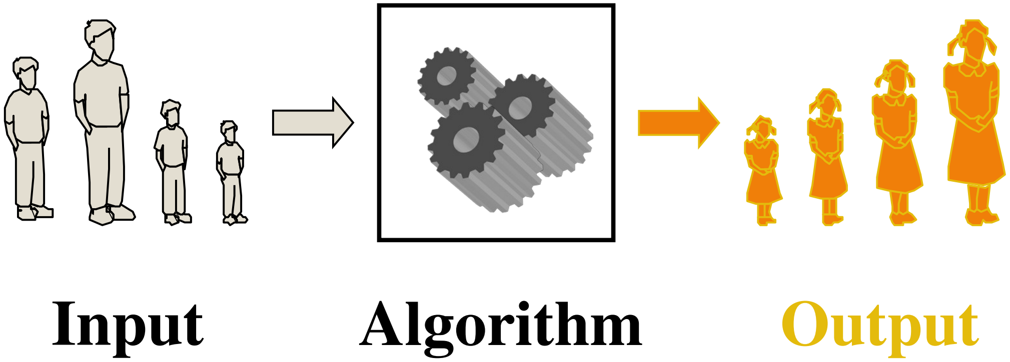
Design and Analysis of Algorithms

Lecture 01: Introduction



Running Time Analysis

- 2 techniques:
  - Experimental Studies
  - Theoretical Analysis

1. Big-Oh Notation (O)

- The function T(n) is O(F(n)) if there exist constants c and N such that $T(n) \le c.F(n)$ for all $n \ge N$.

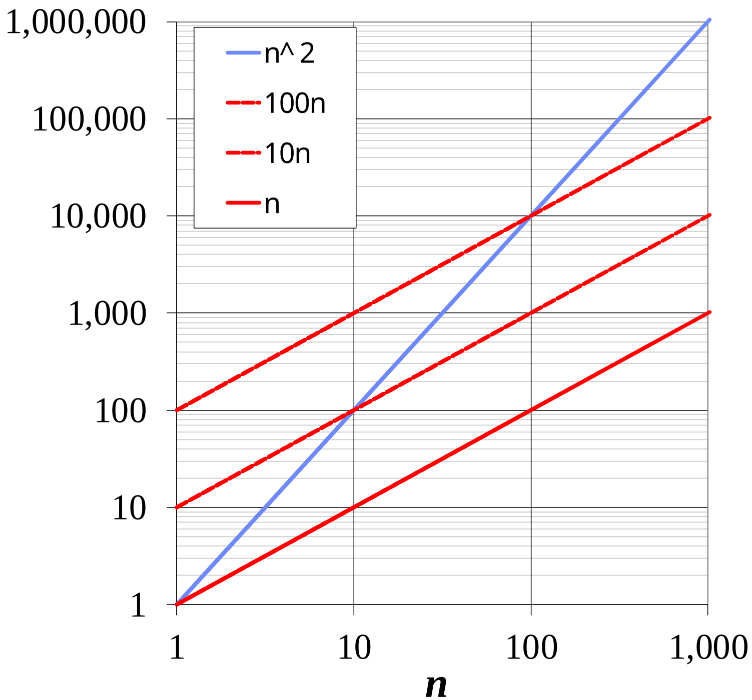
- As n increases, T(n) grows no faster than F(n) or in the long run (for large n) T grows at most as fast as F

- It computes the tight upper bound of T(n)

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \le cn$
  - $(c - 2)\, n \ge 10$
  - $n \ge 10/(c - 2)$
  - Pick $c = 3$ and $N = 10$

1. Big-Oh Notation

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



Properties of Big-Oh Notation

- If T(n) is O(h(n)) and F(n) is O(h(n)) then T(n) + F(n) is O(h(n)).
- The function a.nk is O(n^k) for any a and k
- The function logan is O(logbn) for any positive numbers a and b ≠ 1
- The big-O 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-O notation to rank functions according to their growth rate

Big-Oh Notation

- Exercise:
- Find $F(n)$ such that $T(n) = O(F(n))$ for $T(n) = 3n+5$

2. Big – Omega

- The function T(n) is Ω(F(n)) if there exist Constants c and N such that T(n) ≥ c.F(n) for all n ≥ N.

- As n increases T(n) grows no slower than F(n) or in the long run (for large n) T grows at least as fast as F

- It computes the tight lower bound of T(n).

3. Big Theta Notation

- The function T(n) is Θ(F(n)) if there exist constants c1, c2 and N such that c1.F(n) ≤ T(n) ≤ c2.F(n) for all n ≥ N.
- As n increases, T(n) grows as fast as F(n)
- It computes the tight optimal bound of T(n).
- **Example:**

Find F(n) such that T(n) = Θ(F(n)) for T(n)=$2n^2$.

Solution: c1n2 ≤ $2n^2$ ≤ $c2n^2$

F(n)=n^2 because for c1=1 and c2=3,

n2 ≤ 2n2 ≤ 3n2 for all n.

4. Little notation

- The function T(n) is o(F(n)) if T(n) is O(F(n)) but T(n) is not Θ(F(n) or for some constants c, N: if $T(n) < cF(n)$ for all $n>=N$

- As n increases F(n) grows strictly faster than T(n).

