Running Time Analysis and Asymptotic Notation

COMP4500/7500

July 24, 2019

What is an algorithm?

- An *algorithm* is a well-defined computation procedure that takes some values as *input* and produces some value, or set of values, as *output* [CLRS CH1].
- Usually defined to solve a specific *computational problem*.
- We would like such algorithms to be *correct* (w.r.t. their problem) and *efficient*.

Sorting Problem

input A sequence of numbers $\langle a_1, a_2, \ldots, a_n \rangle$

output A permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Sorting Algorithm: Insertion Sort

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Try it on $A = [5, 2, 4, 6, 1, 3]$.

Sorting Algorithm: Insertion Sort

INSERTION-SORT(A)

```
for j = 2 to A.length
        // Invariant: A[1..j−1] contains original elements
        // from A[1..j−1], but in sorted order
        key = A[j]
        // Insert A[j] into the sorted sequence A[1..j − 1]
        i = j − 1
        while i > 0 and A[i] > key
                A[i + 1] = A[i]
                i = i − 1
        A[i + 1] = key
```

Loop invariants can be used to prove algorithms are correct.

Execution Time

What does it depend on?

- *input size*, e.g. sorting 10 elements versus 1000 elements
- *input value*, e.g. sorting an already sorted list versus a reverse sorted list

Generally we want an upper bound on execution time.

Execution time: Worst, Average and Best case

Definition (Worst case)

$$T(n) = \text{Maximum execution time over all inputs of size } n$$

Definition (Average case)

$$T(n) = \begin{array}{l}\text{Average execution time over all inputs of size } n, \\ \text{weighted by the probability of the input}\end{array}$$

Definition (Best case)

$$T(n) = \text{Mimimum execution time over all inputs of size } n$$

Running Time Analysis

The *running time* of an algorithm on a given input can be measured in terms of the number of primitive *steps* executed.

Sorting Algorithm: Insertion Sort

Let n be $A.length$.
Let's measure time in terms of the number of array comparisons (in red):

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Worst case: $T(n) = \sum_{j=2}^{n}(j-1) = n(n-1)/2$
Best case: $T(n) = \sum_{j=2}^{n}(1) = n-1$

Running Time Analysis: Recursion

- More complicated than conditionals and loops.
- Running time can often be described by a **recurrence**
- Overall running time on a problem of size n is described in terms of running time(s) on smaller inputs and functions of n. (see CLRS Ch2 Ex: Merge Sort)

Asymptotic Analysis: the general idea

- Groups functions together based on their *rate of growth*.
  - Merge sort $\Theta(n \lg n)$
  - Insertion sort $\Theta(n^2)$
- For large inputs, difference in *order* outweigh constant factors:
  - E.g. Merge sort is ultimately better for large enough n no matter what the constant factors.
- Ignores implementation dependent constants:
  - machine speed
  - compiler

Growth of functions

Largest instance that can be solved in given time

$T(n)$	$1 second$	$1 day$	$1 year$
n	$1,000,000$	$86,400,000,000$	$31,536,000,000,000$
$n \log n$	$62,746$	$2,755,147,514$	$798,160,978,500$
n^2	$1,000$	$293,938$	$5,615,692$
n^3	100	$4,421$	$31,593$
2^n	19	36	44

Asymptotic Analysis: limitations?

- Constant factors are relevant for
  - small input sizes, and
  - algorithms of same order

Asymptotic notation

For functions f and g

$f \in O(g)$ f is asymptotically bounded above by g to within a constant factor

- $n \in O(n^2)$
- $64,000n \in O(n)$

$f \in \Omega(g)$ f is asymptotically bounded below by g to within a constant factor

- $g \in O(f)$
- $n^2 \in \Omega(n)$

$f \in \Theta(g)$ f is asymptotically bounded above and below by g to within a constant factor

- $f \in O(g) \land f \in \Omega(g)$
- $42n \in \Theta(n)$
- $n \notin \Theta(n^2)$

Upper Bounds

Definition (Big-O)

$$O(g) = \{f | \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq f(n) \leq c.g(n)\}$$

$$f \in O(g) \iff \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq f(n) \leq c.g(n)$$

Upper Bounds

Definition (Big-O)

$$O(g) = \{f | \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq f(n) \leq c.g(n)\}$$

Prove that: $2n^3 \in O(n^3)$

Need constants $c, n_0 > 0$ such that $(\forall n \geq n_0 \bullet 0 \leq 2n^3 \leq cn^3)$.

Choosing $c = 2$ and $n_0 = 1$ will suffice since:

$$\begin{aligned} & \forall n \geq 1 \bullet 0 \leq 2n^3 \leq 2n^3 \\ \equiv \quad & \text{true} \end{aligned}$$

Quick question: upper Bounds

Definition (Big-O)

$$O(g) = \{f | \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq f(n) \leq c.g(n)\}$$

Prove that: $2n^2 \in O(n^3 - n^2)$

Lower Bounds

Definition (Ω)

$$\Omega(g) = \{f | \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq c.g(n) \leq f(n)\}$$

$$f \in \Omega(g) \iff \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq c.g(n) \leq f(n)$$

Lower Bounds

Definition (Ω)

$$\Omega(g) = \{f | \exists c, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq c.g(n) \leq f(n)\}$$

Prove that: $2n^3 \in \Omega(n^3)$

Need constants $c, n_0 > 0$ such that $(\forall n \geq n_0 \bullet 0 \leq cn^3 \leq 2n^3)$.

