



Asymptotic Analysis

The Gist

Design and Analysis
of Algorithms I

Motivation

Importance: Vocabulary for the design and analysis of algorithms (e.g. “big-Oh” notation).

- “Sweet spot” for high-level reasoning about algorithms.
- Coarse enough to suppress architecture/language/compiler-dependent details.
- Sharp enough to make useful comparisons between different algorithms, especially on large inputs (e.g. sorting or integer multiplication).

Asymptotic Analysis

High-level idea: Suppress constant factors and lower-order terms

too system-dependent

irrelevant for large inputs

Example: Equate $6n \log_2 n + 6$ with just $n \log n$.

Terminology: Running time is $O(n \log n)$

[“big-Oh” of $n \log n$]

where n = input size (e.g. length of input array).

Example: One Loop

Problem: Does array A contain the integer t ? **Given** A (array of length n) and t (an integer).

Algorithm 1

```
1: for  $i = 1$  to  $n$  do  
2:   if  $A[i] == t$  then  
3:     Return TRUE  
4: Return FALSE
```

Question: What is the running time?

- A) $O(1)$ C) $O(n)$
B) $O(\log n)$ D) $O(n^2)$

Example: Two Loops

Given A, B (arrays of length n) and t (an integer). [Does A or B contain t ?]

Algorithm 2

```
1: for  $i = 1$  to  $n$  do  
2:   if  $A[i] == t$  then  
3:     Return TRUE  
4: for  $i = 1$  to  $n$  do  
5:   if  $B[i] == t$  then  
6:     Return TRUE  
7: Return FALSE
```

Question: What is the running time?

- A) $O(1)$ C) $O(n)$
B) $O(\log n)$ D) $O(n^2)$

Example: Two Nested Loops

Problem: Do arrays A, B have a number in common? **Given** arrays A, B of length n .

Algorithm 3

```
1: for  $i = 1$  to  $n$  do  
2:   for  $j = 1$  to  $n$  do  
3:     if  $A[i] == B[j]$  then  
4:       Return TRUE  
5: Return FALSE
```

Question: What is the running time?

A) $O(1)$ C) $O(n)$

B) $O(\log n)$ D) $O(n^2)$

Example: Two Nested Loops (II)

Problem: Does array A have duplicate entries? **Given** arrays A of length n .

Algorithm 4

```
1: for  $i = 1$  to  $n$  do
2:   for  $j = i+1$  to  $n$  do
3:     if  $A[i] == A[j]$  then
4:       Return TRUE
5: Return FALSE
```

Question: What is the running time?

A) $O(1)$ C) $O(n)$

B) $O(\log n)$ D) $O(n^2)$



Design and Analysis
of Algorithms I

Asymptotic Analysis

Big-Oh: Definition

Big-Oh: English Definition

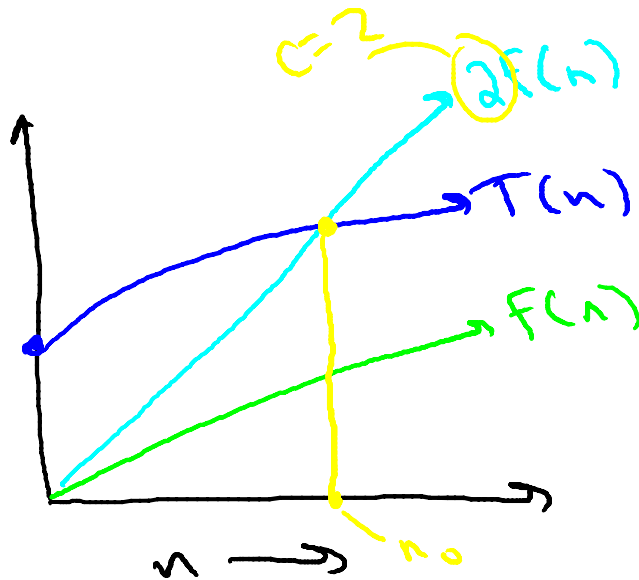
Let $T(n)$ = function on $n = 1, 2, 3, \dots$

[usually, the worst-case running time of an algorithm]

Q : When is $T(n) = O(f(n))$?

A : if eventually (for all sufficiently large n), $T(n)$ is bounded above by a constant multiple of $f(n)$

Big-Oh: Formal Definition



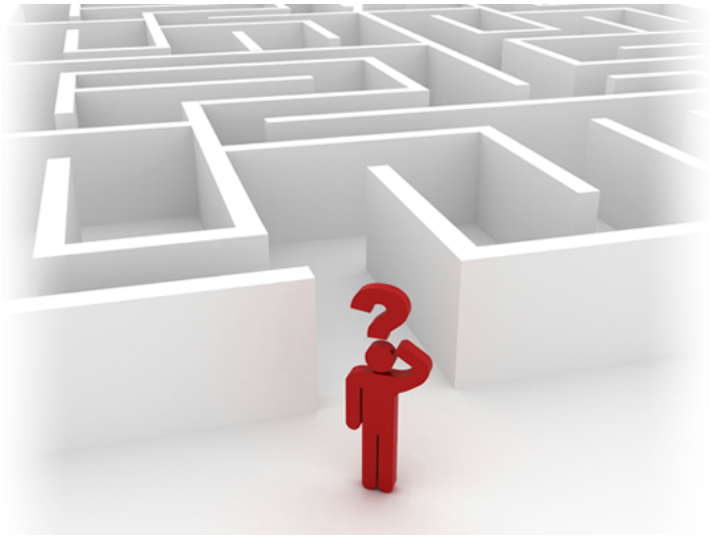
Picture $T(n) = O(f(n))$

Formal Definition : $T(n) = O(f(n))$ if
and only if there exist constants
 $c, n_0 > 0$ such that

