Algorithms

(Algorithm Analysis)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

February 15, 2021

Contents

- Complexity Analysis
- Worst-case, Best-case, and Average-case
- Asymptotic Notations and Complexity Classes

Complexity Analysis

Complexity analysis

- Framework for analyzing the efficiency/complexity of algorithms
  e.g.: time complexity and space complexity
- Running time of a computer program depends on:
  - Algorithm
  - Input size
  - Input data distribution
  - Machine or computing system
  - Operating system
  - Compiler
  - Programming language
  - Coding
- We analyze the running time of algorithms using
  asymptotic analysis

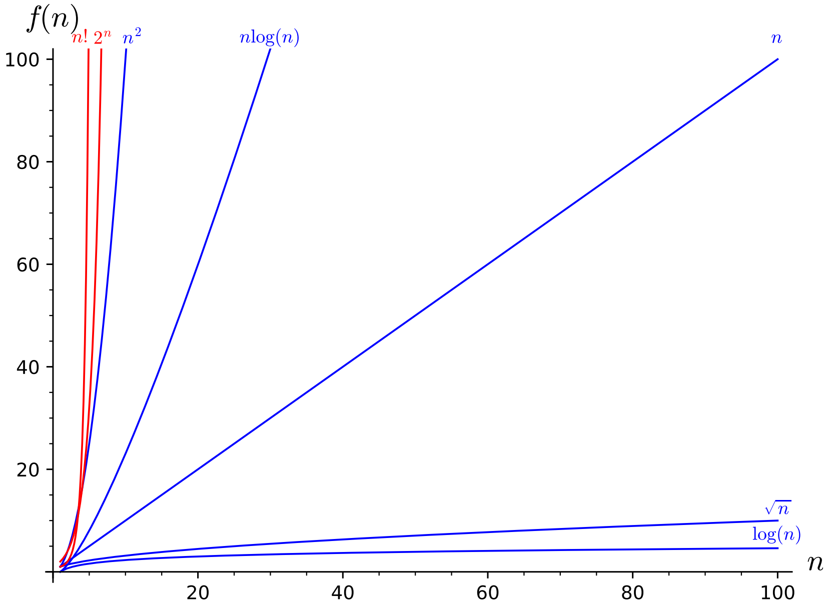
Complexity analysis

- Running time of an algorithm can be considered as a function of the algorithm's input size

Complexity analysis

Problem	Running time
Search in a sorted array	$\mathcal{O}(\log n)$
Search in an unsorted array, Integer addition	$\mathcal{O}(n)$
Generate primes	$\mathcal{O}(n \log \log n)$
Sorting, Fast Fourier transform	$\mathcal{O}(n \log n)$
Integer multiplication	$\mathcal{O}(n^2)$
Matrix multiplication	$\mathcal{O}(n^3)$
Linear programming	$\mathcal{O}(n^{3.5})$
Primality test	$\mathcal{O}(n^{10})$
Satisfiability problem	$\mathcal{O}(2^n)$
Traveling salesperson problem	$\mathcal{O}((n-1)!)$
Sudoku, Chess, Checkers, Go	expo. class
Simulate problem, Halting problem	∞
Program correctness, Program equivalence	∞
Integral roots of a polynomial	∞

Polynomial and exponential functions



Units for measuring running time

- Basic operation is the most important operation of the algorithm. Each basic operation takes constant time.
  - Arithmetic operation ($\times, \div, +, -$)
  - Comparison operation ($<, \leq, =, \neq, >, \geq$)
  - Memory operation ($a \leftarrow b, C[i]$)
  - Function invocation and return

Units for measuring running time

Array-Sum$(A[0..n-1])$

$sum \leftarrow 0$ $\triangleright$ 1 op
for $i \leftarrow 0$ **to** $n-1$ **do** $\triangleright$ $n\times$ (cmp + inc) = 2n ops
  $sum \leftarrow sum + A[i]$ $\triangleright$ $n\times$ (index + add + assign) = 3n ops
return sum $\triangleright$ 1 op

- Runtime $= 1 + n(2+3) + 1 = 5n + 2$ operations
  (n comparisons, n increments, n memory index accesses,
  $n+1$ assignments, n additions, 1 function return)

Worst-case, best-case, and average-case analysis

- Worst-case complexity $T_{\text{worst}}(n)$ of an algorithm.
  Complexity for the worst-case input of size n for which the algorithm runs the longest among all possible inputs of that size.
- Best-case complexity $T_{\text{best}}(n)$ of an algorithm.
  Complexity for the best-case input of size n for which the algorithm runs the shortest among all possible inputs of that size.
- Average-case complexity $T_{\text{avg}}(n)$ of an algorithm.
  Complexity for a typical or random input of size n.
- Amortized complexity $T_{\text{amortized}}(n)$ of an algorithm.
  Average complexity for a sequence of operations.