- It computes the non-tight upper bound of T(n)

Example:

- Find F(n) such that T(n) = o(F(n)) for

Solution: n2<c n2

F(n)=n2 because for c=2,

n2<2n2 for all n>=1

Big-Oh Rules

- If f(n) is a polynomial of degree d, then f(n) is $O(n^d)$, i.e.,
  - Drop lower-order terms
  - Drop constant factors
- Use the smallest possible class of functions
  - Say “2n is O(n)” instead of “2n is $O(n^2)$”
- Use the simplest expression of the class
  - Say “3n + 5 is O(n)” instead of “3n + 5 is O(3n)”

Summary 1

Big-Oh

f(n) is O(g(n)) if f(n) is asymptotically less than or equal to g(n)

big-Omega

f(n) is Ω (g(n)) if f(n) is asymptotically greater than or equal to g(n)

big-Theta

f(n) is Θ (g(n)) if f(n) is asymptotically equal to g(n)

little-oh

f(n) is o(g(n)) if f(n) is asymptotically strictly less than g(n)

little-omega

f(n) is w(g(n)) if is asymptotically strictly greater than g(n)

Summary 2

T(n)	Complexity Category Function (F(n))	Big Oh of T(n)
$10\log(n+5)$	Logn	$T(n)=O(\log n)$
C where C is constant	1	$T(n) = O(1)$
$5\sqrt{n}+5$	$\sqrt{n}$	$T(n) = O(\sqrt{n})$
$5/3(n)+2\log n+2$	N	$T(n) = O(n)$
$10n\log n+2n+2$	nlogn	$T(n)=O(n\log n)$
$3n^2+2n+2$	n^2	$T(n)=O(n^2)$
$5n^3\ 3$	n^3	$T(n) = O(n^3)$
$8n^n + 2^n n^2$	n^n	$T(n)=O(n^n)$

Summary 3 Time Complexity Comparison

- **A fast algorithm has a small time complexity, while a slow algorithm has large time complexity.**

- **Comparison of algorithm speed (fastest to slowest):**

Constant ≈ 1 (fastest)

Logarithmic ≈ log n

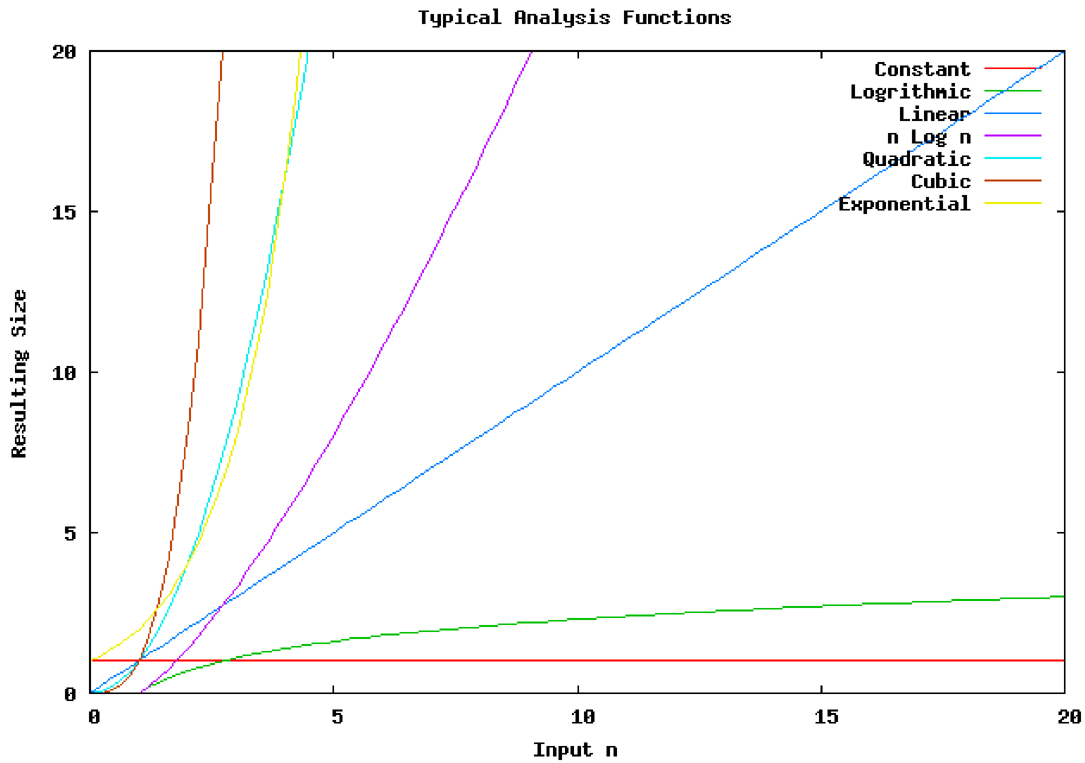
Linear ≈ n

N-Log-N ≈ n log n

Quadratic ≈ n^2

Cubic ≈ n^3

Exponential ≈ 2^n (slowes



Exercise 1: **Finding Big O of given Algorithm**

for (i=1;i<=n;i++)

Cout<<i;

T(n) = 1+n+1+n+n

=3n+2

T(n) = O(n)

2. For (i=1;i<=n;i++)

For(j=1;j<=n;j++)

Cout<<i;

T(n)=1+n+1+n+n(1+n+1+n+n)

=3n2 +4n+2

T(n)=O(n2)

Exercise 1: **Finding Big O of given Algorithm**

3. for (i=1; i<=n; i++)

Cout<<i;

T(n) = 1+n+1+n+n

=3n+2

T(n) = O(n)

Thank You