$$T(n) \leq c \cdot f(n)$$

For all $n \geq n_0$

Warning : c, n_0 cannot depend on n



Design and Analysis
of Algorithms I

Asymptotic Analysis

Big-Oh: Basic Examples

Example #1

Claim : if $T(n) = a_k n^k + \dots + a_1 n + a_0$ then

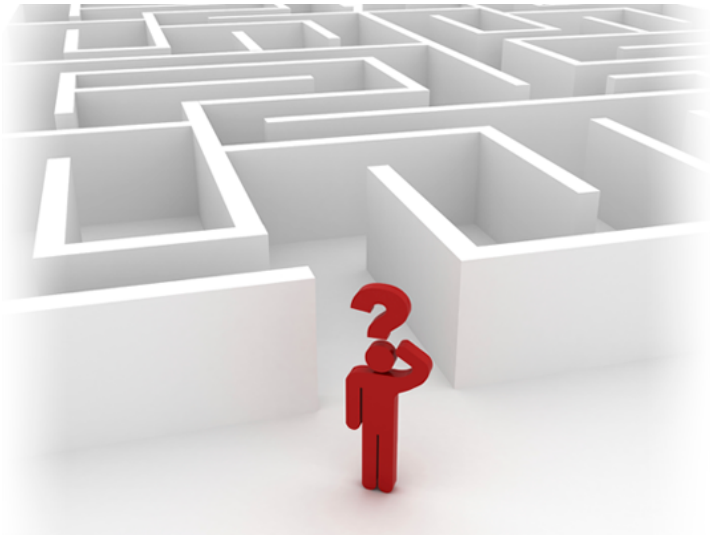
$$T(n) = O(n^k)$$

Proof : Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|$

Need to show that $\forall n \geq 1, T(n) \leq c \cdot n^k$

We have, for every $n \geq 1$,

$$\begin{aligned} T(n) &\leq |a_k|n^k + \dots + |a_1|n + |a_0| \\ &\leq |a_k|n^k + \dots + |a_1|n^k + |a_0|n^k \\ &= c \cdot n^k \end{aligned}$$



Design and Analysis
of Algorithms I

Asymptotic Analysis

Big-Oh: Relatives (Omega & Theta)

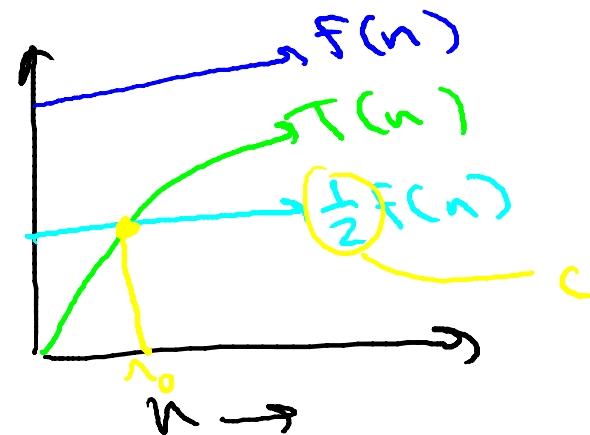
Omega Notation

Definition : $T(n) = \Omega(f(n))$

If and only if there exist constants c, n_0 such that

$$T(n) \geq c \cdot f(n) \quad \forall n \geq n_0.$$

Picture



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

Theta Notation

Definition : $T(n) = \theta(f(n))$ if and only if
 $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$


Equivalent : there exist constants c_1, c_2, n_0 such that


$$c_1 f(n) \leq T(n) \leq c_2 f(n)$$


$$\forall n \geq n_0$$

Let $T(n) = \frac{1}{2}n^2 + 3n$. Which of the following statements are true ? (Check all that apply.)

☐ $T(n) = O(n)$.

 ☐ $T(n) = \Omega(n)$. $[n_0 = 1, c = 4]$

 ☐ $T(n) = \Theta(n^2)$. $[n_0 = 1, c_1 = 1/2, c_2 = 4]$

 ☐ $T(n) = O(n^3)$. $[n_0 = 1, c = 1/2]$

Little-Oh Notation

Definition : $T(n) = o(f(n))$ if and only if for all constants $c > 0$, there exists a constant n_0 such that

$$T(n) \leq c \cdot f(n) \quad \forall n \geq n_0$$

Exercise : $\forall k \geq 1, n^{k-1} = o(n^k)$

Where Does Notation Come From?

