Tutorial 2: Discussion Questions

Exercise 1: Induction Proofs

1.

Given:

$$T(1) = a \tag{1}$$

$$T(n) = T(n/2) + c \tag{2}$$

To prove:

$$T(n) \in O(logn) \tag{3}$$

Another way to think about $T(n) \in O(logn)$ is $T(n) \leq q \times logn$, where q is some constant. Therefore, proving

$$T(n) \leq q \times logn \tag{4}$$

is equivalent to proving $T(n) \in O(logn)$.

Note: It is important to point out that the question specifically requires us to use induction and therefore, any other method (regardless of its correctness) would be considered invalid.

As you may remember from Tutorial 1, formulating an Inductive proof is a 3 step process: Base case, Hypothesis and the Inductive step.

Before we dive into it into the technicalities, let's talk about the base case. If you recall, base case is meant to be a trivial case for which the 'thing' we are trying to prove (here, eq. 4) is true. An obvious trivial case would be $n = 1$ i.e. for base case we want **to show**

$$T(1) \leq q\ log1$$

$$\Rightarrow T(1) \leq 0$$

From eq. 1, we know $T(1) = a$ but how do we show $a \leq 0$?

For starters, it is important to stress that a is a constant. It has a fixed value and it is not a range i.e. $a \leq 0$ does not imply a represents all numbers from $-\infty$ to 0. Secondly, we want to show $a \leq 0$, we don't have that yet.

Since a is a constant and the question doesn't state its value we could argue, we can never show $a \leq 0$. Alternatively, we could argue there is nothing special about a and it is *some* constant and therefore, it may as well be 0 (in which case $a \leq 0$ is in fact true). Regardless, the point I am trying to make here is, depending on how you like to do things, there may not a lot of *meaning* in "proving" the base case in our example or you may not find it very *satisfying*.

With that out of the way, let's discuss the remaining two steps required to formulate an inductive proof.

In this case, for the hypothesis, you would write something like: Let $T(k) \in O(logk)$ for some constant k, where $k > 1$ and then for the inductive step you would want **to show** $T(k+1) \in O(\log(k+1))$ somehow. However, it is surprisingly non-trivial to show this.

Therefore, rather than showing $T(n) \in O(logn)$ is true for **all** $n \in \{1, 2, 3 \ldots\}$, for now, let us consider the cases where $n \in \{1, 2, 4, 8, 16, 32 \ldots\}$ i.e. n is some power of 2 (or $n = 2^y$ where $y \geq 0$).

Why?

a) Powers of 2 work well with log (where the base is 2).
b) We will prove the general case for all values of n shortly.

In other words,

Prove that $T(2^y) \in O(log2^y)$ using induction.

Proof:

Step 1: Base case

$$y = 0$$

$$T(2^0) = T(1)$$

We have discussed this above.

Step 2: Hypothesis

Let $T(2^m) \in O(log2^m)$ for some constant m, such that $m \geq 0$. By definition of Big-O, we know $T(2^m) \leq dlog2^m$ where d is some positive constant. Since $log2^m = mlog2 = m$, we have

$$T(2^m) \leq dm \tag{5}$$

It is important to note that d can be **any** positive constant that **we** pick. **We** get to decide its value. Therefore, **I** choose d such that

$$d \geq c \tag{6}$$

(the same c as in eq. 2). You'll see why shortly.

Step 3: Inductive step

We need to show, $T(2^{m+1}) \in O(log2^{m+1})$.

Consider,

$$T(2^{m+1})$$

From eq. (2), we know,

$$T(2^{m+1}) = T(2^m) + c$$

From eq. (5) we know $T(2^m) \leq dm$. Therefore,

$$T(2^{m+1}) \leq dm + c$$

We also know $c \leq d$ from eq. (6). Therefore,

$$T(2^{m+1}) \leq dm + d$$

$$\Rightarrow T(2^{m+1}) \leq (m+1)d$$

$$\Rightarrow T(2^{m+1}) \leq d(m+1) \times log2$$

$$\Rightarrow T(2^{m+1}) \leq dlog2^{m+1}$$

$$\Rightarrow T(2^{m+1}) \in O(log2^{m+1})$$

Hence, $T(2^y) \in O(log2^y)$ (proved via induction).

