, you would store each student ID h at students[h/100]. Then for example if you wanted to find the student with the ID 200, you calculate 200/100=2. Therefore the student is located at index 2. The hash function can be written as hash(h)=h/100. In this you can see h is the key (the students ID), hash(h) is the location of the key (also known as the index), and h/100 is the calculation done to get the index. Using this type of search, hashing can be used to find targets in a set of data.

**Application**

1. **In this assignment and the next, you are going to build a helper class that will implement all of the Searching and Sorting algorithms we learn in this class.**
   1. **Create a class called SearchSortHelper**
   2. **Implement both searching algorithms as methods in this class. Ensure you have versions of the methods that work for Strings and integers (at least).**
2. **Confirm your hypothesis in the Thinking questions above, by running your program on the dictionary data file provided. Record the amount of time needed to perform 10000, 100000 and 1000000 searches of the following words:**
   1. **about**

* Linear
  + 100000 searches: 6
  + 1000000 searches: 26
  + 10000000 searches: 246
* Binary
  + 100000 searches: 12
  + 1000000 searches: 102
  + 10000000 searches: 1067
  1. **header**
* Linear
  + 100000 searches: 632
  + 1000000 searches: 6284
  + 10000000 searches: 60819
* Binary
  + 100000 searches: 10
  + 1000000 searches: 100
  + 10000000 searches: 992
  1. **yours**
* Linear
  + 100000 searches: 2196
  + 1000000 searches: 22365
  + 10000000 searches: 219089
* Binary
  + 100000 searches: 6
  + 1000000 searches: 55
  + 10000000 searches: 549

As you can see, linear was faster in question a and binary was faster in question b/c. This matches my hypothesis because the word about is near the start of the dictionary. Therefore the linear search requires fewer comparisons to find the target because it only needs to loop the algorithm a few times. Searching for a word like "about" gets closer to the best case for linear. The fact that b/c where faster for binary also matches my hypothesis because the dictionary had many number of items. Although one iteration of the binary algorithm may have taken longer, it had to do fewer comparisons to find the target. Searching for words like "header" and words even more like "yours" gets closer to the worst case for linear. In these situations binary is better. Therefore, it makes sense that the binary algorithm was faster in these cases.

**Note: You can use System.currentTimeMillis to automate your time tracking. Just subtract finish time from start time.**