Kristina Frye
CS 350
January 15, 2015
Assignment 1

Problem 1
For each of the following six functions, state its rate of growth using Θ notation; if possible, use one of the Basic Asymptotic Efficiency Classes from Levitin Table 2.2. Explain your reasoning in one line. Then sort the functions from lowest to highest order of growth.

(a) $34n \in \Theta(n)$ by the definition of Θ notation because 34 is a constant. This is the linear efficiency class.

(b) $n! \in \Theta(n!)$ by the definition of Θ notation. The constant is 1. This is the factorial efficiency class.

(c) $2^{n+1} \in \Theta(2^n)$ by the definition of Θ notation because $2^{n+1} = 2 \cdot 2^n$. The constant is 2. This is the exponential efficiency class.

(d) $\sqrt{(144n)} \in \Theta(n^{\frac{1}{2}})$. The constant is 12. This efficiency class is somewhere been the logarithmic efficiency class and the linear efficiency class.

(e) $(n-4)! \in \Theta((n-4)!)$ with a constant of 1. This efficiency class is less than factorial since $(n-4)! < n!$ but greater than the exponential efficiency class since $(n-4)! > 2^n$ as n converges to infinity.

(f) $2\log_2 n^5 \in \Theta(\log n)$ because $2\log_2 n^5 = 2 \cdot 5 \log_2 n = 10 \log_2 n$ and 10 and 2 are just multiplicative constants. This is the logarithmic efficiency class.

(g) $n^4/200 + 100n^3 + 500000 - n \in \Theta(n^4)$ with a constant of $\frac{1}{200}$ because of the theorem on page 56 of the book that states the largest element of a sum of terms is the only element that needs to be considered when determining the rate of growth. Since $\frac{n^4}{200} \in \Theta(n^4)$ is the largest term in the sum, the entire sum is also in $\Theta(n^4)$.

Ordering: f, d, a, g, c, e, b

Problem 2
Prove (by using the definitions of the notations involved) that if $g(n) \in \Omega(t(n))$, then $t(n) \in O(g(n))$.

Proof:
If $g(n) \in \Omega(t(n))$, then $g(n) \geq ct(n)$ for some c by the definition of Ω. This implies $\frac{1}{c}g(n) \geq t(n)$ by the rules of algebra since $c > 0$. This implies $t(n) \in O(g(n))$ by the definition of O $\square$.

Problem 3
$p(n) = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} + a_{k-3} n^{k-3} + ... + a_0$, where the a_i are constants, is a polynomial of degree k. Prove that every polynomial of degree k belongs to $\Theta(n^k)$.

Proof:

Let:

$$
\begin{aligned}
t_1 &= a_k n^k, \\
t_2 &= a_{k-1} n^{k-1}, \\
t_3 &= a_{k-2} n^{k-2}, \\
&... \\
t_k &= a_0 n^0
\end{aligned}
$$

Then:

$$
\begin{aligned}
t_1 &\in \Theta(n^k), \\
t_2 &\in \Theta(n^{k-1}), \\
t_3 &\in \Theta(n^{k-2}), \\
&... \\
t_k &\in \Theta(n^0)
\end{aligned}
$$

by the definition of Θ. But $(t_1 + t_2 + t_3 + ... + t_k) \in \Theta(\max\{n^k, n^{k-1}, n^{k-2}, ..., n^0\})$ by the theorem on page 56 in the book and $\max\{n^k, n^{k-1}, n^{k-2}, ..., n^0\}) = n^k$. Therefore $p(n) = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} + a_{k-3} n^{k-3} + ... + a_0 \in \Theta(n^k)$ which means that every polynomial $\in \Theta(n^k)$ $\square$

Problem 4
Levitin mentions, in section 2, that one can check whether all elements of an array are distinct by a two-part algorithm that first sorts the array, and then scans through the array looking at adjacent elements. If the sorting is done by an algorithm with the time efficiency in $\Theta(n \log n)$, what will be the time efficiency class of the entire algorithm? Explain why.

Answer
The time efficiency class of the entire algorithm will also be $\Theta(n \log n)$. In a two part algorithm, the efficiency of the entire algorithm is the addition of the second part of the algorithm to the first part of the algorithm. Let the first part of the algorithm be $t_1(n) \in \Theta(n \log n)$. Let the second part of the algorithm (scanning through the array looking at adjacent elements) be $t_2(n)$. $t_2(n) \in \Theta(n)$ because each member of the array will need to be visited once (after they have been sorted) to confirm that each element is unique. This is linear time ($\Theta(n)$). The time efficiency class of the entire algorithm is $\Theta(\max\{n \log n, n\}) = \Theta(n \log n)$ by the theorem on page 56 in the book.

Problem 5
The range of a finite nonempty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indicate the time efficiency classes of these algorithms using the most appropriate notation ($O, \Theta,$ or Ω).

(a) An unsorted array: Set local variables max and min equal to the first element of the array. Step through each element of the array. If the value is greater than max, set max equal to the value. If the value is less than min, set min equal to the value. After each element has been visited, subtract min from max and return the resulting value. This is $\Theta(n)$ since each element of the array is visited exactly once.

(b) A sorted array: Subtract the first element of the array (the smallest number) from the last element of the array (the largest number). This is constant time: $\Theta(1)$.

(c) A sorted singly-linked list: Set the first element of the list equal to a local variable min. Traverse through the list until you get to the last element. Set the value of the last element equal to local variable max. Subtract min from max and return the result. This is linear time: $\Theta(n)$ since each element must be visited once.

(d) A binary search tree: Go to the first element of the binary search tree by choosing the left branch when it is exists. If no left branch exists, choose the right. Continue until a leaf is reached. This leaf will contain the minumum value. Store into a local variable. Starting from the root node, traverse the tree, choosing the right branch when it is available. If no right branch is available, choose the left branch. Continue until a leaf is reached. This leaf will contain the maximum value. Subtract the minimum from this value and return the result. The efficiency class is dependent upon the structure of the binary search tree. When finding the min and max values, the worst case is $O(n)$ when the tree is set up similar to a linked list with the desired value at the end of a long chain. The efficiency class could be $\Theta(\log n)$ if the tree is balanced. Therefore, the proper categorization is the linear efficiency class: $O(n) + O(n) = O(n)$.