UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

Big Oh - Analyzing Scalability of Algorithms

Dr. Mark R. Floryan

September 10, 2019

1 SUMMARY

For this homework, you will work to understand the scalability of algorithms, and the differences between common runtimes. You will also continue practicing your programming by implementing some (but not all) of the methods you will be analyzing.

1. Download the starter code and import the project into Eclipse.
2. Implement the missing methods from BigOh.java
3. Run each method on increasing array sizes. Use completion times to fill out the BigOh chart.
4. **FILES TO DOWNLOAD:** BigOh.zip
5. **FILES TO SUBMIT:** BigOh.zip (includes BigOh.pdf with chart)

1.1 DOWNLOAD STARTER CODE AND IMPLEMENT THREE METHODS

You can download the starter code for this homework from the course repository here. Once you have done so, you should import the project into Eclipse.

This project contains one Java file (BigOh.java). The file contains the following methods (some of which are not yet implemented):

```
// Searches for item in sorted array a.
// Returns true iff item is found in a
// Should run in Theta(logn) time
public static binarySearch(int[] a, int item);

// Finds the largest item in a (a might not be sorted)
// Should run in Theta(n) time
public static int max(int[] a);

// Invokes binarySearch() with a.length random numbers
// Returns how many times those random numbers were found in a
// Should run in Theta(nlogn) time
public static int multipleBinarySearch(int[] a);

// Counts how many pairs of items in a sum to a multiple of 5
// Should run in Theta(n^2) time
public static int allPairs(int[] a);

// Counts how many a,b,c combinations there are in a such that a+b=c
// Should run in Theta(n^3) time
public static int allTriads(int[] a);

// loops through all the subsets of array a
// e..g, {1,2,3} would print {},{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}
// Should run in Theta(2^n) time
public static int allSubsets(int[] a);
```

Your first task is to implement the first five of these methods. The allSubsets() method is provided as it is much more tricky than the others.

Many of you have probably not seen **binary search** before. Binary search is an efficient method of searching through sorted data. Because the data is sorted, we can efficiently search by looking in the middle of the data and reducing our search to the half that is guaranteed to contain the item. We do this continuously until the search narrows all the way down to one item. This is similar to looking up a word efficiently in a dictionary. First, we look in the middle of the book. If our word is in the lower half we open halfway in that direction, and so on until we narrow in on the word we are searching for. Some psuedo-code for binary search can be found below:

```
binarySearch(int[] a, int item){
    set variables min=0 and max=a.length-1
    continue while min is less than max
```

```
4           look at the middle between min and max
            if that element is item, then found!
6           if item is less than middle element
                repeat search between min and cur
8           if item is more than middle element
                repeat search between cur+1 and max
10
        if this loop exits, then the item was not found
12  }
```

The other methods should be self-explanatory from the comments above. Let course staff know if you have any confusion about these. Make sure to write small tests to ensure your code is working correctly. We are not providing tests for this assignment.

1.2 Analyzing Scalability

Your second task is to analyze how large inputs can reasonable get at each runtime. We have provided a main function to you that asks for two inputs: The method you would like to execute, and how large you'd like the input size (array) to be. The method also provides some code that times how long the operation takes and reports the time to you.

Run the code multiple times and fill out the chart below. Submit the chart along with your code as BigOh.pdf. As your filling out this chart, think about how much larger the input can get as your code becomes more and more efficient. If a time takes more than 1 minute, don't wait for the code to finish, just report that the code took more than 1000 ms. Please report all times in ms (a time of 0 ms is fine if the code finishes quickly).

Input Size:	1	10	100	1,000	10,000	10^5	10^6	10^7	10^8
Binary Search (logn)									
Max (n)									
Multi Binary Search (nlogn)									
All Pairs (n^2)									
All Triads (n^3)									
All Subsets (2^n)									