# Analyzing the worst-case run time of an algorithm

Supplemental notes for the lecture on September 15, 2014

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

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

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 $a_1,\ldots,a_m$, $\max_i a_i \le U$. 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 $a_{i^*} \le U$. 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* $1 \le i \le m$, $a_i \le U$. Then conclude that $\max_i a_i \le U$.

Let us consider an example to illustrate the two strategies above. Let us say for whatever reason we are interested in showing that the age of the oldest person in 331 lectures is at most 100. Since there are 118 students registered and I am always present in class, there are at most $m = 119$ folks in the class. Let us order them somehow and let $A_i$ denote the age of the $i$'th person. Then we want to show that $\max\{a_1,\ldots,a_{119}\} \le 100$ (i.e. $U = 100$). The first strategy above would be to first figure out who is the oldest person in room: say that is the $i^*$'th person (where $1 \le i^* \le 119$) and then check if $a_{i^*} \le 100$. However, this strategy is somewhat invasive: e.g. the oldest person might not want to reveal that they are the oldest person in the room. This is where the second strategy works better: we ask every person in the room if their age is $\le 100$: i.e. we check if for every $1 \le i \le 119$, $a_i \le 100$. If everyone says yes, then we have proved that $\max_i a_i \le 100$ (without revealing the identity of the oldest person).

Mathematically the above two strategies are the same. However, in "practice," using the second strategy turns out to be much easier. (E.g. this was true in the age example above.) 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 $a_1,\dots,a_m$, $\max_i a_i \geq L$. 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 $1 \leq i \leq m$, such that $a_i \geq L$. Then conclude that $\max_i a_i \geq L$.

Let us go back to the class roam example. Now let us say we are interesting in proving that the oldest person in the room is at least 25 years old. (So $a_1,\dots,a_m$ is as in Section 2 but now $L = 25$.) Again, the first strategy would be to first figure out the oldest person, say $i^*$ and check if $a_{i^*} \geq 25$. However, as we saw in Section 2, this strategy is somewhat invasive. However, consider the the following implementation of the second strategy above. Say for the sake of mathematics, I come forward and volunteer the information that my age is at least 25. Since the oldest person's age has to be at least mine, this proves that $\max_i a_i \geq 25$, as desired.

Mathematically the above two strategies are the same. However, in "practice," using the strategy second turns out to be much easier. (E.g., this was true in the age example above.) 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 strategies from Section 2 and Section 3 to asymptotically bound the run-time of a simple algorithm. Consider the following simple problem: given $n+1$ numbers $a_1,\dots,a_n; v$, we should output $1 \leq i \leq n$ 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: $a_1,\dots,a_n; v$
OUTPUT: $i$ if $a_i = v$; $-1$ otherwise

1: FOR every $1 \leq i \leq n$ 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 $O(n)$.*

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

*Proof.* We will use the strategy outlined in Section 2. Let $a_1, \dots, a_n; v$ 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

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

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

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

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

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

Thus, we have

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

Since we have shown the existence of one input for each $n \ge 1$ 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 $T(n) \ge \Omega(n)$.

**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.

## 5 The Best-Case Input “Trap"

We now briefly talk about a common mistake that is made when one starts trying to prove $\Omega(\cdot)$ on $T(N)$. Note that in Section 3, it says that one can prove that $T(N)$ to be $\Omega(f(N))$ for every large enough $N$, one only needs to pick *one* input of size $N$ for which the algorithm takes $\Omega(f(N))$ steps.

The confusing part about the strategy in Section 3 is how does one get a hand on that special input that will prove the $\Omega(f(N))$ bound. There is no mechanical way of finding this input. Generally, speaking you have to look at the algorithm and get a feel for what input might force the algorithm to spend a lot of time. Sometimes, the analysis of the $O(\cdot)$ bound itself gives gives us a clue.

However, one way of picking the "special" input that **almost always never works** *in practice* is to consider (for every large enough $N$), the "best-case input," i.e. an input of size $N$ on which the algorithm runs very fast. Now such an input will give you a valid lower bound but it would almost never give you a *tight* lower bound.

So for example, let us try to prove Lemma 4 using the best case input. Here is one best case input: $a_i = i$ for every $i \in [n]$ and $v = 1$. Note that in this case the algorithm finds a match in the first iteration and this terminates in constant many steps. Thus, this will prove an $\Omega(1)$ lower bound but that is not tight/good enough.

Another common mistake is to make an argument for a fixed value of $N$ (say $N = 1$). However, note that in this case one can never prove a bound better than $\Omega(1)$ and again, this trick never works in proving any meaningful lower bound.