ACOL 202

Extra Lecture (28th April)

Analysis of Algorithms

$$f(n) = 3 \cdot n^2$$

$$f(n) = 0.01 \cdot n^2$$

$$f(n) = \begin{cases} 202 & \text{if } 0 < n < 100 \\ n^7 & \text{if } 100 < n < 1000 \\ 2025 \cdot n^2 & \text{otherwise} \end{cases}$$

All these functions grow at the same rate.

$f : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ || non-negative domain and range

because we are going to be using this to describe the number of steps / units-of-time that an algorithm takes on an input of a particular size.

Big O (big-oh notation)

$$f(n) = O(g(n))$$

$\exists\, c > 0,\ n_0 \geq 0$ such that $\forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$

f(n) grows no faster than g(n)

Exercise Prove that $f(n) = 3n^2 + 2$ is $O(n^2)$.

We need to find a $c > 0$ and an $n_0 \geq 0$ such that

$\forall\, n \geq n_0 \qquad 3n^2 + 2 \leq c \cdot n^2$

Claim $n_0 = 1$; $c = 5$

Exercise Prove that $g(n) = 4n$ is $O(n^2)$.

Exercise Prove that $f(n) = n^3$ is not $O(n^2)$.

Proof by contradiction.

Suppose $f(n)$ is $O(n^2)$.

$\exists\, c > 0,\ n_0 \geq 0$ such that

$$\boxed{\forall n \geq n_0 \qquad n^3 \leq c \cdot n^2}$$

Let us choose $m = \max(n_0, c+1)$

Clearly, $m \geq n_0$

$m^3 > cm^2$ $\Updownarrow$ Contradiction

Homework Exercises

Prove that

i) $f(n) = O(g(n) + h(n))$ iff $f(n) = O\left(\max(g(n), h(n))\right)$

ii) If $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

iii) If $f(n) = O(h_1(n))$ and $g(n) = O(h_2(n))$ then

$$f(n) + g(n) = O(h_1(n) + h_2(n))$$

$$f(n) \cdot g(n) = O(h_1(n) \cdot h_2(n))$$

Let $p(n) = \sum_{i=0}^{k} a_i n^i$. Then $p(n) = O(n^k)$

Logarithmic functions grow more slowly than polynomials.

Let $\epsilon > 0$ be an arbitrary constant, and let $f(n) = \log n$

Then $f(n) = O(n^{\epsilon})$

Polynomials do not grow as fast as exponentials

Let $b > 1$ and $k \geq 0$ be constants, and let $p(n) = \sum_{i=0}^{k} a_i n^i$ be a polynomial. Then $p(n) = O(b^n)$

The base of a logarithm does not matter

(– $\log_a n$ and $\log_b n$ grow at the same rate)

but that of an exponentiation does.

(a^n and b^n do not grow at the same rate).

Let $b > 1$ and $k > 0$ be arbitrary constants.

Then $f(n) = \log_b n^k$ is $O(\log n)$.

Made with Goodnotes

Let $b \geq 1$ and $c \geq 1$ be arbitrary constants. Then

$f(n) = b^n$ is $O(c^n)$ iff $b \leq c$.

Big-Omega

A function f grows no slower than g

$f(n) = \Omega(g(n))$ if $\exists\, d > 0, n_0 \geq 0$ such that $\forall\, n \geq n_0$ $f(n) \geq d \cdot g(n)$.

Big-Theta

f grows at the same rate as g

written as $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$$f(n) = 3n^2 + 1$$

$$f(n) = \Omega(n)$$

$$f(n) = \Theta(n)$$

The running time of an algorithm A on an input x is the number of primitive steps that A takes when run on x.

Consider binary search

0 1 2 3 4 5 6 7 8 9 10

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31], 4

binsearch (list, elem)

$$k = \left\lfloor \frac{\text{size of list}}{2} \right\rfloor$$

if list[K] == elem return K

else if list[K] > elem

binsearch(list [0... K-1], elem)

else binsearch (list [K+1 .... n], elem)

return -1

Worst-case running time

$$\text{Steps}(n) = \text{Steps}(n/2) + 1$$

$$T(n) = T(n/2) + 1$$ // recurrence relation

Next lecture — Selection Sort / Merge Sort