Worst-case, best-case, and average-case analysis

Problem

What are the worst-case, best-case, and average-case analyses for the sequential search algorithm?

$\textsc{Sequential-Search}(A[0..n-1], key)$

Input: An array A and search key key
Output: The index of the first element in A that matches key or -1 if there are no matching elements

1. $i \leftarrow 0$
2. **while** $i < n$ **and** $A[i] \neq key$ **do**
3. $\quad i \leftarrow i + 1$
4. **if** $i < n$ **then return** i
5. **else return** -1

Worst-case, best-case, and average-case analysis

Solution

- $T_{\text{worst}}(n) = n$ $\triangleright$ Why?
- $T_{\text{best}}(n) = 1$ $\triangleright$ Why?
- $T_{\text{avg}}(n) = \left.\begin{cases} \frac{n+1}{2} & \text{if search is successful,} \\ n & \text{if search is unsuccessful.} \end{cases}\right\}$ $\triangleright$ Why?

Let $p \in [0, 1]$ be the probability of successful search
The prob. of first match occurring at any position be the same
$T_{\text{avg}}(n) = \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}\right) + n \cdot (1 - p)$
Simplifying, $T_{\text{avg}}(n) = \frac{p(n+1)}{2} + n(1 - p)$.

What do you get when you set $p = 1$ or $p = 0$?

Asymptotic Notations and Complexity Classes

Asymptotic notations

Notation	Meaning
$\mathcal{O}\left(g(n)\right)$ at most $g(n)$	Set of all functions with the same or lower order of growth as $g(n)$ $3n^2 \in \mathcal{O}\left(n^2\right)$, $n^2/17+n \in \mathcal{O}\left(n^2\right)$, $n(n-1)/2 \in \mathcal{O}\left(n^2\right)$ $n \in \mathcal{O}\left(n^2\right)$, $4\sqrt{n}+3\log^2 n \in \mathcal{O}\left(n^2\right)$, $2000 \in \mathcal{O}\left(n^2\right)$ $n^3 \notin \mathcal{O}\left(n^2\right)$, $0.001n^{\pi-1} \notin \mathcal{O}\left(n^2\right)$, $n^4+n+1 \notin \mathcal{O}\left(n^2\right)$
$\Omega\left(g(n)\right)$ at least $g(n)$	Set of all functions with the same or higher order of growth as $g(n)$ $3n^2 \in \Omega\left(n^2\right)$, $n^2/17+n \in \Omega\left(n^2\right)$, $n(n-1)/2 \in \Omega\left(n^2\right)$ $n^3 \in \Omega\left(n^2\right)$, $0.001n^{\pi-1} \in \Omega\left(n^2\right)$, $n^4+n+1 \in \Omega\left(n^2\right)$ $n \notin \Omega\left(n^2\right)$, $4\sqrt{n}+3\log^2 n \notin \Omega\left(n^2\right)$, $2000 \notin \Omega\left(n^2\right)$
$\Theta\left(g(n)\right)$ same as $g(n)$	Set of all functions with the same order of growth as $g(n)$ $3n^2 \in \Theta\left(n^2\right)$, $n^2/17+n \in \Theta\left(n^2\right)$, $n(n-1)/2 \in \Theta\left(n^2\right)$

