Katherine Wilsdon

Dr. Isaac Griffith

CS 2235 Data Structures and Algorithms

11 September 2019

Part 1 - Algorithm Analysis

(2 Points each) Using the definition of big-oh notation, prove or disprove the following assertions (i.e., you must provide the $c > 0$ and $n_0 \geq 1$ that fulfills the definition, or give a formal argument for why the assertion is false).

1. $2n^3 - 7n^2 + 100n - 36$ is in $O(n^3)$

   $|2n^3 - 7n^2 + 100n - 36| \leq 2n^3 + |7n^2| + 100n + |36|$

   $\leq 2n^3 + 7n^3 + 100n^3 + 36n^3$

   $\leq 145n^3$

   Thus, $2n^3 - 7n^2 + 100n - 36 \leq 145n^3$ for all $n \geq 1$

   So, $2n^3 - 7n^2 + 100n - 36$ is in $O(n^3)$

2. $10n + 3 \log(n)$ is in $O(n)$

   $|10n + 3 \log(n)| \leq 10n + 3 \log(n)$

   $\leq 10n + 3\log(n)n$

   $\leq 13\log(n)n$

   Thus, $10n + 3 \log(n) \leq 13\log(n)n$ for all $n \geq 1$

   So, $10n + 3 \log(n)$ is in $O(n)$

3. n/1000 is in O(1)

   $|n/1000| \leq (1/1000)n$

   $\leq (1/1000)n$

   Thus, $n/1000 \leq (1/1000)n$ for all $n \geq 1$

   The assertion is false; n/1000 is in O(n)

4. $\log(n)^2 + \log(n)/30$ is in $O(\log(n)^2)$

   $|\log(n)^2 + \log(n)/30| \leq \log(n)^2 + (1/30)\log(n)$

   $\leq \log(n)^2 + (1/30)\log(n)^2$

   $\leq (31/30) \log(n)^2$

   Thus, $\log(n)^2 + \log(n)/30 \leq (31/30) \log(n)^2$ for all $n \geq 1$

   So, $\log(n)^2 + \log(n)/30$ is in $O(\log(n)^2)$

5. $n^2/ \log(n) + 3n$ is in $O(n^2)$

   $|n^2/ \log(n) + 3n| \leq (1/\log(n))n^2 + 3n$

   $\leq (1/\log(n))n^2 + 3n^2$

   $\leq (3/\log(n))n^2$

   Thus, $n^2/ \log(n) + 3n \leq (3/\log(n))n^2$ for all $n \geq 1$

   So, $n^2/ \log(n) + 3n$ is in $O(n^2)$

(2 Points each) For each function below, provide the "tightest" big-oh bound. You can do this from the definition of big-oh if you are unsure of the answer, or you can use shortcuts and just provide the big-oh value.

6. 36n is in O(n)

7. $n^2/2 + 15n$ is in $O(n^2)$

8. $(n^2/4)(8/n)$ is in O(n)

9. n + 10 log(n) is in O(n)

10. 87262 is in O(1)

(4 Points each) For each method below, provide the tightest big-oh running time.

```
public int m1FindLargest(int[] array) {
    if (array.length != 0) {
        int value = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > value) {
                value = array[i];
            }
        }
        return value;
    }
    return -1;
}
```

The algorithm is in O(n) (worst case scenario).

```java
public void m2PrintTriangle(int size) {
    for (int i = 1; i <= size; i++) {
        for (int j = 1; j <= 1; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

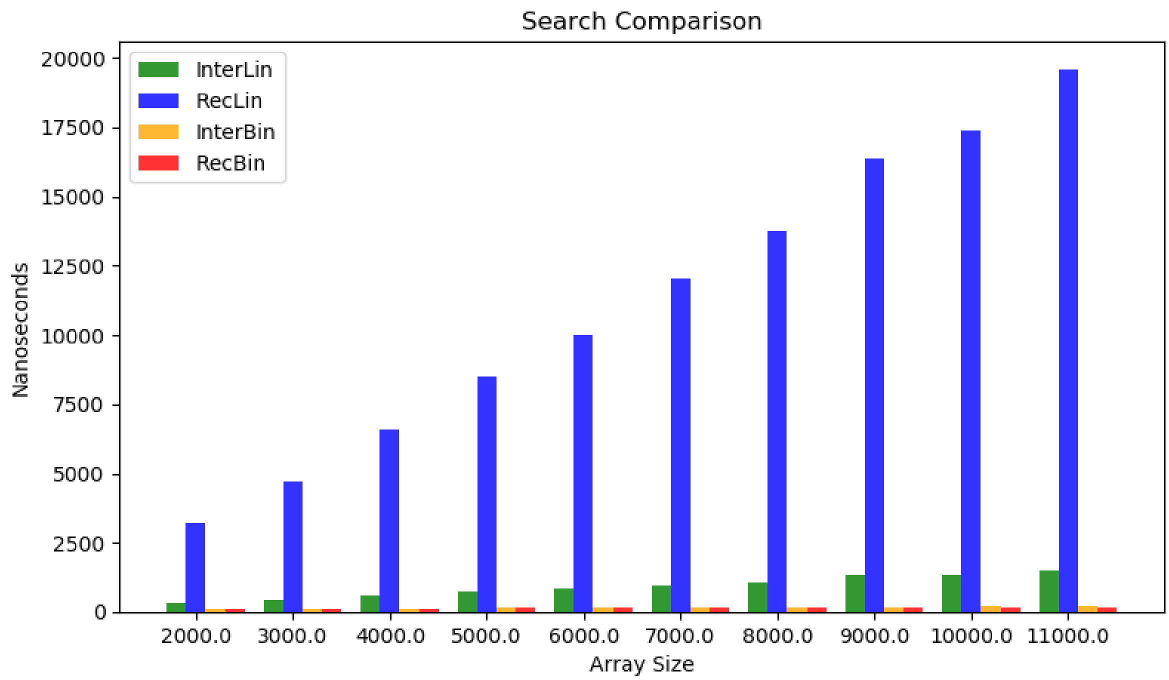
The algorithm is in O(n^2).

```java
public void m3PrintBooks(String books[], int[] stars) {
    if (books.length == stars.length) {
        for (int i = 0; i < books.length; i++) {
            System.out.print(books[i] + "'s stars: ");
            for (int j = 0; j < stars[i]; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

The algorithm is in O(n^2) (worst case scenario).

Part 2 - Binary Search

N, IterLin , RecLin , IterBin , RecBin
2000,321,3199,115,120
3000,435,4722,115,106
4000,573,6570,119,117
5000,726,8531,139,144
6000,831,10002,142,140
7000,949,12027,143,138
8000,1074,13757,164,150
9000,1313,16376,170,174
10000,1329,17414,190,167
11000,1511,19610,186,168



The computation time increases as the array size increases for iterative linear search and recursive linear search. The computation time is relatively constant for iterative binary search and recursive binary search while slightly increasing. Recursive linear search has the greatest computation time. Iterative binary search and recursive binary search have the least computation time. Benchmarking was performed on a 4-core processor with 4.0 GHz core frequency.