Solutions to Midterm 1 - Monday, July 29th, 2009

CPSC320, Summer2009. Instructor: Dr. Lior Malka. (liorma@cs.ubc.ca)

---

1. Circle the correct answer:

   (a) $n^2 + n = \Omega(n^2)$ T

   Explanation: $n^2 + n \leq \frac{n^2}{2}$ for sufficiently large n.

   (b) $n^3 = O(\sqrt{n^5} + \sqrt{4n^3})$ F

   Explanation: $n^{5/2} + 2 \cdot n^{3/2} < 2n^{5/2} < n^3$ for sufficiently large n. Thus, there is no constant c such that $c \cdot 2n^{5/2} \geq n^3$ (which is equivalent to $2c \geq n^{\frac{1}{2}}$) for all large n.

   (c) if $f(n) = \Theta(g(n))$, then $g(n) = \Theta(f(n))$. T

   Explanation: follows immediately from the definition.

   (d) For any function f it holds that $f(n) = \Theta(f(n))$. T

   Explanation: follows immediately from the definition.

   (e) Any algorithm runs in constant time on inputs of size less than 1000 bits. T

   Explanation: Let A be any algorithm, which means that A terminates, and therefore there is a function f describing the running time of A. For any of the 1000! possible inputs x, let f_x denote the number of steps used by $A(x)$. Let c be the maximum over all of the values f_x. Since c is a constant, A runs in constant time on all inputs of size at most 1000.

   (f) The time complexity of any sorting algorithm is $\Omega(n \cdot \log(n))$. F

   Explanation: the lower bound applies to *comparison sorts*. It does not apply to algorithms like `RadixSort`.

   (g) `MergeSort` is balanced. T

   Explanation: the `Merge` sub-procedure preserves the order of elements it merges.

   (h) `QuickSort` is balanced. F

   Explanation: Consider the `Partition` sub-procedure. As it scans the array, it may encounter an element $A[j]$ smaller than the pivot. To make room for this element, it swaps it with an element $A[i]$ bigger than the pivot. Notice that inside $A[i, \ldots, j]$ there could be an element $A[i'] = A[i], i' > i$. Thus, when $A[i]$ is swapped with $A[j]$, the order between $A[i]$ and $A[i']$ is not maintained.

2. (a) Complete the pseudocode of `MergeSort`:

   `MergeSort`(A, p, r)

if $p < r$

  $q = \lfloor \frac{p+r}{2} \rfloor$

  `MergeSort`(p, q)

  `MergeSort`(q+1, r)

  `Merge`(A, p, q, r)

(b) What is the recurrence $T(n)$ describing the running time of `MergeSort`?

$T(n) = 2 \cdot T(\frac{n}{2}) + n$, ignoring the constant on n.

(c) Use the *Master Theorem* to solve the recurrence $T(n)$ from section (b).

Notice that $a = 2, b = 2, f(n) = n$, and $n^{\log_b(a)} = n$. Since $n = \Theta(n)$, the second case of the master theorem applies. Thus, $T(n) = \Theta(n^{\log_b(a)} \log(n)) = \Theta(n \cdot \log(n))$.

3. In this question you will analyze the time complexity of *variants* of `MergeSort`.

(a) We define a new variant of `MergeSort`. In this variant, instead of splitting A into two, we split A into four. We combine the four parts using the same technique used in `Merge`.

`4MergeSort`(A, p, r)

  if $p < r$

    $q_0 = \lfloor \frac{3p+r}{4} \rfloor, q_1 = \lfloor \frac{p+r}{2} \rfloor, q_2 = \lfloor \frac{p+3r}{4} \rfloor$

    `4MergeSort`(A, p, q_0)

    `4MergeSort`$(A, q_0 + 1, q_1)$

    `4MergeSort`$(A, q_1 + 1, q_2)$

    `4MergeSort`$(A, q_2 + 1, r)$

    `4Merge`(A, p, q_0, q_1, q_2, r)

Let $f(n)$ be a function describing the number of steps required by `4Merge`(A, p, q_0, q_1, q_2, r) to combine four sub arrays of size $n/4$ each. Complete the following:

$f(n) = \Theta(n)$ because we maintain four indices: one to each sub-array. At each stage we pick the smallest element pointed to by these four indices. This takes a constant number of comparisons.

Since we do it n times, $f(n) = O(n)$

(b) What is the recurrence $T(n)$ describing the running time of `4MergeSort`? To simplify the analysis, you can ignore constants.

$T(n) = 4T(\frac{n}{4}) + n$ for $n > 1$ (ignoring the constant) and $T(n) = 1$, for $n = 1$.

(c) Draw a recursion tree to solve $T(n)$ from section (b).

Answer. When we start building the tree, it has n at the root, and four siblings with cost

$T(\frac{n}{4})$. The next stage is to replace the terms $T(\frac{n}{4})$ with their cost. Thus, each of these four nodes is replaced with $\frac{n}{4}$ and four siblings of cost $T(\frac{n}{16})$. In turn, each of these 16 nodes will be replaced, until we get to the bottom of the tree at depth $\log_4 n$. That is, we have $1 + \log_4 n$ levels in total. Notice that the sum at each level is n: at level 0 the cost of the root is n, at level 1 there are 4 nodes with cost $\frac{n}{4}$ each, at level 2 there are 16 nodes with cost $\frac{n}{16}$ each, and so on, until at level $\log_4 n$ (the leaves) we have $4^{\log_4 n}$ nodes with cost 1. Thus, the total is $n(1 + \log_4 n) = n + \frac{n}{2} \log n = \Theta(n \cdot \log n)$.

(d) `4Merge`(A, p, q_0, q_1, q_2, r) merges elements x from four sub-arrays into $A[p, \dots, r]$. We want to debug this procedure. Thus, we modify `4Merge` so that, **for each x that it merges**, we scan all the elements in $A[p, \dots, r]$, print those smaller than x in blue, and those greater than x in red. This takes $r - p$ steps. Considering this modification, what is the new recurrence $T(n)$ describing the running time of `4MergeSort`? To simplify the analysis, you can ignore constants.

$T(n) = 4T(\frac{n}{4}) + n^2$, ignoring the constants on n^2

(e) Use the Master Theorem to solve the recurrence $T(n)$ from section (d).

Notice that $a = 4, b = 4, f(n) = n^2$, and there is $\epsilon > 0$ (say, $\epsilon = \frac{1}{2}$) such that $n^2 = \Omega(n^{\log_b(a)+\epsilon})$ $= \Omega(n^{\frac{3}{2}})$. The third case of the master theorem applies because there is c (say, $c = \frac{1}{4}$) such that $a \cdot f(\frac{n}{b}) \leq c \cdot f(n) \iff 4(\frac{n}{4})^2 \leq c \cdot n^2 \iff \frac{1}{4} \leq c$. Thus, $T(n) = \Theta(f(n)) = \Theta(n^2)$.