Harvard University
Computer Science 121

Section Handout 9

1 Concept Review

1.1 Asymptotic Notation

Definition 1.1 (Big-O) *For functions $f, g : \mathbb{N} \to \mathbb{R}^+$, $f(n) = O(g(n))$ if there exists c, N, such that for all $n > N$, $f(n) \leq c \cdot g(n)$. We can also write this as $g(n) = \Omega(f(n))$.*

The intuition to have about these is that $f = O(g)$ means "$f \leq g$", ignoring constant factors, for large enough n. Correspondingly $g = \Omega(f)$ means "$g \geq f$", ignoring constant factors, for large enough n. These two statements are equivalent.

Definition 1.2 (Theta) *For functions $f, g : \mathbb{N} \to \mathbb{R}^+$, $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.*

This intuitively corresponds to $f = g$, ignoring constant factors, for large enough n.

Definition 1.3 (Small-O) *For functions $f, g : \mathbb{N} \to \mathbb{R}^+$, $f(n) = o(g(n))$ if for* any *$c > 0$, there exists N, such that $n > N \Rightarrow f(n) < c \cdot g(n)$. We can also write this as $g(n) = \omega(f(n))$.*

The intuition for little-o is that if $f(n) = o(g(n))$, $f < g$, again ignoring constant factors, and for large enough n.

These definitions let us compare the running time of various algorithms in a way somewhat reminiscent to the way we used mapping reductions to reason about the comparative difficulty of deciding and recognizing languages.

1.2 Time Complexity

$$\text{TIME}(t(n)) = \{L : L \text{ is decided by a deterministic Turing machine that runs in } O(t(n))\}.$$

Does using a deterministic machine matters in this definition?
Does having a polynomial time algorithm guarantee solving the problem efficiently?

$$\mathcal{P} = \bigcup_k \text{TIME}(n^k).$$

When we say $\langle M \rangle$ we mean a reasonable encoding of a Turing machine. When talking about decidability / recognizability, the representation often does not matter. Polynomial time algorithms, however, are on the size of the input. So, the size of your input matters, i.e. it matters how you represent inputs.

1. What is a reasonable way to represent numbers?
2. What about graphs?

2 Exercises

Exercise 2.1 *Which of the following relations hold?*

1. $64n + 2^5 = o(3n^4)$
2. $12\log_2(n) = o(2\log_4(n^2))$
3. $5n = \Omega(6n)$
4. $2^n n^4 = \Theta(2^n)$
5. $\omega(n^2) = O(2^n)$
6. $\log_2(n)\Theta(2^n) = \omega(n^2)$

2.1 Solution

1. True. $\lim_{n\to\infty} \frac{64n+2^5}{3n^4} = 0$
2. False. $2\log_4(n^2) = 4\log_4(n) = 2\log_2(n)$, which is proportional to $12\log_2(n)$.
3. True. $c = 1/2$, and $5n \geq (1/2)6n$ for all $n \geq 1$.
4. False. For any c, no matter how large, with $n = c$ we will have $c^4 * 2^c > c * 2^c$, so we cannot bound it by a constant.
5. False. 3^n is $\omega(n^2)$ but not $O(2^n)$.
6. True. $\lim_{n\to\infty} \frac{n^2}{c*log_2(n)2^n} = 0$

Exercise 2.2 *Give a proof or counterexample for the following claims:*

1. *If* $f = o(g)$ *then* $f = O(g)$.
2. *If* $f \neq O(g)$ *then* $g = O(f)$.
3. *If* $f = O(g)$, *and* $g = \Theta(h)$, *then* $h = \Omega(f)$
4. *If* $f = O(g)$, *and* $h = O(g)$, *then* $f = \Theta(h)$

2.2 Solution

1. True. $f = o(g)$ means for all c there exists an N such that $f(n) < c * g(n)$ for all $n \geq N$. Therefore take any c, say $c = 1$, and then there will be an N such that $f(n) \leq c * g(n)$ for all $n \geq N$.
2. True. If there is no c such that as n gets large $f(n) \leq c * g(n)$, then $\lim_{n\to\infty} g(n)/f(n) = 0$. This means $g = o(f)$, and so $g = O(f)$ as well.
3. True. There exists some c_1, N_1 such that $f(n) \leq c_1 * g(n)$ for $n \geq N_1$. Also because $g = O(h)$ there exists some c_2, N_2 such that $g(n) \leq c_2 * h(n)$ for all $n \geq N_2$. This means to show $f = O(h)$ (equivalent to $h = \Omega(f)$) we can see $f(n) \leq c_1 * c_2 * h(n)$ for all $n \geq \max(N_1, N_2)$

4. False. $f = n$, $h = n^2$, $g = n^3$.

Exercise 2.3 *Recall that a TM computes a function f if on input w it halts with $f(w)$ on the tape. Suppose we want a turing machine that computes F such that $F(\langle n \rangle) = \langle F_n \rangle$, where is the F_n is the nth fibonacci number. Show that this can't be done in polynomial time if the same encoding is used for $\langle n \rangle$ and $\langle F_n \rangle$.*

Exercise 2.4 *Consider an arcade game where the Pac-Man starts in the $(1, 1)$ (top left) cell of an n by n matrix, and at each step can move either downward or rightward. When Pac-Man enters cell (i, j), it collects $M_{i,j}$ coins. Give a polynomial-time algorithm that, given matrix M, computes a path for Pac-Man to collect as many coins as possible.*

Solution. Let $s(i, j)$ be the maximum number of coins that can be collected when Pac-Man reaches cell (i, j). Then s can be defined recursively as follows:

$$s(i,j) = \begin{cases} \max\{s(i-1,j), s(i,j-1)\} + M_{i,j} & (i \geq 1, j \geq 1) \\ 0 & (i = 0 \text{ or } j = 0) \end{cases}$$

where we extend the domain of s to include $i = 0$ or $j = 0$, for convenience. Thus we can use dynamic programming to compute $s(i, j)$ for all i, j, e.g. proceed by the rows. Once we have computed all $s(i, j)$, we can recover the optimal path by tracing backwards from (n, n); at each cell (i, j), we move upward or leftward, depending on which of $s(i - 1, j)$ and $s(i, j - 1)$ is larger.