Now that we have established that, $T(n) \in O(logn)$ for all $n \in \{1, 2, 4, 8 \ldots\}$, let's discuss how we can prove it for all $n \in \{1, 2, 3, 4, 5 \ldots\}$.

(from hereon, n can be any natural number ≥ 1, but remember with Big-O we only really care about $n \to \infty$)

To give you an intuition on how we could do so, let's first establish something. Looking back at eq. (2) we know $T(n) = T(n/2) + c$. You can see that $T(n)$ is non-decreasing if n increases i.e. $T(10) \leq T(100)$, $T(61) \leq T(64)$ etc. "obviously".

Now, for any number n, let's consider some power of 2 that is greater than or equal to n. For e.g. if $n = 31$, some power of 2 greater than or equal to n is 32 (or 64 or 128 whatever). More concretely, let's try to find the **some number** b such that

$$n \le 2^b \tag{7}$$

$$\Rightarrow logn \le log2^b$$

$$\Rightarrow logn \le blog2$$

$$\Rightarrow logn \le b$$

Let's say, $b = \lfloor \text{logn} \rfloor + 1$ (as it is consistent with $logn \le b$).

Since $n \le 2^b$, based on what we discussed above | ref. $T(61) \le T(64)$,

$$T(n) \le T(2^b)$$

We proved above that, $T(2^y) \in O(log2^y)$ using induction i.e.

$$T(2^y) \le dlog2^y \Rightarrow T(2^y) \le dy$$

Therefore, $T(2^b) \le db$.

$$\Rightarrow T(n) \le T(2^b) \le db$$

$$\Rightarrow T(n) \le db$$

And $b = \lfloor \text{logn} \rfloor + 1$

$$\Rightarrow T(n) \le d(\lfloor logn \rfloor + 1)$$

$\lfloor \text{logn} \rfloor$ by definition is less than or equal to $logn$

$$\Rightarrow T(n) \le d(logn + 1)$$

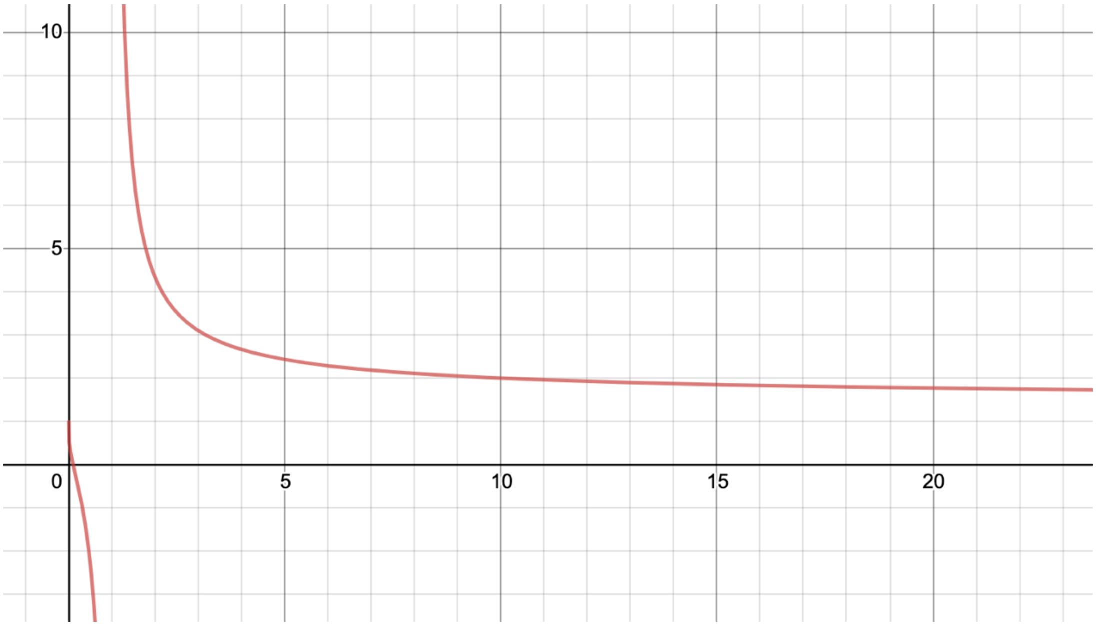
$$\Rightarrow T(n) \le dlogn + d \tag{8}$$

In order to prove $T(n) \in O(logn)$, we need to show $T(n) \le (some\ constant) \times logn$. We're close, but not quite there yet. We need to get rid of the (… $+d$) part from the equation above somehow. If we can show that $dlogn + d \le zlogn$, for some constant z, then we can show $T(n) \le zlogn$ and we'd be done.

