EC330: Applied Algorithms for Engineers

Asymptotic Notation

What is an algorithm?

- An unambiguous list of steps (program) to transform some input into some output

- Pick a Problem (set)
- Find method to solve
  1. Correct for all cases (elements of set)
  2. Each step is finite ($\Delta\ t_{step}$ < max time)
  3. Next step is unambiguous
  4. Terminate in finite number of steps

What is an Algorithm?

```
Selection Sort(A[1…n])
for (i=1 to n)
      for j=i+1 to n
            if A[i]>A[j] then
                  swap A[i] and A[j]
```

``loop invariant”

Lemma: After iteration *j* of the inner loop A[i] is the smallest element of A[i…j].

Theorem: Selection Sort puts the elements of A[] into increasing order.

Proof by induction.

How fast is Selection Sort?

```
Selection Sort(A[1…n])
for (i=1 to n)
      for j=i+1 to n
            if A[i]>A[j] then
                  swap A[i] and A[j]
```

2 $\quad \sum_{j=i+1}^{n} 2$ $\quad \sum_{i=1}^{n}\sum_{j=i+1}^{n} 2$

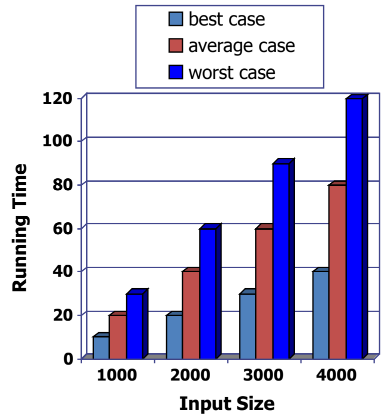
It depends!

- What computer? What language? What compiler? What operating system? What architecture?
- How long does it take to compare A[i] and A[j]? …to swap A[i] and A[j]? To find A[i]?
- If swap and comparison take 1 unit of time, how much time do they take?

  $$\sum_{i=1}^{n}\sum_{j=i+1}^{n} 2$$
- How does this compare to n^3, n^2, $n^3/100$?

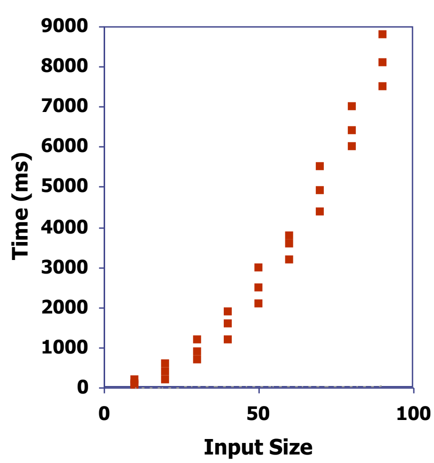
Running Time

- Most algorithms transform input objects into output objects
- The running time of an algorithm typically grows with the input size
- Average case time is often difficult to determine
- We often focus on the worst case running time
  - Easier to analyze
  - Crucial to applications such as games, finance and robotics



Experimental Studies

- Write a program to implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a method like clock() to get an accurate measure of the actual running time
- Plot the results



Limitations of Experiments

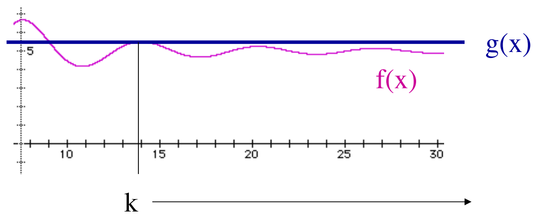
- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment
- In order to compare two algorithms, the same hardware and software environments must be used

Theoretical Analysis

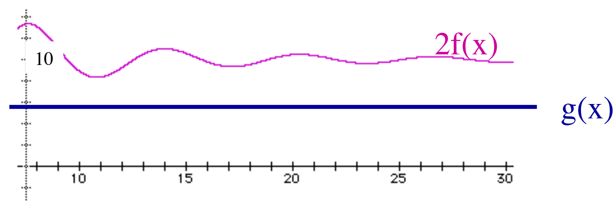
- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, *n*
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/ software environment

