Overview

This homework contains several problems related to the order notation and algorithm analysis. Written solutions must be turned in at the start of class on Tuesday, November 20th. One of the TAs will pick up the solutions within the first 10 minutes of class. After that time no homework will be accepted.

Note: This will be the last written homework of the semester. For the material covers after the Thanksgiving break, there will be practice problems, but nothing to submit.

Problems

1. (10 points) For each of the functions $g(n)$ below give a new function $f(n)$, such that $g(n)$ is $f(n)$. For example, if $g(n) = 5n - 16$ then $f(n) = n$. Similarly, if $g(n) = 123$, then $f(n) = 1$. You may use the hierarchy given in the lecture notes.

   (a) $g(n) = 145n - 12$.
   Solution: $O(n)$

   (b) $g(n) = n + 10n^2 + n^3$.
   Solution: $O(n^3)$

   (c) $g(n) = 5n \log n + 10n$.
   Solution: $O(n \log n)$

   (d) $g(n) = n + \log n$.
   Solution: $O(n)$

2. (10 points) For each of the following code segments, use order notation to describe the number of operations required. Briefly (very briefly) explain your answers:

   (a) 
   ```c
   for ( int i=0; i<n; ++i )
   sum += i;
   ```
   Solution: $O(n)$ because the loop requires $n$ iterations.
(b) for ( int i=10; i<n-1; ++i )
    sum += i;
Solution: $O(n)$. The loop requires $n - 11$ iterations, but this is $O(n)$.

(c) for ( int i=0; i<n; ++i )
    for ( int j=i; j<n; ++j )
        sum += j;
Solution: $O(n^2)$. The inner loop requires $O(n)$ iterators for each of the $O(n)$ iterations of the outer loop.

(d) for ( int i=0; i<n; ++i )
    for ( int j=i; j<n; ++j )
        sum += j;
    for ( int k=0; k<n; ++k )
        sum += k;
Solution: $O(n^2)$. The first set of loops is $O(n^2)$, just as in the previous problem. The separate second loop is $O(n)$. In combining these, the $O(n^2)$ predominates.

(e) for ( int i=0; i<n; ++i )
    for ( int j=i; j<n; ++j )
        for ( int k=j; k<n; ++k )
            sum += k;
Solution: $O(n^3)$. The upper bound on the number of operations of the inner loop is $O(n)$ iterations per iteration of the middle loop, which requires $O(n)$ iterations per iteration of the outer loop, which requires $O(n)$ iterations.

3. (10 points) Which is more expensive: (a) starting from a pointer to the head node of a singly linked list and inserting at the end of the list, or (b) inserting at the back of a vector (assume there is no need for reallocation)? Justify your answer by giving (and briefly explaining) the number of operations (using order notation) that would be required for each.

Solution: Option (a) would be more expensive. It would require step through all nodes to get to the end. This is an $O(n)$ operation. Option (b) requires just directly accessing the new location at the end of the array, an $O(1)$ operation.
4. **(10 points)** Suppose you need to read in an unsorted sequence of integers into a vector and you wanted the vector sorted after you are done reading in the values. Would it be cheaper to (a) insert the values in order or would it be cheaper to (b) read in all the values and then sort them? Justify your answer by explaining how each technique would work and giving numbers of operations using order notation.

**Solution:** In (a) each insert could be as bad as $O(n)$ if it had to occur at the front to maintain order. Overall, this would give an $O(n^2)$ time because it would need to occur $O(n)$ times. In (b) the insert would require $O(n)$ time overall, followed by $O(n \log n)$ for sorting. Since these happen in succession, the $O(n \log n)$ time would predominate. Thus, (b) is faster.