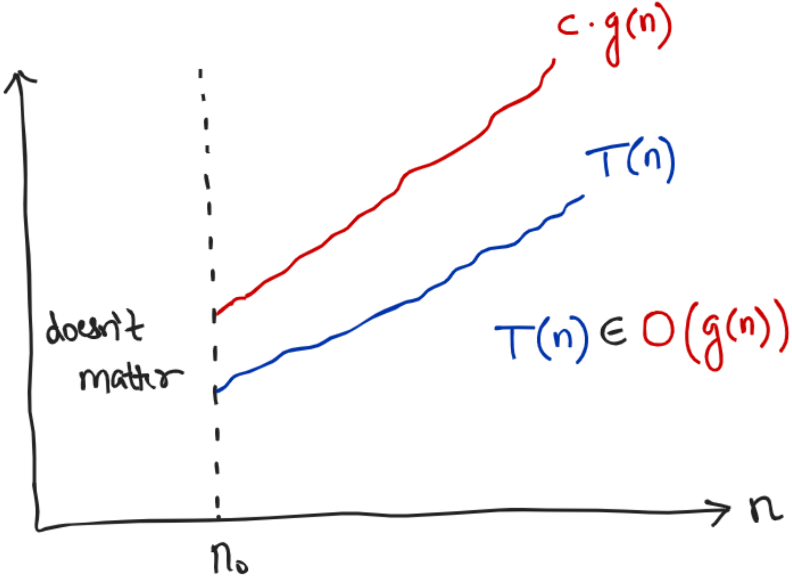
$\mathcal{O}\left(\right)$ notation

Definition

A function $T(n)$ is said to be in $\mathcal{O}(g(n))$, denoted $T(n) \in \mathcal{O}(g(n))$, if $T(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and nonnegative integer n_0 such that

$$\boxed{T(n) \leq c \cdot g(n) \qquad \text{for all } n \geq n_0}$$

$\mathcal{O}()$ notation



$\mathcal{O}\left(\right)$ notation

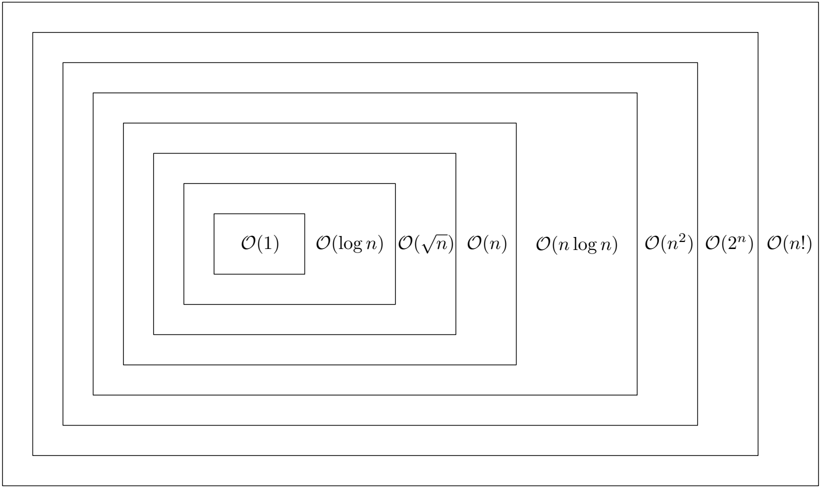
Problem

Show that if $f(n)$ is a polynomial of degree d, that is,
$T(n) = a_d n^d + a_{d-1} n^{d-1} \cdots + a_1 n + a_0$
and $a_d > 0$, then $T(n) \in \mathcal{O}\left(n^d\right)$.

Solution

- For $n \geq 1$, we have $1 \leq n \leq n^2 \leq \cdots \leq n^d$.
- So, $a_d n^d + \cdots + a_1 n + a_0 \leq (|a_d| + \cdots + |a_1| + |a_0|) n^d$
- By choosing $c = (|a_d| + \cdots + |a_1| + |a_0|)$ and $n_0 = 1$, we get $T(n) \in \mathcal{O}\left(n^d\right)$

