COMP 250 Fall 2004 - Midterm examination

October 18th 2003, 13:35-14:25

1 Running time analysis (20 points)

For each algorithm below, indicate the running time using the simplest and most accurate big-Oh notation, as a function of n. Assume that all arithmetic operations can be done in constant time. The first algorithm is an example. No justifications are required.

Algorithm	Running time in big-Oh notation
Algorithm Example(n) $x \leftarrow 0$ **for** $i \leftarrow 1$ **to** n **do** . $\quad x \leftarrow x + 1$	**O(n)**
Algorithm exam1(n) $i \leftarrow 1$ **while** $(i < n)$ **do** . $\quad i \leftarrow i * 2$	
Algorithm exam2(n) $x \leftarrow 0$ **for** $i \leftarrow 1$ **to** n **do** . $\quad$ **for** $j \leftarrow 1$ **to** $n - i$ **do** . $\qquad x \leftarrow x + 1$	
Algorithm exam3(n) **for** $i \leftarrow 1$ **to** 1000 . $\quad j \leftarrow 1$. $\quad$ **while** $j < i$ **do** . $\qquad j \leftarrow j + 1 + \log(i) + \sqrt{j}$	**Hint:** Don't spend more than a constant amount of time on this one!
Algorithm exam4(n) $i \leftarrow n$ **while** $(i > 0)$ **do** . $\quad i \leftarrow i - 10$	

2 Short answer questions (16 points)

a) **True or false? Justify your answer.** If the worst-case running time of an algorithm A is $O(n^{1.58})$ and the worst-case running time of an algorithm B is $O(n^2)$, then algorithm A will run faster than algorithm B on all input.

b) What does it mean for an algorithm A to run in constant time (i.e. in time $O(1)$)?

c) What kind of utilization of a list abstract data type would make an implementation using an array more efficient (in terms of running time) than an implementation using a linked-list?

d) Explain why, in an induction proof, it is absolutely necessary to prove the base case. Use at most three lines of text.

3 Linked lists and stacks (14 points)

The following algorithm takes a linked list as input and check if it has a certain property. What is that property? Under what condition will the checkProperty method return true?

Algorithm checkProperty(linkedList L)
Input: A linked-list L
Output: Returns true if the list L has the property, and false otherwise
Stack $s \leftarrow$ **new** Stack();
node $n \leftarrow L$.head;
while ($n \neq$ **null**) **do**
. s.push(n.getValue());
. $n \leftarrow n$.getNext();
while (L.head $\neq$ **null**) **do**
.   **if** (L.getFirst() + s.top() $\neq$ 10) **then return** false;
. L.removeFirst();
. s.pop();
return true;

4 Big-Oh relations (16 points)

a) Prove, using only the definition of the big-Oh notation, that
$3+(\sin(n))^2 \cdot n^{\cos n} \in O(n)$.

b) Prove, using any valid technique you want, that $n^2 + 10\log(n) + 10 \in \Theta(n^2)$.

5 Analysis of recursive algorithms (16 points)

Recall the pseudocode for the mergeSort algorithm:

Algorithm mergeSort(A, l, r)
Input: An array A of numbers, and indices l and r.
Output: The elements of $A[l...r]$ are sorted.
if $(l < r)$ **then**
. $mid \leftarrow \lfloor (l + r)/2 \rfloor$
. mergeSort(A, l, mid)
. mergeSort$(A, mid + 1, r)$
. merge(A, l, mid, r)

Suppose that by some miracle, someone provided you with a version of the "merge" algorithm for which the number of primitive operations performed was constant, say 100, instead of the linear-time algorithm seen in class.

a) (6 points) Let $T(n)$ be the total number of primitive operations performed by this miracle mergeSort when sorting an array of $n = r - l + 1$ elements. Write a recurrence equation for $T(n)$. For simplicity, assume that n is an exact power of two.

b) (10 points) Solve this recurrence equation to obtain an explicit formula for $T(n)$, using any method you want. Again, for simplicity, assume that n is an exact power of two.

6 Recursive algorithms (18 points)

You are a biologist conducting an experiment where you have prepared an array of n samples $S[0...n-1]$. You know that *at most one* of your samples is infected with a virus, but you don't know which one (if any). You have a machine that can take any consecutive subset of samples $S[i...j]$ and determine, using a single test kit, whether one of the samples in $S[i...j]$ is infected, but without telling exactly which sample it is. You have the time to conduct only $\lceil \log_2 n \rceil$ such test. Suppose this test is provided to you in the form of an algorithm:

Algorithm testSample(S, i, j)
Input: An array of samples S, and two indices i and j.
Output: Returns true if one of the samples in $S[i...j]$ is infected, and false otherwise.

Question: Write a recursive algorithm that returns the index of the infected sample, or -1 if no sample is infected. When executed on an array of n samples, your algorithm should call the testSample method at most $\lceil \log_2(n) \rceil$ times (but you don't need to prove that it does).

Algorithm findInfected($S, start, stop$)
Input: An array S of samples. Indices *start* and *stop*.
Output: The index of the infected sample between *start* and *stop* inclusively, or -1 if $A[start...stop]$ contains no infected sample.
/* WRITE YOUR PSEUDOCODE HERE */

7 Bonus question (5 points)

Prove that $\log(n!) \in \Theta(n \log(n))$.