Choosing $c = 2$ and $n_0 = 1$ will suffice since:

$$\begin{aligned} & \forall n \geq 1 \bullet 0 \leq 2n^3 \leq 2n^3 \\ \equiv \quad & \text{true} \end{aligned}$$

Tight Bounds

Definition (Θ)

$$\Theta(g) = \{f |\ \exists c_1, c_2, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)\}$$

$$f \in \Theta(g) \iff \exists c_1, c_2, n_0 > 0 \bullet \forall n \geq n_0 \bullet 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$$

Tight Bounds

Definition (Θ)

$$\Theta(g) = \{f \mid \exists c_1, c_2, n_0 > 0 \bullet \\ \forall n \geq n_0 \bullet 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)\}$$

Prove that: $2n^3 \in \Theta(n^3)$

Need constants $c_1, c_2, n_0 > 0$ such that
$(\forall n \geq n_0 \bullet 0 \leq c_1 n^3 \leq 2n^3 \leq c_2 n^3)$.

Choosing $c_1 = c_2 = 2$ and $n_0 = 1$ will suffice since:

$$\begin{aligned} & \forall n \geq 1 \bullet 0 \leq 2n^3 \leq 2n^3 \leq 2n^3 \\ \equiv \quad & \text{true} \end{aligned}$$

Tight Bounds: Prove: $\frac{n^2}{2} - 2n \in \Theta(n^2)$

Need constants $c_1, c_2, n_0 > 0$ such that

$$(\forall n \geq n_0 \bullet 0 \ \leq \ c_1 n^2 \ \leq \ \frac{n^2}{2} - 2n \ \leq \ c_2 n^2) \ .$$

$$\begin{array}{lll} & 0 \leq c_1 n^2 & \\ \equiv & \text{true} & \text{if } n \geq 0 \end{array}$$

$$\begin{array}{lll} & c_1 n^2 \leq \frac{n^2}{2} - 2n & \\ \equiv & 2 \leq n(\frac{1}{2} - c_1) & \text{if } n > 0 \\ \equiv & 8 \leq n & \text{if } c_1 = \frac{1}{4} \end{array}$$

$$\begin{array}{lll} & \frac{n^2}{2} - 2n \leq c_2 n^2 & \\ \equiv & \text{true} & \text{if } n \geq 0 \text{ and } c_2 \geq \frac{1}{2} \end{array}$$

Choosing $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$ and $n_0 = 8$ will therefore suffice.

Properties of asymptotic notation

Theorem

$$f \in O(g) \implies g + f \in \Theta(g)$$

For example, $n \in O(n^2) \implies n^2 + n \in \Theta(n^2)$

Theorem

For $k > 0$,

$$k.n^a \in \Theta(n^a)$$

Theorem

For $k > 0$ and $0 \leq a \leq b$

$$k.n^a \in O(n^b)$$

Properties of asymptotic notation

Theorem

For any functions f and g

$$f \in O(g) \iff g \in \Omega(f)$$

Theorem

For any functions f and g

$$f \in \Theta(g) \iff f \in O(g) \land f \in \Omega(g)$$

Comparing Functions

Definition (Asymptotically non-negative)

A function f is *asymptotically non-negative* if

$$\exists n_0 \bullet (\forall n \geq n_0 \bullet f(n) \geq 0)$$

Theorem

If f and g are asymptotically non-negative functions, and

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \begin{array}{lll} = c \geq 0 & \textit{then} & f \in O(g) \\ = c > 0 & \textit{then} & f \in \Theta(g) \\ = \infty & \textit{then} & g \in O(f) \end{array}$$

where c is a real-valued constant.

Cannot compare all functions

For example the functions

$$n \text{ and } n^{1+\sin n}$$

are not comparable.

Polynomials

$$p(n) = \sum_{k=0}^{d} a_k n^k = a_0 n^0 + a_1 n^1 + a_2 n^2 + \cdots + a_d n^d$$

Theorem

If $a_d > 0$,

$$p(n) \in \Theta(n^d)$$

$$\begin{aligned}
&\lim_{n\to\infty} \tfrac{p(n)}{n^d} \\
&= \lim_{n\to\infty} \textstyle\sum_{k=0}^{d} \tfrac{a_k n^k}{n^d} \\
&= \lim_{n\to\infty} \tfrac{a_d n^d}{n^d} + \textstyle\sum_{k=0}^{d-1} \tfrac{a_k n^k}{n^d} \\
&= \lim_{n\to\infty} a_d + \textstyle\sum_{k=0}^{d-1} \tfrac{a_k}{n^{d-k}} \\
&= a_d
\end{aligned}$$

Polynomials

Theorem

For $a > 1$ and $d \geq 0$,

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

$$\begin{aligned}
& \lim_{n\to\infty} \frac{n^d}{a^n} \\
= & \quad \text{by L'Hôpital's rule} \\
& \lim_{n\to\infty} \frac{dn^{d-1}}{a^n \ln a} \\
= & \quad \text{by L'Hôpital's rule} \\
& \lim_{n\to\infty} \frac{d(d-1)n^{d-2}}{a^n (\ln a)^2} \\
= & \quad d \text{ times} \\
& \lim_{n\to\infty} \frac{d(d-1)(d-2)...1 n^0}{a^n (\ln a)^d} \\
= & \lim_{n\to\infty} \frac{d!}{a^n (\ln a)^d} \\
= & \quad \text{as } d! \text{ and } (\ln a)^d \text{ are constants and } a^n \text{ tends to } \infty \\
& 0
\end{aligned}$$