$\mathcal{O}()$ notation



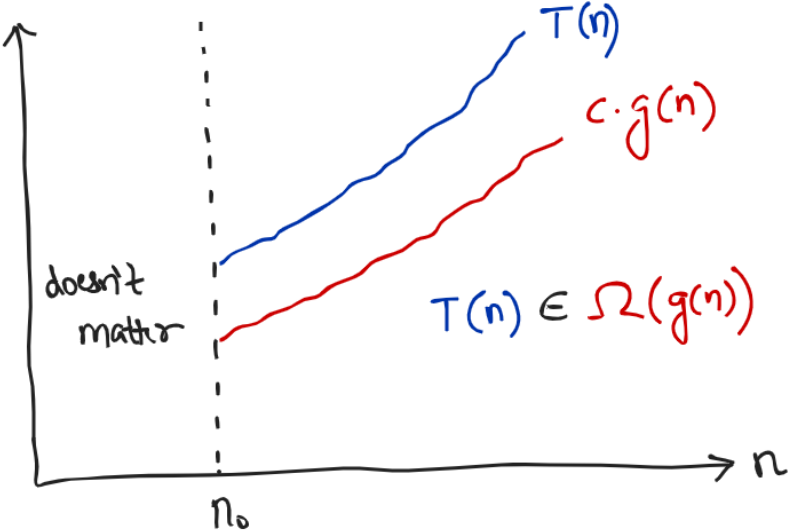
$\Omega\left(\right)$ notation

Definition

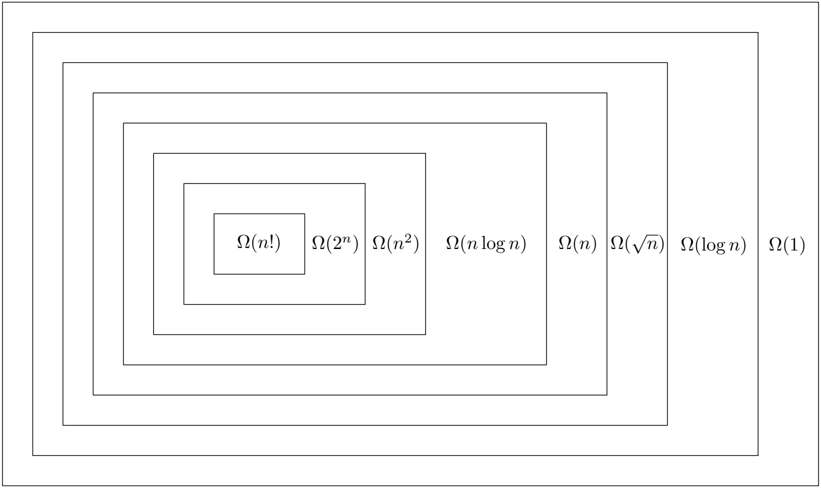
A function $T(n)$ is said to be in $\Omega(g(n))$, denoted $T(n) \in \Omega(g(n))$, if $T(n)$ is bounded below by some constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and nonnegative integer n_0 such that

$$\boxed{T(n) \geq c \cdot g(n) \qquad \text{for all } n \geq n_0}$$

$\Omega()$ notation



$\Omega()$ notation

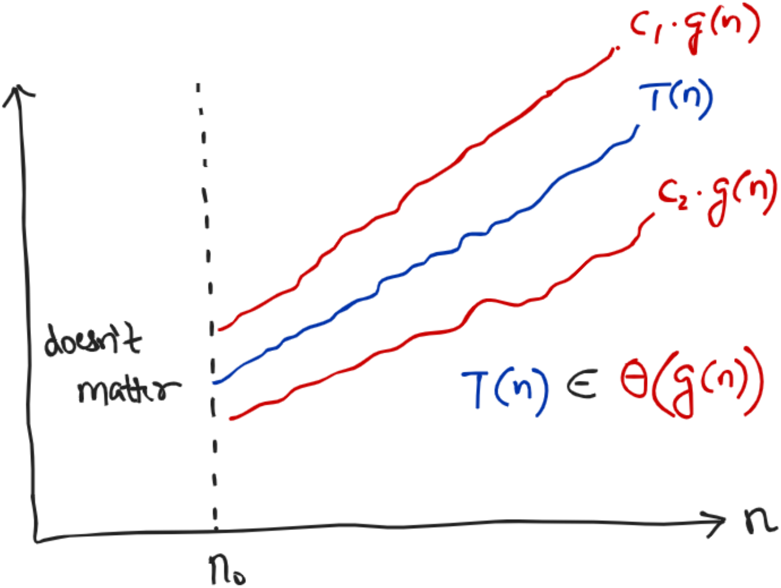


$\Theta\left(\right)$ **notation**

Definition

- A function $T(n)$ is said to be in $\Theta\left(g(n)\right)$, denoted $T(n) \in \Theta\left(g(n)\right)$, if $T(n) \in \mathcal{O}\left(g(n)\right)$ and $T(n) \in \Omega\left(g(n)\right)$.
- A function $T(n)$ is said to be in $\Theta\left(g(n)\right)$, denoted $T(n) \in \Theta\left(g(n)\right)$, if $T(n)$ is bounded both above and below by some constant multiples of $g(n)$ for all large n, i.e., if there exist some positive constants c_1, c_2 and nonnegative integer n_0 such that $\boxed{T(n) \in [c_2 \cdot g(n), c_1 \cdot g(n)] \qquad \text{for all } n \geq n_0}$

$\Theta()$ notation



$\Theta()$ notation

Problem

Show that $\frac{1}{2}n(n-1) \in \Theta\left(n^2\right)$.

