Imperial College London – Department of Computing

MSc in Computing Science

580: Algorithms
Tutorial 1 (Model Answer)

1. Using asymptotic notation, state an upper and lower bound for the time complexity of the SimpleSearch procedure for *any input*. Can you also give a tight (Θ) bound?

Answer: Although asymptotic notation applies to functions, in computing it is also (abusively) applied to algorithm performance even when the performance cannot be characterised by a single function. So, we say that the time complexity of the algorithm for any input has an upper bound $O(g(N))$ if its worst case running time is $O(g(N))$, and it has a lower bound $\Omega(g(N))$ if its best case running time is $\Omega(g(N))$.

Since the time taken by simple search in the worst case is $aN+b$, the algorithm never takes more time than this for *any* input, so the upper bound is $O(N)$.

The best case for simple search is that x is the first element of the sequence. This is down to luck, but it is still *possible* for the algorithm to finish after checking just this one element. The time taken in this case is independent of the length of the sequence, so it is constant for all N. Let's call it c. This possibility still exists for any input, so the lower bound is $\Omega(1)$.

In order to give a tight bound, there now does have to be a single function $g(N)$ that provides both an upper *and* a lower bound for the running time of the algorithm (within a constant factor). If you are considering a single case, like worst case, then this will be possible. If you are considering any input, it does not always exist. In this case, since the time complexity has different upper and lower bounds there is no tight bound. Neither $T(N) = \Theta(N)$ nor $T(N) = \Theta(1)$ are correct.

2. (Cormen Exercise 3.1-4). The formal definition of O is:

$$O(g(N)) = \left\{ f(N) \mid \begin{array}{l} \text{there are positive constants } c \text{ and } N_0 \\ \text{such that } 0 \le f(N) \le c\,g(N) \text{ for all } N \ge N_0 \end{array} \right\}.$$

Using this definition, show whether each of the following statements is true or false.

(a) $2^{N+1} = O(2^N)$

(b) $2^{2N} = O(2^N)$

Answer: (a) is true. Instantiating the set condition, we need to show that there are positive c and N_0 such that the inequalities $0 \leq 2^{N+1} \leq c2^N$ are true for all $N \geq N_0$. Firstly, $2^{N+1} \geq 0$, for all N, so this is true for all N greater than any N_0. Next, we need to solve $c2^N \geq 2^{N+1}$ for c:

$$\begin{aligned} c2^N &\geq 2^{N+1} && \text{, so (since } 2^N \text{ is positive)} \\ c &\geq \frac{2^{N+1}}{2^N} && \text{, and so} \\ c &\geq 2. \end{aligned}$$

So, $c2^N \geq 2^{N+1} \geq 0$ when $c \geq 2$, for all N. So, for say $c = 2$ and $N_0 = 1$, the $O(2^N)$ condition is true. Therefore, such a c and N_0 exist and (a) is true.

(b) is false. We need to show that there are positive c and N_0 such that $0 \leq 2^{2N} \leq c2^N$ for all $N \geq N_0$. As before, $2^{2N} \geq 0$, for all N. Now, solving $c2^N \geq 2^{2N}$ for c:

$$\begin{aligned} c * 2^N &\geq 2^{2N} && \text{, so} \\ c &\geq \frac{2^{2N}}{2^N} && \text{and} \\ c &\geq 2^N. \end{aligned}$$

There are no constants c and N_0 such that $c \geq 2^N$ for all $N \geq N_0$.

3. If a_0 and a_1 are constants, and a_2 is a positive constant, show that $a_2N^2 + a_1N + a_0$ is not in $O(N)$. What are the consequences for algorithm design? What are the limitations of these consequences?

Answer: Instantiating the $O(N)$ set condition, we need to show that there are positive c and N_0 such that the inequalities $0 \leq a_2N^2 + a_1N + a_0 \leq cN$ are true for all $N \geq N_0$. We expect the second inequality to fail, so we will start there. We need to solve $a_2N^2 + a_1N + a_0 \leq cN$ for c:

$$\begin{aligned} cN &\geq a_2N^2 + a_1N + a_0 && \text{, iff (since } N \text{ is positive)} \\ c &\geq a_2N + a_1 + \frac{a_0}{N}. && \text{1 mark} \end{aligned}$$

Now

$$\begin{aligned} \lim_{N\to\infty} a_2N &= \infty && \text{, and} \\ \lim_{N\to\infty} a_1 &= a_1 && \text{, and} \\ \lim_{N\to\infty} \frac{a_0}{N} &= 0 && . \end{aligned}$$

So

$$\lim_{N\to\infty} (a_2N + a_1 + \frac{a_0}{N}) = \infty + a_1 + 0 = \infty, \quad \text{1 mark}$$

and so there are no c and N_0 that satisfy the set condition.

In terms of algorithm design, this means that any algorithm with a time complexity that has an N^2 term will take longer than any algorithm whose time complexity has a highest order term aN, when N is sufficiently large. So, if the best algorithm we have has N^2 time complexity, then we know that it is worth trying to find a "linear algorithm", because it should be a faster solution.

The limitation is what the exact value of "large N" turns out to be. (i.e. N_0 in the definition of O.) Depending on the constants involved, the size of the input for which the $O(N)$ algorithm becomes faster might be impractically large. It must become faster for some N, but if this N is larger than the largest input you are going to need to handle, or simply larger than the average input, it will be of no use.