Caleb Faruki
Dr. Xiaoxu Han
CISC4080 Algorithms
October 4, 2013

Homework 1b: Questions 1 & 2

1.

$$[f_1(n)=O(g_1(n)) \ \wedge \ f_2(n)=O(g_2(n))] \Rightarrow f_1(n)\cdot f_2(n)=O(g_1(n)\cdot g_2(n))$$

$$(f_1 \le c_1\cdot g_1 \ \wedge \ f_2 \le c_2\cdot g_2) \Rightarrow (f_1\cdot f_2 \ \le \ c_1\cdot g_1\cdot c_2\cdot g_2),$$

$$Let\, c=c_1\cdot c_2 \ , \ f=f_1\cdot f_2 \ , \ g=g_1\cdot g_2, \ so \ f\le c\cdot g$$

This matches our definition of Big-O. Therefore, the product of f1 and f2 is upwardly bounded by the product of g1 and g2.

2. $$3^n=\omega(n^k), where\ constant\ k>1 \Rightarrow \frac{limit}{n\to\infty}\ \frac{f(n)}{g(n)} \ = \ \frac{3^n}{n^k} \Rightarrow \infty$$

This means that ***f(n)*** must grow at a faster rate than ***g(n)***. If ***k*** is a constant, that means for some input size, ***n*** will exceed ***k***. We know that $\mathbf{2^n}$ grows faster that $\mathbf{n^2}$. Therefore, we may induce that there exists an ***N*** where: $3^N \ge N^k,\ where\ N>k$

Insertion Sorting Analysis

2. WORST = O(n^2), BEST = O(n), AVERAGE = O(n^2)
3.

It took 0 clicks, (0.000000 seconds) to sort this array of 10 entries
It took 0 clicks, (0.000000 seconds) to sort this array of 100 entries
It took 0 clicks, (0.000000 seconds) to sort this array of 1000 entries
It took 120000 clicks, (0.120000 seconds) to sort this array of 10000 entries
It took 11630000 clicks, (11.630000 seconds) to sort this array of 100000 entries
It took 1168800000 clicks, (1168.800000 seconds) to sort this array of 1000000 entries

We can infer from these results that insertSort() gets exponentially more **in**efficient as the input size grows past 10^4.

4.

It took 10000 clicks, (0.010000 secs) to sort a 10^6 array best case.
It took -1825547296 clicks, (-1825.547296 secs) to sort a 10^6 array worst case.
It took 1168800000 clicks, (1168.800000 seconds) to sort a 10^6 array average case.

While there may be cases where an array is pretty much sorted, that situation is trivial since a sort function is only useful for unsorted arrays. An array that needs sorting is assumed to be unsorted. Thus, we need an algorithm that is reliably faster in its worst- and average-case scenario.

If the array is small, insertSort() is great. But if we have a much larger array, we need a sort function that operates faster in its worst case. In fact, insertSort() is so slow in worst case for large input that it exceeds the max number of bytes allocated for the int data-type, causing overflow. Therefore, wee should not rely on insertSort() for sorting an array based on these results.