

<p>CSCI 393 - Algorithm Design and Analysis Spring 2008 Homework Number 1</p>

1. (5 points) Let n be the number of atoms in the universe. Find $\text{Log}^*(n)$.
2. (15 points) Implement Algorithm 1.1 (Sequential Search) and Algorithm 1.5 (Binary Search) in Java. Once you have both algorithms working properly, I want you to run them on lists of increasing size and compare their performance. You will need to save the number of iterations and elapsed time required by each algorithm each time it is run. You can use the Java class `StopWatch`¹ (available from the class Web site) to measure the elapsed time. The values used for the input lists should be a randomly generated integers between 1 and $2n/3$. The lengths of the lists should be large enough so the difference in elapsed time is noticeable. For each different value of n , you should run each algorithm on 5 different lists of size n . Once you have gathered the data from your program runs, graph the performance of both algorithms. There should be two graphs, one showing the average number of iterations as a function on n and one showing the elapsed time of each algorithm as a function of n .

For example, say you discover that to show how the algorithms perform as n grows, you need to use lists of 20 different sizes ranging from $n = 1,000,000$ to $n = 20,000,000$. Then you should create five lists of each of these sizes, do all the runs, saving the data (number of iterations and elapsed time) for each run. Then you should construct one chart with n on the x -axis and the average number of iterations for each algorithm for each value of n on the y -axis. You should also construct a second chart with n on the x -axis and the average elapsed time on the y -axis.

Since the binary search algorithm requires the input list to be sorted, in order to make a valid comparison you will need to sort your lists before running the algorithms. Fortunately, the Java `Arrays` class has a `sort` method that you can use. Below is a short Java program demonstrating how to use this method.

¹From "Big Java" 3rd edition, by Cay Horstmann

```
import java.util.Arrays;
public class ArraySortDemo
{
    public static void main(String[] args)
    {
        int[] arrays = {4,3,6,7,4,5,43,8,7,5,4};

        Arrays.sort(arrays);

        for(int i=0; i < arrays.length; i++)
            System.out.println(arrays[i]);
    }
}
```

3. (15 points) Implement Algorithm 1.6 (n^{th} Fibonacci Term (Recursive)) and Algorithm 1.7 (n^{th} Fibonacci Term (Iterative)) in Java. Once again, you are to compare the time complexity of these two algorithms. For this comparison you only need to consider the elapsed time. You should graph the elapsed time as a function of n . Once again, choose values of n that show how the algorithms perform as n grows.