Let's assume **for a second** that $dlogn + d \le zlogn$

$$\frac{d(logn + 1)}{logn} \le z$$

If you look at the curve $y = \frac{(logn+1)}{logn}$,



We can see that for $n \geq 5$ (note: **all** values of n that are ≥ 5),

$$\frac{(logn + 1)}{logn} < 3$$

$$\Rightarrow \frac{d(logn + 1)}{logn} < 3d$$

Therefore, if $z = 3d$ then

$$z \geq \frac{d(logn + 1)}{logn}$$

will in fact be true for all $n \geq 5$, which means

$$dlogn + d \leq zlogn$$

Note: This is something I said, "let's assume" above. But now as long as $n \geq 5$ and $z = 3d$, we know this is **in fact true**.

"Why $n \geq 5$ and $z = 3d$? This looks like a hack!"
If you think back to the whole point of Big-O complexity, z can be any constant (why not $3d$?) and we only care about very large values of n (or specifically as $n \to \infty$) so $n \geq 5$ works!

Coming back to our eq. (8)

$$T(n) \leq dlogn + d$$

$$\Rightarrow T(n) \leq dlogn + d \leq zlogn$$

$$\Rightarrow T(n) \leq zlogn$$

Therefore, $T(n) \in O(logn)$.

Phew!

2.

This question is phrased a bit funny. It is under "Induction Proofs" sections but doesn't really ask us to *prove* anything. Just *find* something.

Given:

$$F(1) = F(2) = 1$$

$$F(n) = F(n-1) + F(n-2) \tag{1}$$

$$F(n) \in O(a^n)$$

Goal: Find the smallest value of a.

As $F(n) \in O(a^n)$, we know

$$F(n) \leq c \times a^n \tag{2}$$

where c is some **constant > 0**.

Also, if $F(n) \in O(a^n)$, then $F(n-1) \in O(a^{n-1})$ which implies

$$F(n-1) \leq c \times a^{n-1} \tag{3}$$

Similarly,

$$F(n-2) \leq c \times a^{n-2} \tag{4}$$

Add eq. (3) and eq. (4)

$$F(n-1) + F(n-2) \leq ca^{n-1} + ca^{n-2} \tag{5}$$

From eq. (1) we have

$$F(n) = F(n-1) + F(n-2)$$

We know, L.H.S (left hand side) is $\leq ca^n$ from eq. (2) and we also know R.H.S is $\leq ca^{n-1} + ca^{n-2}$ from eq. (5). Since we have an upper-bound for L.H.S and R.H.S and L.H.S = R.H.S, if we equate the two bound constraints, we can solve for a.

More concretely,

$$ca^n = ca^{n-1} + ca^{n-2}$$

Divide the equation by ca^{n-2} (because $c \neq 0$ and a^{n-2} cannot be 0).

$$\Rightarrow a^2 = a + 1$$

Solve for a using the quadratic formula

$$a = \frac{1+\sqrt{5}}{2}, \frac{1-\sqrt{5}}{2}$$

Now we want the *smallest* value of a. Therefore, you might be inclined to pick $\frac{1-\sqrt{5}}{2}$. However, note that $\frac{1-\sqrt{5}}{2}$ is actually a negative number and $O(\frac{1-\sqrt{5}}{2}^n)$ doesn't actually make sense. Think "time complexity" and "negative time".

Therefore, $a = \frac{1+\sqrt{5}}{2}$.

Exercise 2: Approximation Algorithms

1

Generate a uniform random number in [0,1] and if the values < 0.25, return the inverted sign or else return the correct sign.

2

Evaluations on the sign can be made using 'Bad Sign'. There is a 75% chance that 'bad sign' being correct. Therefore, with majority voting when n is large (say 100), then the 'better sign' could be more accurate.

3

If 'bad sign' is incorrect 49% of the time, we can increase the number of votes and use majority voting to obtain better approximation. In the case where 'bad sign' is incorrect 51%, then when n is large it is 51% incorrect. In this case we can flip the sign returned by 'bad sign' and make the output 49% accurate.