ANALYZING THE WORST-CASE RUN TIME OF AN ALGORITHM

Supplemental notes for the lecture on September 19, 2012

Let $\mathscr{A}$ be the algorithm we are trying to analyze. Then we will define $T(N)$ to be the worst-case run-time of $\mathscr{A}$ over all inputs of size N. Slightly more formally, let $t_{\mathscr{A}}(\mathbf{x})$ be the number of steps taken by the algorithm $\mathscr{A}$ on input $\mathbf{x}$. Then

$$T(N) = \max_{\mathbf{x}:\mathbf{x} \text{ is of size } N} t_{\mathscr{A}}(\mathbf{x}). \tag{1}$$

In this note, we present two useful strategies to prove statements like $T(N)$ is $O(g(N))$ or $T(N)$ is $\Omega(h(N))$. Then we will analyze the run time of a very simple algorithm.

1 Preliminaries

We now collect two properties of asymptotic notation that we will need in this note (we saw these in class today).

Lemma 1. *If f and g are both $O(h)$ ($\Omega(h)$ resp.) then $f+g$ is $O(h)$ ($\Omega(h)$ resp.).*

Lemma 2. *If f is $O(h_1)$ ($\Omega(h_1)$ resp.) and g is $O(h_2)$ ($\Omega(h_2)$ resp.) then $f \cdot g$ is $O(h_1 \cdot h_2)$ ($\Omega(h_1 \cdot h_2)$ resp.).*

2 Proving $T(N)$ is $O(f(N))$

We start off with an analogy. Say you wanted prove that given m numbers $a_1, \ldots, a_m$, $\max_i a_i \leq U$. Then how would you go about doing so? One way is to argue that the maximum value is attained at i^* and then show that $a_{i^*} \leq U$. Now this is a perfectly valid way to prove the inequality we are after but note that you will *also* have to prove that the maximum value is attained at i^*. Generally, this is a non-trivial task. However, consider the following strategy:

> Show that for *every* $1 \leq i \leq m$, $a_i \leq U$. Then conclude that $\max_i a_i \leq U$.

Mathematically the above two strategies are the same. However, in "practice," using the strategy above turns out to be much easier. Thus, here is the strategy to prove that $T(N)$ is $O(f(N))$:

> For every large enough N, show that for **every** input $\mathbf{x}$ of size N, $t_{\mathscr{A}}(\mathbf{x})$ is $O(f(N))$. Then conclude that $T(N)$ is $O(f(N))$.

3 Proving $T(N)$ is $\Omega(f(N))$

We start off with the same analogy as in the previous section. Say you wanted prove that given m numbers $a_1, \ldots, a_m$, $\max_i a_i \geq L$. Then how would you go about doing so? Again, one way is to argue that the maximum value is attained at i^* and then show that $a_{i^*} \geq L$. Now this is a perfectly valid way to prove the inequality we are after but note that you will *also* have to prove that the maximum value is attained at i^*. Generally, this is a non-trivial task. However, consider the following strategy:

Show that there *exists* an $1 \le i \le m$, such that $a_i \ge L$. Then conclude that $\max_i a_i \ge L$.

Mathematically the above two strategies are the same. However, in "practice," using the strategy above turns out to be much easier. Thus, here is the strategy to prove that $T(N)$ is $\Omega(f(N))$:

For every large enough N, show that there **exists** an input $\mathbf{x}$ of size N, $t_{\mathscr{A}}(\mathbf{x})$ is $\Omega(f(N))$. Then conclude that $T(N)$ is $\Omega(f(N))$.

4 An Example

Now let us use all the above tools to asymptotically bound the run-time of a simple algorithm. Consider the following simple problem: given $n+1$ numbers $a_1, \ldots, a_n; v$, we should output $1 \le i \le n$ if $a_i = v$ (if there are multiple such i's then output any one of them) else output -1. Below is a simple algorithm to solve this problem.

Algorithm 1 Simple Search

INPUT: $a_1, \ldots, a_n; v$
OUTPUT: i if $a_i = v$; -1 otherwise

1: FOR every $1 \le i \le n$ DO
2: IF $a_i = v$ THEN RETURN i
3: RETURN -1

We will show the following:

Theorem 1. *The Simple Search algorithm 1 has a run time of* $\Theta(n)$.

We will prove Theorem 1 by proving[1] Lemmas 3 and 4.

Lemma 3. *$T(n)$ for Algorithm 1 is $O(n)$.*

Proof. We will use the strategy outlined in Section 2. Let $a_1, \ldots, a_n; v$ be an arbitrary input. Then first note that there are at most n iterations of the for loop in Step 1. Further, each iteration of the for loop (i.e. Step 2) can be implemented in $O(1)$ time (since it involves one comparison and a potential return of the output value). Thus, by Lemma 2, the total times taken overall in Steps 1 and 2 is given by

$$T_{12} \le O(n \cdot 1) = O(n).$$

Further, since Step 3 is a simple return statement, it takes time $T_3 = O(1)$ time. Thus, we have that

$$t_{\text{Algorithm 1}}(a_1, \ldots, a_n; v) = T_{12} + T_3 \le O(n) + O(1) \le O(n),$$

where the last inequality follows from Lemma 1 and the fact that $O(1)$ is also $O(n)$. Since the choice of $a_1, \ldots, a_n; v$ was arbitrary, the proof is complete. □

Lemma 4. *$T(n)$ for Algorithm 1 is $\Omega(n)$.*

[1] Note that I am not presenting separated out proof ideas so these are not "ideal" solutions for the HWs.

Proof. We will follow the strategy laid out in Section 3. For every $n \geq 1$, consider the specific input $a'_i = n+1-i$ (for every $1 \leq i \leq n$) and $v' = 1$.

For the above specific input, it can be easily checked that the condition in Step 2 is only satisfied when $i = n$. In other words, the for loop runs at least (actually exactly) n times. Further, each iteration of this loop (i.e. Step 2) has to perform at least one comparison, which means that this step takes $\Omega(1)$ time. Since n is $\Omega(n)$, by Lemma 2 (using notation from the proof of Lemma 3), we have

$$T_{12} \geq \Omega(n \cdot 1) = \Omega(n).$$

Thus, we have

$$t_{\text{Algorithm 1}}(a'_1, \ldots, a'_n; v') \geq T_{12} \geq \Omega(n).$$

Since we have shown the existence of one input for each $n \geq 1$ for which the run-time is $\Omega(n)$, the proof is complete. □

A quick remark on the proof of Lemma 4. Since by Section 3, we only need to exhibit only *one* input with runtime $\Omega(n)$, the input instance in the proof of Lemma 4 is only one possibility. One can choose other instances: e.g. we can choose an instance where the output has to be -1 (as a specific instance consider $a_i = i$ and $v = 0$). For this instance one can make a similar argument as in the proof of Lemma 4 to show that $T(n) \geq \Omega(n)$.

Exercise. If you think you need more examples to work through to make yourself comfortable with analyzing $T(n)$ for different algorithms, show that the binary search algorithm on n sorted numbers takes $\Theta(\log n)$ time.