The Land of O()

- Which function is "bigger"?
  - Only care about eventual size (asymptotic)



  - Don't care about multiplicative constants



The Land of O()

Big-O: for functions that map real numbers (or a subset) to real numbers (or a subset)

Intuitively: f(n) is O(g(n)) if *f* is eventually smaller or equal to *g* or some multiple of g

"Real" Definition: f(n) is O(g(n)) iff
$\exists c>0 \ \exists n_0>0 \quad |f(n)| \leq c|g(n)|$ whenever $n > n_0$.

"Math-freak" Definition: f(n) is O(g(n)) if
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ is finite.

O() Examples

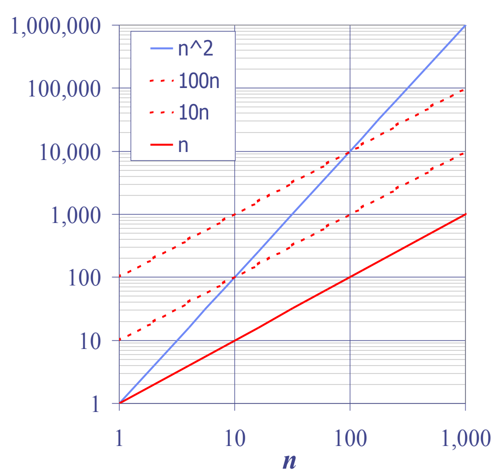
"Real" Definition: f(n) is O(g(n)) iff
$\exists c>0\ \exists n_0>0\ \ |f(n)| \le c|g(n)|$ whenever $n > n_0$.

	c	n_0	
n is O(n)	1	1	
n is O(5n)	1	1	
n is O(n/2-17)	100	1	
n is O($n/10^{100}$)	10^{100}	1	
5n+3 is O(n^2)	25	6	[proof by induction]

n^2 is not O(5n+3)

Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \le cn$
  - $n \le c$
  - The above inequality cannot be satisfied since c must be a constant

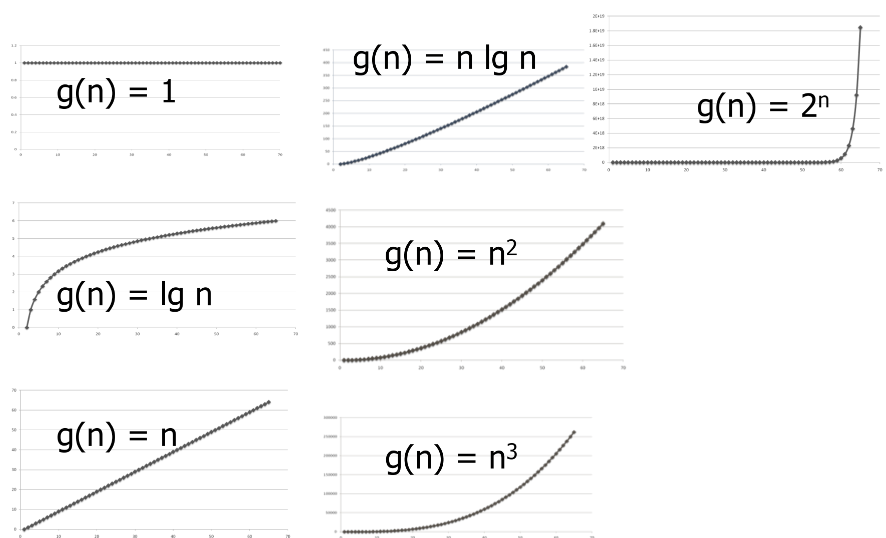


Why is big-O Important?