“On the basis of the issues discussed here, I propose that members of SIGACT, and editors of computer science and mathematics journals, adopt the O , Ω , and Θ notations as defined above, unless a better alternative can be found reasonably soon”.

-D. E. Knuth, “Big Omicron and Big Omega and Big Theta”, SIGACT News, 1976. Reprinted in “Selected Papers on Analysis of Algorithms.”

Example #2

Claim : for every $k \geq 1$, n^k is not $O(n^{k-1})$

Proof : by contradiction. Suppose $n^k = O(n^{k-1})$

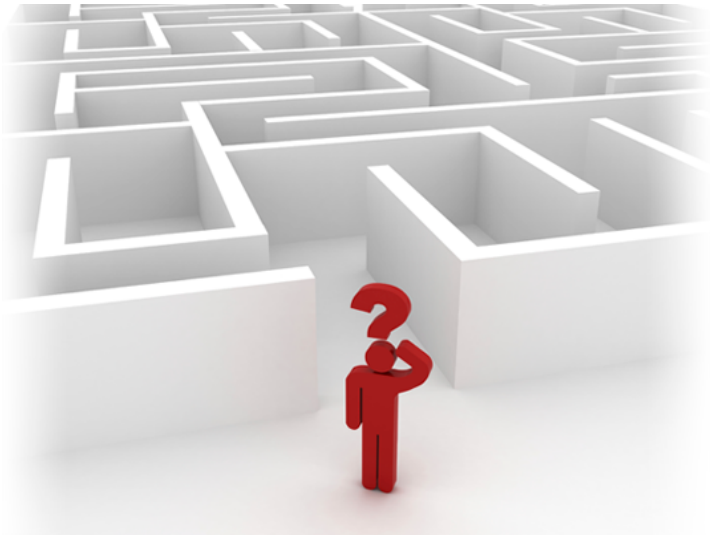
Then there exist constants c, n_0 such that

$$n^k \leq c \cdot n^{k-1} \quad \forall n \geq n_0$$

But then [cancelling n^{k-1} from both sides]:

$$n \leq c \quad \forall n \geq n_0$$

Which is clearly False [contradiction].



Design and Analysis
of Algorithms I

Asymptotic Analysis

Additional Examples

Example #1

Claim: $2^{n+10} = O(2^n)$

Proof: need to pick constants c, n_0 such that

$$(*) \quad 2^{n+10} \leq c \cdot 2^n \quad n \geq n_0$$

Note: $2^{n+10} = 2^{10} \times 2^n = (1024) \times 2^n$

So if we choose $c = 1024, n_0 = 1$ then $(*)$ holds.

Q.E.D

Example #2

Claim : $2^{10n} \neq O(2^n)$

Proof : by contradiction. If $2^{10n} = O(2^n)$ then there exist constants $c, n_0 > 0$ such that

$$2^{10n} \leq c \cdot 2^n \quad n \geq n_0$$

But then [cancelling 2^n]

$$2^{9n} \leq c \quad \forall n \geq n_0$$

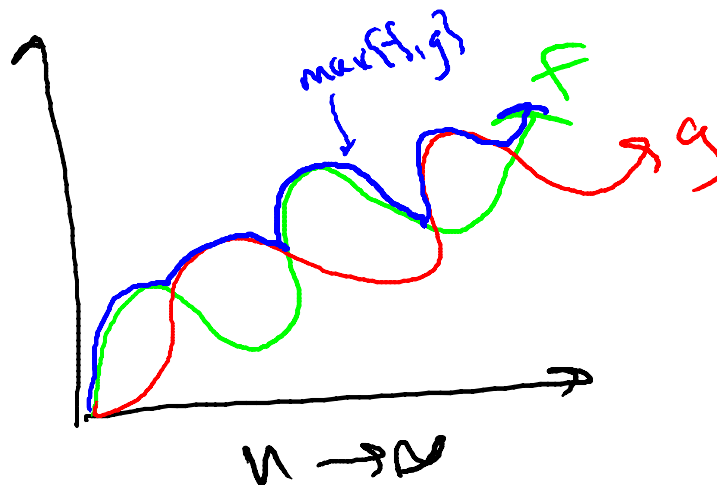
Which is certainly false.

Q.E.D

Example #3

Claim : for every pair of (positive) functions $f(n)$, $g(n)$,

$$\max\{f, g\} = \theta(f(n) + g(n))$$



Example #3 (continued)

Proof : $\max\{f, g\} = \theta(f(n) + g(n))$

For every n , we have

$$\max\{f(n), g(n)\} \leq f(n) + g(n)$$

And

$$2 * \max\{f(n), g(n)\} \geq f(n) + g(n)$$

Thus $\frac{1}{2} * (f(n) + g(n)) \leq \max\{f(n), g(n)\} \leq f(n) + g(n) \quad \forall n \geq 1$
 $\Rightarrow \max\{f, g\} = \theta(f(n) + g(n))$ [where $n_0 = 1, c_1 = 1/2, c_2 = 1$]

Q.E.D