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** . $x \leftarrow x + 1$	**O(n)**
Algorithm exam1(n) $i \leftarrow 1$ **while** $(i < n)$ **do** . $i \leftarrow i * 2$	O(log n). After k iterations of the loop, i has value 2^k, so when $k > log(n)$, $i > n$, and the loop stops.
Algorithm exam2(n) $x \leftarrow 0$ **for** $i \leftarrow 1$ **to** n **do** . **for** $j \leftarrow 1$ **to** $n - i$ **do** . $x \leftarrow x + 1$	$O(n^2)$
Algorithm exam3(n) **for** $i \leftarrow 1$ **to** 1000 . $j \leftarrow 1$. **while** $j < i$ **do** . $j \leftarrow j + 1 + \log(i) + \sqrt{j}$	O(1). Running time doesn't depend on n.
Algorithm exam4(n) $i \leftarrow n$ **while** $(i > 0)$ **do** . $i \leftarrow i - 10$	O(n). The loop runs for $n/10$ iterations

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.
Answer: This is false for several reasons:
(i) We are only talking about worst-case here. B may be faster than A in some other cases.
(ii) The big-Oh notation only allows conclusions when n is very large. For small n, B may be faster than A. That was the case for standardMultiplication versus fastRecursiveMultiplication.

b) What does it mean for an algorithm A to run in constant time (i.e. in time $O(1)$)?
Answer: The running time of A is bounded above by a constant that doesn't depend on the size of the input. Notice that it's false to say that the running time is independent of the input size; the running time may vary, but it has to be bounded by a constant.

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?
Answer: An utilization where one of these is true: (i) the maximum number of elements to be stored is known in advance and not too large, (ii) we need fast access to elements at arbitrary positions of the list (ii) insertions and removal of elements are rare, outside of that at the end of the list.

d) Explain why, in an induction proof, it is absolutely necessary to prove the base case. Use at most three lines of text.
Answer: In an induction proof, the induction step only shows that if $P(n)$ is true, then $P(n+1)$ will be true. If we can't somehow show that $P(a)$ is true for at least one a, then we can't conclude anything from the induction step.

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;

Answer: Let L be the length of the input list. The algorithm verifies that the sum of the n-th element and the $L - n - 1$-th element of the list is equal to 10, for all n between 0 and $L - 1$.

4 Big-Oh relations (16 points)

a) Prove, using only the definition of the big-Oh notation, that $3+(\sin(n))^2 \cdot n^{\cos n} \in O(n)$.
Answer: We need to find c and n_0 such that $3+(\sin(n))^2 \cdot n^{\cos n} \leq c \cdot n$. Since $\sin(n) < 1$ and $\cos(n) < 1$, $3+(\sin(n))^2 \cdot n^{\cos n} \leq 3 + 1 \cdot n^1 = 3 + n$. Choosing $n_0 = 3$, we find that if $n \geq n_0$, we get $3 + n \leq n + n = 2n = c \cdot n$, for $c = 2$. Thus, for $c = 2$ and $n_0 = 3$, we have that if $n \geq n_0$ $3+(\sin(n))^2 \cdot n^{\cos n} \leq c \cdot n$.

b) Prove, using any valid technique you want, that $n^2 + 10\log(n) + 10 \in \Theta(n^2)$.
Answer: The easiest solution is to use the ratio test:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{n^2 + 10\log(n) + 10}{n^2} &= \lim_{n\to+\infty} \frac{n^2 + 10\log(n) + 10}{n^2} \\
&= \lim_{n\to+\infty} \frac{d/dn(n^2 + 10\log(n) + 10)}{d/dn(n^2)} \\
&= \lim_{n\to+\infty} \frac{2n + 10/n}{2n} \\
&= \lim_{n\to+\infty} 1 + 5/n^2 \\
&= 1
\end{aligned}$$

Thus, by the ratio test, since the limit exists, is different from zero and from $+\infty$, we can conclude 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.
Answer:
$T(n) = 106 + 2T(n/2)$ if$n > 1$
T(1) = 2

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.

$$\begin{aligned} T(n) &= 106 + 2T(n/2) \\ &= 106 + 2(106 + 2T(n/4)) = (1 + 2)106 + 4T(n/4) \\ &= (1 + 2)106 + 4(106 + 2T(n/8)) = (1 + 2 + 4)106 + 8T(n/8) \\ &= ... \\ &= (1 + 2 + 4 + ... + 2^{k-1})106 + 2^k T(n/2^k) \end{aligned}$$

When $n = 2^k$, the recursion stops, with $T(1) = 2$. Thus, $T(n) = (1 + 2 + 4 + ... + 2^{k-1})106 + 2^k T(1) = (2^k - 1)106 + 2^k 2 = 106(n - 1) + 2n = 108n - 106$.

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 */
if (start = stop) then
.       if (testSample(S, start, stop)) then return start;
.       else return -1;
else
.       if (testSample(S,start,⌊(start + stop)/2⌋)) then
.               return findInfected(S, start,⌊(start + stop)/2⌋)
.        else return findInfected(S,⌊(start + stop)/2⌋,stop)
```

7 Bonus question (5 points)

Prove that $\log(n!) \in \Theta(n \log(n))$.
Answer: See solution to homework 3.