Putting bounds on functions and runtimes

§**1. General setting.** Consider a function $F : \mathbb{N} \to \mathbb{R}$ defined by some means. Let's note in passing that the definition does not fix any method of computing $F(n)$ given n it simply fixes a unique value by appropriate conditions[1]. We say that a function $U : \mathbb{N} \to \mathbb{R}$ is an *upper bound* for F if $F(n) \le U(n)$ for all n. Similarly a function $L : \mathbb{N} \to \mathbb{R}$ is a *lower bound* for F if $L(n) \le F(n)$ for all n. In many situations we don't mind if the inequalities fail to hold for some initial values as long as they hold for all large enough values of n (recall that this is the same as saying that there is an $n_0 \in \mathbb{N}$ such that the inequality holds for all $n \ge n_0$). So with this situation a lower and upper bound give us the information that

$$L(n) \le F(n) \le U(n) \qquad (\dagger)$$

for all large enough values of n. Note that the definition of a lower bound is entirely symmetrical to that of an upper bound, the only difference is the inequality.

There can be various reasons for wanting to find upper and lower bounds, for us the main one is because although we have a precise definition of F we cannot obtain an exact formula for it. If we have upper and lower bounds we are of course interested in how good they are. To illustrate the point suppose we are seeking to find information about some integer M. After some work we find that $0 \le M \le 1000000$. This certainly gives us information but of a rather imprecise nature because the lower and the upper bounds are very far apart. If we worked a bit more and found that $12 \le M \le 20$ we'd be in a better position. In terms of functions we want $L(n)$ and $U(n)$ to be as close as possible. Of course the ideal is that $L(n) = U(n)$ in which case we know $F(n)$ but this is often not possible so we must settle for something less precise[2].

There are two further refinements we can make, we will consider the first one here and the second one later on. We are interested in putting bounds on functions whose values are non-negative and what is of interest is to bound the size of their values from above and below. So, e.g., if it so happens that (unknown to us) $F(n) = n$ it is not informative to produce the lower bound $-n^{10} \le F(n)$; anything negative is a lower bound. For this reason we amend (†) to

$$0 \le L(n) \le F(n) \le U(n) \qquad (\ddagger)$$

for all large enough values of n. (An alternative is to take absolute values throughout but this is notationally heavier and is essentially equivalent to our approach.)

§**2. Putting bounds on runtimes.** Suppose we have an algorithm $\mathcal{A}$. Recall that we assume we have a method of assigning a size to the inputs of $\mathcal{A}$ such that for

[1]A frequent error is to talk about the runtime of a mathematical function. This is meaningless until we have chosen an algorithm to compute it. In any case it can be proved that for most functions there is no algorithm to compute them. (This is surprisingly easy given some fundamental notions.)

[2]In fact even when we know F explicitly we might be interested in bounding it from above and below by functions that are easier to work with and are still close enough to F.

a given size there are only finitely many possible inputs. Now let R_n denote the set of runtimes that result by running $\mathcal{A}$ on all inputs of size n. (Of course this set depends on $\mathcal{A}$ as well but we have not indicated this in the notation just to keep it simple, no confusion can arise as we will only be considering $\mathcal{A}$ in the discussion.) Since there are only finitely many inputs to $\mathcal{A}$ for any given n it follows that R_n is a finite set. Recall that we made the following definitions in the course:

(1) The *worst case runtime* of $\mathcal{A}$ is the function $W : \mathbb{N} \to \mathbb{R}$ defined by

$$W(n) = \max R_n.$$

(2) The *best case runtime* of $\mathcal{A}$ is the function $B : \mathbb{N} \to \mathbb{R}$ defined by

$$B(n) = \min R_n.$$

(We are not terribly interested in the best case runtime except for illustrative purposes.) It is trivially the case that

$$0 \leq B(n) \leq U(n).$$

In other words $B(n)$ is a lower bound for $U(n)$ and of course $U(n)$ is an upper bound for $B(n)$. However these can be too far apart to be of much use. For example we saw that of linear search the best case runtime is a constant while the worst case is proportional to n. You should also consider the best and worst case runtimes of insertion sort, again they are far apart. Of course there are algorithms for which $B(n) = W(n)$, the point is that this is not even remotely true of all of them.

Now pick any $r \in R_n$. By the definition of our two functions we have

$$B(n) \leq r \leq W(n).$$

Spelling this out, if we pick *any* input to $\mathcal{A}$ of size n and find the runtime then this is an upper bound for the best case runtime and also a lower bound for the worst case. Once again note the symmetry.

Let us now focus on $W(n)$ for some fixed n. By definition an upper bound $U(n)$ to $W(n)$ is anything that satisfies

$$\max R_n \leq U(n).$$

Note that we just demand a bound on a *single* element of the set R_n but as this is the largest element it follows automatically that it is an upper bound to *all* elements of R_n. So when putting an upper bound we are in fact just concerned with one element of R_n but we don't know its value. So how can we put an upper bound on it? Well if we find that for *all* inputs of size n the runtime is at most $U(n)$ then of course this claim is also true of $\max R_n$, i.e., of $W(n)$. In practice we consider a general input of size n and argue from the pseudocode that the algorithm runtime will be at most a certain function of n (e.g., if we have a loop that is executed at most n times and the body costs at most a constant c

then the cost of the entire loop is at most cn). This process leaves open the possibility that we have overestimated by a significant amount so to check this we look for lower bounds to $W(n)$.

Let us now consider putting a lower bound on $W(n)$. By definition a lower bound $L(n)$ to $W(n)$ is anything that satisfies

$$0 \le L(n) \le \max R_n.$$

Once again our interest is in putting a bound on a single element of R_n. We could do this by considering all inputs of size n but for most algorithms this would lead to a severe under estimate (recall linear search). As observed above, if we find the runtime r for any particular input of size n then we have found a lower bound to $\max R_n$, i.e., to $W(n)$. So when trying to put a lower bound we look at the structure of the algorithm and try to identify an input of size n that will make it do the *greatest* amount of work.

There is now an asymmetry in what we do. However this arises from the definition of the function which we seek to bound, *not* from the notion of upper and lower bounds.

At this point you should consider what it means to put upper and lower bounds on $B(n)$. It should be clear that in putting a lower bound we consider all inputs of size n but for an upper bound we need only consider a single appropriate input of size n. In this second case we look at the structure of the algorithm and and try to identify an input of size n that will make it do the *least* amount of work. Thus the situation is a mirror symmetry of that for $W(n)$.

§**3. Asymptotic notation.** We discussed above one refinement to the notion of bounds. For the second refinement we allow the possibility of adjusting the values of U and L by some strictly positive multiplicative constants, i.e., we focus on growth rates. This can be for various reasons, e.g., the definition of F involves unknown constants as in the case of runtimes of algorithms. So we amend the inequalities (‡) in the definition to allow the use of constants $c_1 > 0$ and $c_2 > 0$ s.t.

$$0 \le c_1 L(n) \le F(n) \le c_2 U(n),$$

for all large enough values of n. Note that this says *exactly* the same as that $U = \Omega(L)$ and $F = O(U)$. Once again we ask just how good are such bounds? The best we could hope for is that $L = U$ and if this is so then we say that $F = \Theta(U)$. Of course even if this is so we do not know the value of $F(n)$, for large enough n, because of the multiplicative constants but we do have a very good idea of the growth rate. For the final time, the definitions of O and Ω are entirely symmetrical, any asymmetry in arguments involving them comes from the nature of the functions being studied.