COMP4500/7500 Advanced Algorithms & Data Structures Tutorial Exercises 2 (2014/2)[*]

School of Information Technology and Electrical Engineering, University of Queensland

August 4, 2014

This material aims to familiarise you with asymptotic notation and develop your skill at manipulating formulae involving asymptotic notation. A good treatment of the basics of asymptotic notation may be found in CLRS 3.1 or CLR 2.1. It also aims to familiarise you with

- divide-and-conquer algorithms;
- the use of recurrence equations to represent the running time of recursive algorithms; and
- develop your skill at solving recurrences.

Good treatments of divide-and-conquer and recurrences may be found in Chapters 2 and 4 of CLRS, respectively (1 and 4 of CLR).

1. (See CLRS Exercise 1.2-2, p14 [3rd], p13 [2nd], CLR Exercise 1.4-1, p17 [1st])
   Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, assume insertion sort runs in $10n^2$ steps, while merge sort runs in $100n \lg n$ steps, where $\lg n$ stands for $\log_2 n$. For which values of n does insertion sort beat merge sort? You will need to use a bit of algebra, but may then use trial and error if you wish.

   How might one write a sorting algorithm which has the advantages of insertion sort for small inputs and the advantages of merge sort for larger inputs? (For an extension, see CLRS Problem 2-1 p39 [3rd], p37 [2nd], CLR Problem 1-2, p17 [1st]).

2. (CLRS Exercise 3.1-1, p52 [3rd], p50 [2nd], CLR Exercise 2.1-1, p31 [1st])
   A function h is asymptotically nonnegative if there exists an n_h such that

   $$(\forall n \bullet n \geq n_h \Rightarrow h(n) \geq 0)$$

   Let f and g be asymptotically nonnegative functions. Using the definition of Θ-notation, prove that the function $\max(f(n), g(n)) \in \Theta(f(n) + g(n))$, where for each value of n the function $\max(f(n), g(n))$ returns the maximum of $f(n)$ and $g(n)$. As abbreviations, define functions M and S so that, for all n

   $$\begin{aligned} M(n) &= \max(f(n), g(n)) \\ S(n) &= f(n) + g(n) \end{aligned}$$

3. (CLRS Exercise 3.1-3, p53 [3rd], p50 [2nd], CLR Exercise 2.1-3, p31 [1st])
   Explain why the statement, "The running time of algorithm A is at least $O(n^2)$", is content-free.

4. (CLRS Exercise 3.1-4, p53 [3rd], p50 [2nd], CLR Exercise 2.1-4, p31 [1st])
   Is $2^{n+1} \in O(2^n)$? Is $2^{2n} \in O(2^n)$?

5. Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{25}$ of the functions satisfying $g_1 \in \Omega(g_2)$, $g_2 \in \Omega(g_3)$, ..., $g_{24} \in \Omega(g_{25})$. Partition your list into equivalence classes such that g_i and g_j are in the same class if and only if $g_i \in \Theta(g_j)$.

   $$\begin{array}{ccccc}
   (\frac{3}{2})^n & (\sqrt{2})^{\lg n} & \lg^* n & n^2 & (\lg n)! \\
   n^3 & (\lg n)^2 & \lg(n!) & 2^{2^n} & n^{\frac{1}{\lg n}} \\
   \lg\lg n & n \cdot 2^n & n^{\lg\lg n} & \ln n & 2^n \\
   2^{\lg n} & (\lg n)^{\lg n} & 4^{\lg n} & (n+1)! & \sqrt{\lg n} \\
   n! & 2^{\sqrt{2\lg n}} & n & n\lg n & 1
   \end{array}$$

   Try to rank as many of the above functions as you can. You may need to refer to CLRS 3.2 or CLR 2.2 for the definition of $\lg^* n$ and help.

[*]Copyright © reserved August 4, 2014

6. (CLRS Exercise 3.1-2, p52 [3rd], p50 [2nd], CLR Exercise 2.1-2, p31 [1st])
   Show that for any real constants a and b, where $b > 0$,
   $$(n+a)^b \in \Theta(n^b).$$

7. (Programming problem)
   Algorithms for both insertion sort and merge sort have been studied. You may use any implementations that you wish.

   Perform an empirical analysis of these algorithms by running them in order to determine their execution time for different inputs (e.g., the best-case and worst-case inputs for insertion sort) and different sizes of input (eg, 1000, 2000, 4000). Compare your results with the results of the theoretical analysis of these algorithms. (You may find it useful to graph your experimental results.)

   You may also like to experiment with profiling the execution of the algorithms.

8. A *local minimum* of an array segment $A[lo..hi]$ is an element $A[k]$ satisfying,
   $$A[k-1] \geq A[k] \leq A[k+1]$$
   for $lo \leq k \leq hi$. We assume that $lo \leq hi$, and that the indices of the whole array, A, range from (at least) $lo-1$ through to $hi+1$. This allows us to also assume that
   $$A[lo-1] \geq A[lo] \wedge A[hi] \leq A[hi+1]. \tag{1}$$
   This condition guarantees that a local minimum must exist; without (1) the array may be either strictly increasing or strictly decreasing, in which case there is no local minimum.

   (a) Design an algorithm for finding a local minimum of the array segment $A[lo..hi]$ which is substantially faster than the obvious $O(n)$ one in the worst case. Hint: use a divide-and-conquer approach similar to binary search.

   (b) Give an analysis of your algorithm to determine the number of comparisons of array elements required to find a local minimum of an array of size n.

9. (See CLRS Exercise 4.4-6 p93 [3rd], 4.2-2 p72 [2nd], CLR Exercise 4.2-2, p60 [1st])
   Argue that the solution to the recurrence
   $$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + cn$$
   where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree. That is, we would like a lower bound on the solution to the recurrence.

10. (See CLRS Problem 4-1, p107 [3rd], p85 [2nd], CLR Problem 4-1, p72 [1st])
    Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 1$. Make your bounds as tight as possible, and justify your answers. Hint: use the master theorem.

    (a) $T(n) = 2T(\frac{n}{2}) + n^3$.
    (b) $T(n) = T(\frac{9n}{10}) + n$.
    (c) $T(n) = 16T(\frac{n}{4}) + n^2$.
    (d) $T(n) = 7T(\frac{n}{3}) + n^2$.
    (e) $T(n) = 7T(\frac{n}{2}) + n^2$.

11. (See CLRS Problem 4-1, p107 [3rd], p85 [2nd], CLR Problem 4-1, p72 [1st])
    Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers. Hint: use iteration.

(a) $T(n) = T(n-1) + n$.

(b) $T(n) = T(\sqrt{n}) + 1$.

12. Solve (find an asymptotic upper bound for) the recurrence

$$T(n) = T(n-a) + T(a) + n$$

where $a \geq 1$ is a constant, by using iteration to generate a guess and then using substitution to verify it.