Andrew Bastien
CSE 2010, Section E4
HW2 – Written Portion

1. Order the following functions by asymptotic growth rate.
*** 2^10 = O(1)**, constant time. Fixed number of operations.
*** 2 log n = O(log n)**, logarithmic time as it's the only thing involved.
*** 3n + 100 log n = O(n)**, linear time. The 3n has far more impact than the logarithmic part.
*** 4n = O(n)**, also linear time but is higher value than 3n.
*** 4n log n + 2n = O(n log n),** linearithmic, the 4n log n is much more significant than 2n.
*** n^2 + 10n = O(n^2),** dominated by n^2, quadratic.
*** n^3 = O(n^3),** cubic time.
*** 2^n = O(2^n),** exponential time. Really bad news…

2. Give a big-Oh characterization, in terms of n, of the running time of the example5 method shown in Code Fragment 4.12.

I'm going with **O(n^3)**, cubic time. We've got an arithmetic series (which is already n^2) and wrapping it in yet another loop of the same n.

3. Show that O(max{ f (n), g(n)}) = O(f (n) + g(n)).
The max function is equal to the highest size of its arguments. It's being given two functions that both take **n**, so it is going to be either the total of f(n) or g(n), not both. The highest it could possibly be is **2n** which is still linear time and boils down to **O(n)**.

4. Consider f (n) = 4n^2 + 3n − 1, mathematically show that f (n) is O(n^2), Ω(n^2), and Θ(n^2).
It's a fixed set of operations based only n^2. Quadratic growth dominates.
Simple
f(n) = 4n^2 + 3n -1 <= 4n^2 + 3n^2 – n^2 = 6n^2 using 6 as an example, for n >= 1, so it's definitely **O(n^2).**
f(n) = 4n^2 + 3n – 1 >= 4n^2 using 4 as an example, for n >= 1, once again it's **Ω(n^2)**
Since O(n^2) == Ω(n^2), then Θ(n^2).

5. For finding an item in a sorted array, consider "ternary search," which is similar to binary search. It compares array elements at two locations and eliminates 2/3 of the array. To analyze the number of comparisons, the recurrence equations are T(n) = 2 + T(n/3), T(2) = 2, and T(1) = 1, where n is the size of the array. Explain why the equations characterize "ternary search" and solve for T(n).

Ternary, or "composed of three parts." (I didn't know that until just looking it up. I would've off-handedly thought it's trinary.)
T(n) = 2 + T(n/3) = This is the case for where the number of elements is > 2. There are two comparisons to do and another recursive split.
T(2) = 2, T(1) = 1 are for when the above has resulted in a search that is 1 or 2 elements, and there's not enough to split again. Either our answer is here or it's not.
The bulk of any ternary search would be based on **2 log3(n) and a final T(2) or T(1).**

6. To analyze the time complexity of the "brute-force" algorithm in the programming part of this assignment, we would like to count the number of all possible schedules.

a. Explain the number of all possible schedules in terms of n (number of candidate courses) and m (number of time slots per candidate course).

The maximum amount of checks would be m^n, this is unfortunately very exponential.

There are some oddities in the process though… for every course there must be at least one time slot, but these time slots may or may not be duplicated. The order of the courses does matter as well as the order of the time slots, it’s very possible that there’s no contention on many checks but it’s also possible that there’s a LOT of contention for time slots...

b. Consider a computer that can process 1 billion schedules per second, n is 10, m is 20, explain the number of hours needed to process all possible schedules.

1 billion = 10^9
But m^n is 20^10

20^10 divided by 10^9 is 10,240 seconds… is about 2.85 hours.

That’s way too long.

c. If we do not want the computer to spend more than 1 minute, explain the largest n the computer can process when m is 20.

If m is fixed at 20 then we have anywhere from 1-20 n…

n is obtained using logarithmic operations.
N = log(60 * 10^9) / log(20) = 8.284….

The highest n is 8.