Solution

- Step 1. Show that $\frac{1}{2}n(n-1) \in \mathcal{O}\left(n^2\right)$
  $\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$ for all $n \geq 0$
- Step 2. Show that $\frac{1}{2}n(n-1) \in \Omega\left(n^2\right)$
  $\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n = \frac{1}{4}n^2$ for all $n \geq 2$
- As $c_2 = \frac{1}{4}$, $c_1 = \frac{1}{2}$, and $n_0 \geq 2$, we have the result.

$\Theta\left(\right)$ notation

$\Theta(1)$	$\Theta(\log n)$	$\Theta(\sqrt{n})$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(2^n)$	$\Theta(n!)$

Properties

Notation	Reflexivity	Symmetry	Transitivity
$\mathcal{O}\left(\right)$	✓	✗	✓
$\Omega\left(\right)$	✓	✗	✓
$\Theta\left(\right)$	✓	✓	✓

- $f(n) \in \mathcal{O}\left(g(n)\right)$ if and only if $g(n) \in \Omega\left(f(n)\right)$
- If $t_1(n) \in \mathcal{O}\left(g_1(n)\right)$ and $t_2(n) \in \mathcal{O}\left(g_2(n)\right)$, then $t_1(n) + t_2(n) \in \mathcal{O}\left(\max(g_1(n), g_2(n))\right)$
- How do you formally prove the propositions above?

Comparing orders of growth

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies } T(n) \text{ has smaller growth rate than } g(n), \\ c & \text{implies } T(n) \text{ has same growth rate as } g(n), \\ \infty & \text{implies } T(n) \text{ has larger growth rate than } g(n). \end{cases}$$

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \begin{cases} 0 & \text{implies } T(n) \in o\left(g(n)\right), \\ c & \text{implies } T(n) \in \Theta\left(g(n)\right), \\ \infty & \text{implies } T(n) \in \omega\left(g(n)\right). \end{cases}$$

$$\lim_{n\to\infty} \frac{T(n)}{g(n)} = \lim_{n\to\infty} \frac{T'(n)}{g'(n)} \qquad \text{(L'Hôpital's rule)}$$

Example

Problem

- Prove that $\log(n!) \in \mathcal{O}(n \log n)$.

Solution

- Show that $\log(n!) \leq cn \log n$ for some $c > 0$ and $n \geq n_0$
- S.T. $\log(n!) \leq \log((n^n)^c)$
- S.T. $\log(n!) \leq \log(n^n)$ $\triangleright$ set $c = 1$
- S.T. $n! \leq n^n$ $\triangleright$ remove log
- S.T. $\Pi_{i=1}^n i \leq \Pi_{i=1}^n n$
- S.T. $i \leq n$ for all $i \in [1, n]$ $\triangleright$ set $n_0 = 1$
- This is trivially true from the constraints.
- Thus, the theorem follows.

Determining complexities from pseudocodes

Sequence-of-Statements

```
statement s_1
statement s_2
statement s_3
```

$$\text{total time} = \text{time}(s_1) + \text{time}(s_2) + \text{time}(s_3)$$

If-Else-Ladder

```
if condition1 then
  block b_1
else if condition2 then
  block b_2
else
  block b_3
```

$$\text{total time} = \max(\ \text{time}(b_1),\ \text{time}(b_2),\ \text{time}(b_3)\ )$$

Determining complexities from pseudocodes

LOOPS

1. **for** $i \leftarrow 1$ **to** m **do**
2. **for** $j \leftarrow 1$ **to** n **do**
3. block b

$$\text{total time} = mn \times \text{time}(b)$$

(assuming block b takes the same time in every iteration)

FUNCTIONS

1. **for** $i \leftarrow 1$ **to** m **do**
2. **for** $j \leftarrow 1$ **to** n **do**
3. $\mathrm{F}(i,j)$ ▷ Suppose this takes $\Theta(ij)$ time

$$\text{total time} = \sum_{i=1}^{m}\sum_{j=1}^{n} \text{time}(\mathsf{F}(i,j))$$