CMPT 307 (Fall 2019)

1 Growth of Functions (Chapter 0 of the text).

We note that analyses of algorithms being

- too precise is problematic, and
- too sloppy is unacceptable.

There is a need for an analytical tool that is based on the right simplifications. We are interested in determining the growth rate of the running times of algorithms as the problem size gets larger and larger to infinity. Consider the following examples.

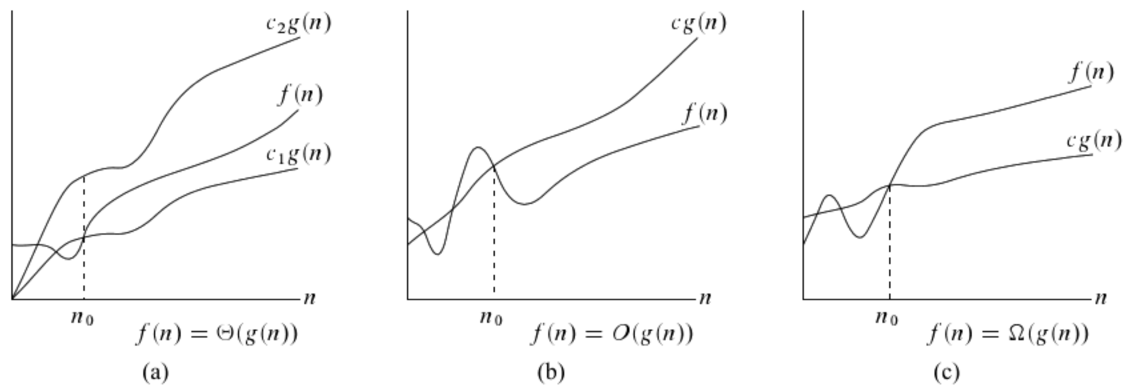
- $f(n) = n^2$ has quadratic growth rate, i.e. if the problem size n is doubled, the running time is increased four times.
- $f(n) = 100n^2$ also has quadratic growth rate.
- $f(n) = 100n^2 + 12n$ also has quadratic growth rate for large n (asymptotically).

All the three examples above can be grouped together to say that each one of them has $O(n^2)$ running time (growth rate).

Let $f, g : \mathbb{N} \to \mathbb{N}$ be two increasing (positive) functions where $\mathbb{N}$ is the set of positive integers. The following asymptotic notations are commonly used in the analysis.

$O(g(n))$	$=$	$\{ f(n) \mid$ growth rate of $f(n)$ is no more than that of$g(n)$ $\}$
	$=$	$\{ f(n) \mid \exists\, c, n_0 > 0,\ \forall\, n \geq n_0\ f(n) \leq c\, g(n) \}$
$\Omega(g(n))$	$=$	$\{ f(n) \mid$ growth rate of $f(n)$ is at least that of$g(n)$ $\}$
	$=$	$\{ f(n) \mid \exists\, c, n_0 > 0,\ \forall\, n \geq n_0\ f(n) \geq c\, g(n) \}$
$\Theta(g(n))$	$=$	$\{ f(n) \mid$ growth rate of$f(n)$ is the same as that of$g(n)$ $\}$
	$=$	$\{ f(n) \mid \exists\, c_1, c_2, n_0 > 0,\ \forall\, n \geq n_0\ c_1 g(n) \leq f(n) \leq c_2\, g(n) \}$

Note that $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.



1.1 Alternate definitions

Let $f(n)$ and $g(n)$ be the functions as defined above. Let

$$lim_{n\to\infty}\frac{f(n)}{g(n)} = c.$$

We can then show that

- $0 < c < \infty$ if and only if (iff) $f(n) \in \Theta(g(n))$.
- $c = 0$ iff $f(n) \in O(g(n))$.
- $c = \infty$ iff $f(n) \in \Omega(g(n))$.

Example: Show that $\log_b n \in \Theta(\log_2 n)$ for any $b > 1$.

Note that (change of basis) $\log_b n = \frac{\log_2 n}{log_2 b}$. We now get the result easily through the application of limits.

In order take the limit when $lim_{n\to\infty} f(n)$ and $lim_{n\to\infty} g(n)$ are either both zero or both infinity, the following rule called the **L'Hospital Rule** is very convenient. It says that

$$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{f'(n)}{g'(n)}.$$

Here $f'(n)$ is the derivative of $f(n)$ with respect to n. This process is repeated if $lim_{n\to\infty} f'(n)$ and $lim_{n\to\infty} g'(n)$ are either both zero or both infinity as well.

Example: Show that $n\sqrt{n} \in \Omega(n \log_b n)$ for any positive constant b.

Example: Show that $\sum_{i=1}^{n} i^2 \in \Theta(n^3)$.

Example: Show that $\log(n!) \in \Theta(n \log n)$. Note that there is no mention of the base to the logarithm. The statement is true for any positive constant base.

1.2 Approximation by integrals

The last two problems can be solved using the method of integration. Let us solve the last example.

We want to show that $\log n! \in \Theta(n \log n)$.

