Math 234

Big-O notation Day 21

1. For each pair of functions below, decide whether they are asymptotically similar or, if not, which is asymptotically smaller than the other.

   (a) $f(n) = n^2 + 15$ and $g(n) = n^2 + 6n + 4$

   (b) $f(n) = n^{31} + n^{17}$ and $g(n) = 2n^{31} + n^{19}$

   (c) $f(n) = 6^n + 15n^4 + 5$ and $g(n) = 2 \cdot 6^n + 3n^3 + 1$

   (d) $f(n) = 5 \cdot n!$ and $g(n) = 3 \cdot n!$

2. Suppose that $f(n) \ll h(n)$ and $g(n) \ll h(n)$. Must it be true that $f(n) \cdot g(n) \ll h(n)$? Why or why not?

3. Suppose that $f(n) \ll h(n)$ and $g(n) \ll h(n)$. Prove that $f(n) \cdot g(n) \ll [h(n)]^2$.

4. Verify that asymptotic analysis discards multiplicative constants by proving that if $c > 0$ and $f(n) \ll g(n)$, then $c \cdot f(n) \ll g(n)$.

5. Suppose $f(n) \approx h(n)$ and $g(n) \approx h(n)$. Must it be true that $f(n) \cdot g(n) \approx [h(n)]^2$? Prove that your answer is correct.

6. Suppose you need to implement an algorithm to compute $1 + 2 + 3 + \cdots + n$. Here are two ways to do this:

   (a) You could use a loop to add up the first n positive integers. Use big-O notation to express how many mathematical operations this requires.

   (b) You could use the arithmetic sum formula $1+2+3+\ldots+n = \frac{1}{2}n(n+1)$. Use big-O notation to express how many mathematical operation's this requires.

7. Consider algorithms for searching for an item in a list. Let n be the length of the list.

   (a) If the list is unsorted, then the best you can do is examine each item in the list sequentially until the desired item is found. Use big-O notation to express the number of operations this requires.

   (b) If the list is sorted, describe an algorithm that allows you to search for a desired element with a number of operations that is asymptotically smaller than your answer in part (a).

8. The **bubble sort** is an easy-to-code algorithm for sorting a list.

   Given a list $(a_1, a_2, \ldots, a_n)$ of numbers, the bubble sort performs a "sweep" through the list, comparing each pair of consecutive numbers. The algorithm first compares the first pair of numbers, a_1 and a_2 to see if they are in ascending order. If they are not, the two numbers are swapped, placing the pair in ascending order. If the two values are already in ascending order, no swap is performed. In either case, the algorithm then proceeds to consider the second pair of numbers, a_2 and a_3, swapping them if necessary. This process continues until all consecutive pairs have been considered, after which a_n is known to be the largest number in the list and in its correct position.

   The algorithm then performs a second sweep of the list, considering just the entries a_1 through a_{n+1}. After this second sweep, both a_{n-1} and a_n are in their correct positions.

   After $n - 1$ sweeps have been performed, the list is sorted in ascending order.

   (a) In the first sweep, how many pairs must be considered for swapping? (Your answer should depend on n.)

   (b) In the second sweep, how many pairs must be considered for swapping?

   (c) In the third sweep, how many pairs must be considered for swapping?

   (d) In the last sweep, how many pairs must be considered for swapping?

   (e) For the entire bubble sort on a list of n numbers, how many pairs must be considered for swapping?

   (f) Use big-O notation to express the number of operations required to sort a list of n numbers using the bubble sort.

9. As its name suggest, the **quicksort** is a fast algorithm for sorting a list. One version of quicksort works as follows:

   One entry in the list (ideally the median) is selected as the *pivot*. The algorithm then sweeps through the list, moving all numbers smaller than the pivot to the left of the pivot, and all numbers larger than the pivot to the right of the pivot. The pivot is then in its correct position in the list.

   The algorithm now considers two sublists: one containing all numbers less than the pivot and another containing all numbers greater than the pivot. Each sublist is roughly half the size of the initial list. Each sublist is sorted by (recursively!) applying the quicksort algorithm.

   (a) Suppose the initial length of the list is 2^n. How many times can the list be divided in half?

   (b) How many operations are required for each sweep through the list?

   (c) Use big-O notation to express the number of operations required to sort a list of n numbers using the quicksort.