CS 136 Spring 2007
Tutorial 7: Midterm Review
Sample Solutions

II. Efficiency Analysis

Informal efficiency analysis

The following table gives the asymptotic run-times for the two methods in the two implementations:

	`ss-list%`	`ss-vect%`
`member?`	$O(n)$	$O(\log n)$
`insert!`	$O(1)$	$O(n)$

safe-insert

The asymptotic run-time of `safe-insert!` is $O(n)$ in both implementations since

$$O(n) + O(\log n) = O(1) + O(n) = O(n).$$

Formal analysis

In order to analyze `contains-duplicate?`, we first need to look at the helper function `contains?`. So define $C(n)$ to be the maximal number of steps to evaluate `(contains? lst num)` when `lst` has n elements.

Then we can see that

$$C(n) = \begin{cases} c_1, & n = 0 \\ c_2 + C(n-1), & n \geq 1 \end{cases}$$

where c_1 and c_2 are some positive constants.

Solving this recurrence (not shown — see Lecture Module 5, Page 6) gives us the explicit formula $C(n) = c_1 + c_2 n$.

Now we can write a recurrence for $T(n)$, the maximum number of steps to evaluate `contains-duplicate?` on input of length n, in terms of $T(m)$ for $m < n$ and $C(m)$:

$$T(n) = \begin{cases} c_3, & n = 0 \\ c_4 + C(n-1) + T(n-1), & n \geq 1 \end{cases}$$

where again c_3, c_4 are positive constants.

Substituting the explicit formula for $C(n)$ and creating the new constant $c_5 = c_4 + c_1 - c_2$ gives the simplification:

$$T(n) = \begin{cases} c_3, & n = 0 \\ c_5 + c_2 n + T(n-1), & n \geq 1 \end{cases}$$

III. Proofs

Solving a recurrence

Let's examine some values of $T(n)$ to try to guess a recurrence:

$$\begin{aligned} T(0) &= c_3 \\ T(1) &= c_5 + c_2 + c_3 \\ T(2) &= 2c_5 + (1+2)c_2 + c_3 \\ T(3) &= 3c_5 + (1+2+3)c_2 + c_3 \\ T(4) &= 4c_5 + (1+2+3+4)c_2 + c_3 \end{aligned}$$

From this, we guess that $T(n) = nc_5 + (1 + 2 + \cdots + n)c_2 + c_3$. And we know from some basic math course that $1 + 2 + \cdots + n = n(n+1)/2$. So we have (still a guess) that

$$T(n) = c_5 n + c_2 \frac{n(n+1)}{2} + c_3.$$

Now we want to prove this by induction:

Proof. **Claim:** $T(n) = c_5 n + c_2 n(n+1)/2 + c_3$ for all $n \geq 0$

Base case: $n = 0$ From the recurrence, we know that $T(0) = c_3$. And

$$c_5 \cdot 0 + c_2 \cdot 0 \cdot (0+1)/2 + c_3 = 0,$$

so the claim holds for the base case when $n = 0$.

Induction Hypothesis Assume that $T(k) = c_5 k + c_2 k(k+1)/2 + c_3$ for some $k \geq 0$.

Inductive Step Since $k \geq 0$, $k+1 \geq 1$, so we know from the recurrence that $T(k+1) = c_5 + c_2(k+1) + T(k)$. Then, using the induction hypothesis, we have:

$$\begin{aligned} T(k+1) &= c_5 + c_2(k+1) + c_5 k + c_2 \frac{k(k+1)}{2} + c_3 \\ &= c_5(k+1) + c_2(k+1)\left(1 + \frac{k}{2}\right) + c_3 \\ &= c_5(k+1) + c_2 \frac{(k+1)(k+2)}{2} + c_3 \end{aligned}$$

So the claim holds for $n = k+1$ whenever the claim holds for $n = k$.

Conclusion Therefore, by the principle of mathematical induction, the claim holds for all $n \geq 0$, and we are done.

□

Proving $f(n)$ is $O(g(n))$

We want to prove that $T(n)$ is $O(n^2)$, using the explicit formula we just computed. First, let's simplify our formula for $T(n)$:

$$\begin{aligned} T(n) &= c_5 n + c_2 \frac{n(n+1)}{2} + c_3 \\ &= \frac{c_2}{2} n^2 + \frac{2c_5 + c_2}{2} n + c_3 \end{aligned}$$

So if we create two more contants

$$c_6 = \frac{c_2}{2}, \qquad c_7 = \frac{2c_5 + c_2}{2},$$

then we have $T(n) = c_6 n^2 + c_7 n + c_3$. Now proving $T(n)$ is $O(n^2)$ should be straightforward.

When we are proving something is order of something else, we need to choose the constants c and n_0 to use in the definition. I'll choose $c = c_6 + c_7 + c_3$ and $n_0 = 1$. Many other choices for these constants would also work.

For the proof, we need to show that $T(n) \leq cn^2$ for all $n \geq n_0$. Since $n \geq 1$, we know that $n \leq n^2$ and $1 \leq n^2$, so we can write

$$T(n) = c_6 n^2 + c_7 n + c_3 \leq c_6 n^2 + c_7 c^2 + c_3 n^2 = (c_6 + c_7 + c_3) n^2 = cn^2$$

whenever $n \geq n_0 = 1$. Therefore, by the definition of order notation, $T(n)$ is $O(n^2)$.

Proving $f(n)$ is *not* $O(g(n))$

We want to prove that $T(n)$ is not $O(n(\log n)^2)$. To do this, we will want to use that fact that

$$1 < \log n < (\log n)^2 < n$$

whenever $n > 16$.

In general, to prove something is *not* order of something else, we will use a proof by contradiction. So we will not get to choose the constants c and n_0, but we will choose a special value of n to show a contradiction.

For this proof, assume by way of contradiction that $T(n)$ is $O(n(\log n)^2)$. Then, by the definition of order notation, there exist positive constants c and n_0 such that $T(n) \leq cn(\log n)^2$ whenever $n \geq n_0$. To show a contradiction, let $k = \max\{c(c+1)^2, n_0, 17\}$, and let $n = k^{c+1}$.

Then $k > 16$, so $k > (\log k)^2$. And since $k \geq c(c+1)^2$ and $c \geq 1$, $k^c \geq c(c+1)^2$. Using these facts, we have:

$$\begin{aligned}
T(n) &= c_6 n^2 + c_7 n + c_3 \\
&> n^2 \\
&= nk^c k \\
&> nc(c+1)^2(\log k)^2 \\
&= cn\left((c+1)\log k\right)^2 \\
&= cn(\log k^{c+1})^2 \\
&= cn(\log n)^2
\end{aligned}$$

So $T(n) > cn(\log n)^2$. And since $k \geq n_0$, $n \geq n_0$, so this is a contradiction. Therefore our original assumption must be false; namely $T(n)$ is *not* $O(n(\log n)^2)$.