We can write $\log n! = \log[n(n-1)(n-2)\ldots 3.2.1] = \sum_{i=1}^{n} \log i = \sum_{i=2}^{n} \log i$, since $\log 1 = 0$. We now show that

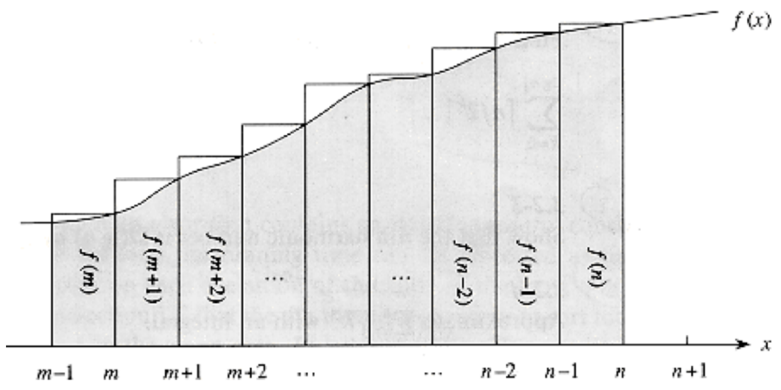
- $\sum_{i=1}^{n} \log i \geq c_1 \times n \log n, \forall\, n \geq n_1$ for some positive constants c_1 and n_1.
- $\sum_{i=1}^{n} \log i \leq c_2 \times n \log n, \forall\, n \geq n_2$ for some positive constants c_2 and n_2, and

Consider the following figure where $f(x)$ is an arbitrary increasing function defined over the range $[m-1..\infty]$. We can write (why?)

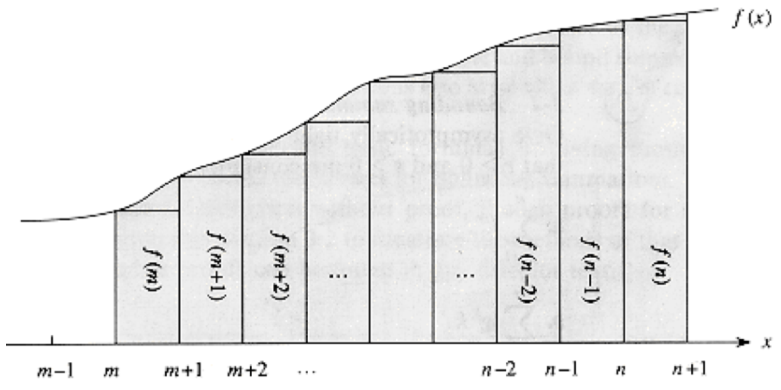
$$\int_{m-1}^{n} f(x)dx \leq \sum_{i=m}^{n} f(i) \quad [Figure(a)]$$

and

$$\sum_{i=m}^{n} f(i) \leq \int_{m}^{n+1} f(x)dx \quad [Figure(b)]$$



(a)



(b)

For our problem, $m = 2$ and $f(x) = \log x$. Therefore, we have established that

$$\int_1^n \log x dx \leq \sum_{i=2}^{n} \log i \;\; [Figure(a)]$$

$$\sum_{i=2}^{n} \log i \leq \int_2^{n+1} \log x dx \;\; [Figure(b)]$$

We know that

$$\int \log_e x \, dx = x \, \log_e x - x.$$

The rest of the stuffs should be easy to show that $\log n! \in \Theta(n \log n)$.

Using this approach we can show (you should try) that for any integer $k \geq 1$,

$$\sum_{i=1}^{n} i^k \in \Theta(n^{k+1}).$$

1.3 Some common running times of programs

The following table lists some of more common running times for programs and their informal names. Note that $O(1)$ is a shorthand for "some constant".

Big-Oh	Informal name
$O(1)$	constant
$O(\log n)$	logarithmic
(n)	linear
$O(n \log n)$	n log n
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(2^n)$	exponential

1.4 Logarithms in running times

From integral calculus, we know that $\log_e n = \int_1^n \frac{1}{x} dx$. It frequently appears in the analyses of divide-and-conquer algorithms. Computer scientists generally think of "$\log n$" as meaning $\log_2 n$, rather than $\log_e n$ or $\log_{10} n$. Note that the value of $log_2 n$ is the number of times n can be divided by 2 to get down to 1. When $n = 2^k$, $\log_2 n = k$.

1.5 An useful reference

I find the following article very useful. This should be read and consulted in case you need some ideas on how to analyze the running times of algorithms.

The Running Time of Programs by J. Ullman
`infolab.stanford.edu/~ullman/focs/ch03.pdf`

1.6 Practice Questions

1. Problems 0.1, 0.2, 0.3 from the text (page 9-10).

2. Consider the following piece of code:

   ```
   sum = 0;
   for (i=1; i<= f(n); i++)
       sum +=i;
   ```

   where $f(n)$ is a function call. Give a tight big-oh bound on the running of this piece of code as a function of n, on the assumption that

   (a) The running time of $f(n)$ is $O(n)$, and the value of $f(n)$ is $n!$

   (b) The running time of $f(n)$ is $O(n)$, and the value of $f(n)$ is n

   (c) The running time of $f(n)$ is $O(n^2)$, and the value of f(n) is n

   (d) The running time of $f(n)$ is $O(1)$, and the value of f(n) is 0