(machine does 1,000,000 steps per second)

time \ input size	**10**	**20**	**30**	**40**	**50**	**60**
log n	3.3μsec	4.4μsec	5μsec	5.3μsec	5.6μsec	5.9μsec
n	10μsec	20μsec	30μsec	40μsec	50μsec	60μsec
n^2	100μsec	400μsec	900μsec	1.5msec	2.5msec	3.6msec
n^5	0.1sec	3.2sec	24.3sec	1.7min	5.2min	13min
3^n	59msec	48min	6.5yrs	385,500yrs	2×10^8 centuries...	
n!	3sec	7.8×10^8 millenia				

Functions Graphed



Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

Properties of big-O

If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$ **then**

- $f_1(n) + f_2(n)$ is $O(g_1(n)+g_2(n))$
- $f_1(n) + f_2(n)$ is $O(\max\{g_1(n),g_2(n)\})$
- $f_1(n) * f_2(n)$ is $O(g_1(n)*g_2(n))$

[prove these to yourself!]

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ **then**
$f(n)$ is $O(h(n))$

- Examples:

Cousins of O()

Like

f(n) is O(g(n)) ["big oh"] iff
$\exists C>0\ \exists n_0\ \ |f(n)| \leq C|g(n)|$ whenever $n > n_0$.

$\leq$

f(n) is Ω(g(n)) ["big Omega"] iff
$\exists C>0\ \exists n_0\ \ |f(n)| \geq C|g(n)|$ whenever $n > n_0$.
$\Leftrightarrow$ g(n) is O(f(n))

$\geq$

f(n) is Θ(g(n)) ["big Theta"] iff
f(n) is O(g(n)) <u>and</u> g(n) is O(f(n))

$=$

f(n) is o(g(n)) ["little oh"] iff
$\forall C>0\ \exists n_0\ \ |f(n)| < C|g(n)|$ whenever $n > n_0$.
$\Leftarrow \lim_{n\to\infty} f(n)/g(n) = 0$
intuitively: f(n) is O(g(n)) <u>but</u> f(n) is **not** Θ(g(n))

$<$

f(n) is ω(g(n)) ["little omega" - not doubleyou] iff
g(n) is o(f(n))
$\Leftarrow \lim_{n\to\infty} f(n)/g(n) = \infty$

$>$

Rules of Thumb

- For polynomials, only the largest term matters.
  $a_k x^k + a_{k-1} x^{k-1} + \ldots + a_0$ is $\Theta(x^k)$
- log n is o(n)
  Proof: $\lim_{n\to\infty} \log(n)/n = 0$
- Some common functions in "non-decreasing" order:
  $1 \quad \log(n) \quad \sqrt{n} \quad n \quad n\log(n) \quad n^2 \quad n^3 \quad n^{100} \quad 2^n \quad 3^n \quad n!$
- Selection sort requires $\Theta(n^2)$ time.

"Math-freak" Definition: f(n) is O(g(n)) if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \text{ is finite}$$

f(n) is O(g(n)) ["big oh"] iff
$\exists C>0\ \exists n_0\ \ |f(n)| \le C|g(n)|$ whenever $n > n_0$.

f(n) is Ω(g(n)) ["big Omega"] iff
$\exists C>0\ \exists n_0\ \ |f(n)| \ge C|g(n)|$ whenever $n > n_0$.
$\Leftrightarrow$ g(n) is O(f(n))

f(n) is Θ(g(n)) ["big Theta"] iff
f(n) is O(g(n)) <u>and</u> g(n) is O(f(n))

f(n) is o(g(n)) ["little oh"] iff
$\forall C>0\ \exists n_0\ \ |f(n)| < C|g(n)|$ whenever $n > n_0$.
$\Leftarrow \lim_{n\to\infty} f(n)/g(n) = 0$
intuitively: f(n) is O(g(n)) <u>but</u> f(n) is **not** Θ(g(n))

f(n) is ω(g(n)) ["little omega" - not doubleyou] iff
g(n) is o(f(n))
$\Leftarrow \lim_{n\to\infty} f(n)/g(n) = \infty$

<u>Like</u>

$\le$

$\ge$

$=$

$<$

$>$