**Name: Akiko Iwamizu**

**Uni: AMI2119**

**COMS W3134 Fall 2014  
Homework 1  
Due: 11:55pm on Monday, September 22, 2014**

**Written Section**

1. Weiss, Exercise 2.1

< 37 < < N < N log log N < N log N ≤ N log(N2) < N1.5 < N2 < N2 log N < N3

< 2N/2 < 2N

Where N log N and N log(N2) are functions that grow at the same rate. This can be shown using the definition of logarithms and Big-O Notation:

N log(N2) = 2N log(N) 🡪 O(N log N)

1. Weiss, Exercise 2.6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Days | 1 | 2 | 3… | N |
| $a | $2 | $4 | $16 | ? |
| 2x | 21 | 22 | 24 | ? |
| $b |  |  |  |  |

The x in 2x is equal to x = 2(N-1), thus the fine on day N would be

b) Using the definition of logarithms I can show how many days it will take for the fine to reach D dollars:

= D iff log2 (D) = 2(N-1)

2(N-1) = log2 (D) iff log2(log2 (D)) = N - 1

Thus, N = (log2(log2 (D)) + 1) and the Big-Oh notation would be **O(log (log (D))**

1. Weiss, Exercise 2.7 - note that part b of this problem is actually part of the programming assignment. Place your answers to part a, and part c in the written solutions.
   1. Give an analysis of the running time in Big-Oh notation
      1. The running time (RT) is **O(n)** because the RT of a for loop is at most the RT of the statements inside times the # of iterations. In this case, it would be 1 \* n.
      2. The running time is **O(n2)** because the RT of a statement inside a group of nested loops is the RT of that statement times the product of the sizes of all the loops. In this case, it would be 1 \* n \* n.
      3. The running time is **O(n3)** because of the same logic in (ii) but in this case it would be 1 \* n \* (n\*n).
      4. The running time is **O(n2)** because of the same logic in (ii).
      5. The running time is **O(n5)** because of the same logic in (ii) but in this case it would be 1 \* n \* (n\*n) \* (n\*n) where j is at most (n2).
      6. The running time is **O(n4)** because of the same logic (ii) but the third for loop can only be as large as n because of the if statement, so 1 \* n \* (n\*n) \*n.
   2. Implement code in Java and give actual running times for several values of N

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 1,000 | 0 |  |
| 10,000 | 1 |  |
| 50,000 | 2 | 1 |
| 75,000 | 2 | 0 |
| 100,000 | 2 | 0 |

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 100 | 1 |  |
| 300 | 3 | 2 |
| 600 | 5 | 0.67 |
| 900 | 9 | 0.80 |
| 1,000 | 11 | 0.22 |

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 100 | 8 |  |
| 300 | 213 | 25.6 |
| 600 | 1743 | 7.18 |
| 900 | 5820 | 2.34 |
| 1,000 | 8051 | 0.38 |

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 100 | 0 |  |
| 300 | 2 |  |
| 600 | 3 | 0.50 |
| 900 | 5 | 0.67 |
| 1,000 | 6 | 0.20 |

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 10 | 0 |  |
| 25 | 10 |  |
| 50 | 255 | 24.5 |
| 75 | 2057 | 7.07 |
| 100 | 8507 | 3.14 |

|  |  |  |
| --- | --- | --- |
| Size (N) | Time (ms) | Growth rate |
| 10 | 0 |  |
| 25 | 2 |  |
| 50 | 7 | 2.50 |
| 75 | 24 | 2.43 |
| 100 | 44 | 0.83 |

* 1. Compare your analysis with the actual running times.
     1. O(n): I would expect that the growth rate to approach 21, and the actual running times are increasing linearly, which was expected.
     2. O(n2): I would expect that the growth rate to approach 22, but it does not. In fact, the growth rate decreases as N increases. This could be due to the IDE used, or that java optimized the byte code.
     3. O(n3): I would expect that the growth rate to approach 24, but it does not. In fact, the growth rate decreases as N increases. This could be due to the IDE used, or that java optimized the byte code.
     4. O(n2): I would expect that the growth rate to approach 22, but it does not. In fact, it increases and decreases occasionally depending on the size of N. This could be due to the IDE used, or that java optimized the byte code.
     5. O(n5): I would expect that the growth rate to approach 25, but it does not. In fact, the growth rate decreases as N increases. This could be due to the IDE used, or that java optimized the byte code.
     6. O(n4): I would expect that the growth rate to approach 24, but it does not. In fact, the growth rate decreases as N increases. This could be due to the IDE used, or that java optimized the byte code.

1. Weiss, Exercise 2.10 (part a): Determine, for the typical algorithms that you use to perform calculations by hand, the running time to do the following:
   1. Add two N-digit integers: **O(n)** because we can use the rule that running times are additive, and take note that it is bad practice to have lower order terms in Big-Oh notation. Thus, O(n+n) = O(2n) = O(n).
2. Weiss, Exercise 2.11: An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume low-order terms are negligible)
   1. Linear:
   2. O(NlogN):
   3. Quadratic:
   4. Cubic:
3. Weiss, Exercise 2.15: Give an efficient algorithm to determine if there exists an integer i such that Ai = i in an array of integers A1 < A2 < A3 < · · · < AN. What is the running time of your algorithm?

**Running time: O(N)**

Search.java

public class Search

{

public Search(int[] arr)

{

for(int i = 0; i < arr.length; i++)

{

if((arr.length - 1) == arr[i])

System.out.println("There exists an integer i such that: " + arr[i] + " = " + (arr.length - 1));

}

}

}

SearchTest.java

public class SearchTest

{

public static void main(String args[])

{

int[] a = {0, 3, 6, 9, 4};

new Search(a);

}

}