# CS 170 Dis 0

### Released on 2017-08-27

## 1 Asymptotic Bound Practice

Prove that for any $\epsilon > 0$ we have $\log x \in O(x^\epsilon)$.

**Solution:**

Observe that $x > \log x \forall x > 0$. We can see this by taking finding the minimum of the function $x - \log x$ over the range $(0, \inf)$ using some calculs (find the critical points, then check concavity). The minimizing $x$ is 1, with value 1.

If $x > \log x$, then we have that $\log x^\epsilon < x^\epsilon$, and therefore $\epsilon \log x < x^\epsilon$. It follows that a constant factor times $x^\epsilon$ is always larger than $\log x$ for $x > 0$. This proves $\log x \in O(x^\epsilon)$.

Here is an alternate argument, using l'Hopital's rule:

$$\begin{aligned}
\lim_{x\to\infty} \frac{\log x}{x^\epsilon} &= \lim_{x\to\infty} \frac{\frac{d}{dx}\log x}{\frac{d}{dx}x^\epsilon} \\
&= \lim_{x\to\infty} \frac{\frac{1}{x}}{\epsilon x^{\epsilon-1}} \\
&= \lim_{x\to\infty} \frac{1}{\epsilon x^\epsilon} = 0
\end{aligned}$$

And so therefore $\log x \in O(x^\epsilon)$.

## 2 Bounding Sums

Let $f(\cdot)$ be a function. Consider the equality

$$\sum_{i=1}^{n} f(i) \ \in \ \Theta(f(n)),$$

Give a function $f_1$ such that the equality holds, and a function $f_2$ such that the equality does not hold.

**Solution:** There are many possible solutions.
$f_1(i) = 2^i$: $\sum_{i=1}^{n} 2^i = 2^{n+1} - 2 \in \Theta(2^n)$.
$f_2(i) = i$: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \in \Theta(n^2) \neq \Theta(n)$.

## 3 In Between Functions

Prove or disprove: If $f : \mathbb{N} \to \mathbb{N}$ is any positive-valued function, then either (1) there exists a constant $c > 0$ so that $f(n) \in O(n^c)$, or (2) there exists a constant $\alpha > 1$ so that $f(n) \in \Omega(\alpha^n)$.

**Solution:** Let $f(n) = 2^{\sqrt{n}}$. $f(n) \in \Omega(n^c)$ for any constant $c > 0$ and the best case is asymptotically slower than $n^c$. $f(n) \in O(\alpha^n)$ for any constant $\alpha > 1$ and the worst case is asymptotically faster than $\alpha^n$.

As a side note, this shows that there are algorithms whose running time grows faster than any polynomial but slower than any exponential. In other words, there exists a nether between polynomial-time and exponential-time.

## 4 Recurrence Relation Practice

Derive an asymptotic *tight* bound for the following $T(n)$. Cite any theorem you use.

(a) $T(n) = 2 \cdot T(\frac{n}{2}) + \sqrt{n}$.

**Solution:** Master theorem: $a = 2, b = 2, d = 1/2$. So that $d < \log_b a = 1$: $T(n) = \Theta(n)$

(b) $T(n) = T(n-1) + c^n$ for constants $c > 0$.

**Solution:** Expanding out the recurrence, we have $T(n) = \sum_{i=0}^{n} c^i$.

By the formula for the sum of a partial geometric series, for $c \neq 1$: $T(n) := \sum\limits_{i=0}^{n} c^i = \frac{1-c^{n+1}}{1-c}$. Thus,

- If $c > 1$, for sufficiently large $n$, $c^{n+1} > c^{n+1} - 1 > c^n$. Dividing this inequality by $c-1$ yields: $\frac{c}{c-1}c^n > T(n) > \frac{1}{c-1}c^n$. Thus, $T(n) = \Theta(c^n)$, since $\frac{1}{c-1}$ is constant.
- If $c = 1$, then every term in the sum is 1. Thus, $T(n) = n + 1 = \Theta(n)$.
- If $c < 1$, then $\frac{1}{1-c} > \frac{1-c^{n+1}}{1-c} = T(n) > 1$. Thus, $T(n) = \Theta(1)$.

(c) $T(n) = 2T(\sqrt{n}) + 3$, and $T(2) = 3$.

**Solution:** The recursion tree is a full binary tree of height $h$, where $h$ satisfies $n^{1/2^h} = 2$. Solving this for $h$, we get that $h = \Theta(\log\log n)$. The work done at every node of this recursion tree is constant, so the total work done is simply the number of nodes of the tree, which is $2^{h+1} - 1 = \Theta(\log n)$, so $T(n) = \Theta(\log n)$.