Foundations of Algorithm SCS1308

Dr. Dinuni Fernando
PhD

Senior Lecturer

Lecture 2

UCSC

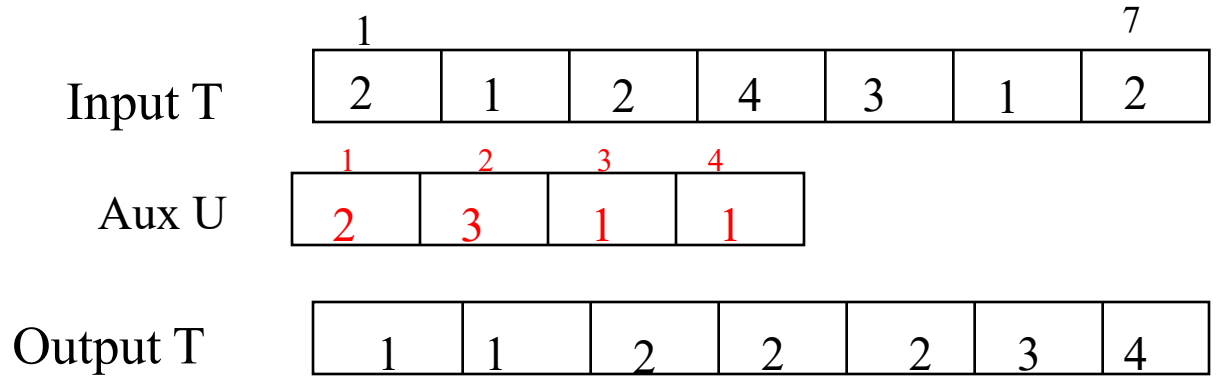
Example of Count: Counting sort

Input: Array *T* containing *n* keys in ranges 1.. *S*

Idea: (Similar to Histogram)

1) Maintain the count of number of keys in an auxiliary array U

2) Use counts to overwrite array in place



Example : counting sort

- T = [2, 1, 4, 3, 2] and s = 4
- **Step 1:** Initialize U = [0, 0, 0, 0]
  Step 2: Count
- T[1]=2 → U[2]++ → U=[0,1,0,0]
- T[2]=1 → U[1]++ → U=[1,1,0,0]
- T[3]=4 → U[4]++ → U=[1,1,0,1]
- T[4]=3 → U[3]++ → U=[1,1,1,1]
- T[5]=2 → U[2]++ → U=[1,2,1,1]
- **Step 3:** Rewrite
  → Sorted T = [1, 2, 2, 3, 4]

Algorithm:

Counting-Sort(T, s)

1. for $i = 1$ to s //initialize U
   $U[i] = 0$
2. for $j = 1$ to n // n = $length[T]$
3. $U[T[j]] = U[T[j]] + 1$ //Count keys
4. q = 1
5. for $j = 1$ to s //rewrite T
6. while U[j] > 0
7. $T[q] = j$
8. $U[j] = U[j] - 1$
9. q = q + 1

Barometer operation

O(s)

O(n)

O(n+s)

Asymptotic Growth Rate

Asymptotic Notation

- Asymptotic notation does not provide an exact running time or space usage for an algorithm, but rather a description of how the algorithm scales with respect to input size.
- A useful tool for comparing the efficiency of different algorithms and for predicting how they will perform on large input sizes.

Big – O (Big-Oh)

- Upper bound on the growth rate of an algorithm's running time or space usage.
- Represent worst case scenario - the maximum amount of time or space an algorithm may need to solve a problem.
- Eg: What does it mean to say O(n) ?
  - It means that the running time of the algorithm increases linearly with the input size n or less.

Example 1 :

- Find upper bound for f(n) = 3n + 8

Solution: $3n + 8 \leq c.g(n)$, for all $n \geq n_0$

$\therefore 3n + 8 = O(n)$ with $c = 4$ and $n_0 = 8$

Example 1 :

- Find upper bound for $f(n) = n^4 + 100n^2 + 50$

Solution: $n4 + 100n2 + 50 \leq cg(n)$, for all $n \geq 1$

$\therefore n4 + 100n2 + 50 = O(n4)$ with $c = 151$ and $n0 = 1$

Does 5n+2 ∈O(n)?

Proof: From the definition of Big Oh, there must exist c > 0 and integer $N > 0$ such that $0 \leq 5n+2 \leq cn$ for all $n \geq N$.

Dividing both sides of the inequality by n > 0 we get:

$0 \leq 5+2/n \leq c$.

- $2/n$ (> 0) becomes smaller as n increases
- For instance, let N = 2 and c = 6

There are many choices here for c and N.

Does $n^2 \in O(n)$?

Does $n^2 \in O(n)$? No.

We will prove by contradiction that the definition cannot be satisfied.

- Assume that $n^2 \in O(n)$. From the definition of Big Oh, there must exist $c > 0$ and integer $N > 0$ such that $0 \leq n^2 \leq cn$ for all $n \geq N$.
- Divide the inequality by $n > 0$ to get $0 \leq n \leq c$ for all $n \geq N$.
- $n \leq c$ cannot be true for any $n > \max\{c, N\}$. This contradicts the assumption. Thus, $n^2 \notin O(n)$.

Are they true? Why or why not?

- **1,000,000 n^2 $\in O(n^2)$?**
- **True**

- **$(n - 1)n / 2 \in O(n^2)$?**
- **True**

- **$n / 2 \in O(n^2)$?**
- **True**

- **$\lg(n^2) \in O(\lg n)$?**
- **True**

- **$n^2 \in O(n)$?**
- **False**

Omega notation (Ω)

- Provides a lower bound on the growth rate of an algorithm's running time or space usage.
- Represent best case scenario – minimum amount of time or space an algorithm may need to solve a problem.
- Eg: What does it mean to say Ω(n) ?
  - It means that the running time of the algorithm increases linearly with the input size n or more.

Example 3 :

- Find lower bound for $f(n) = 5n^2$

Solution: $\exists\ c, n0$ Such that: $0 \leq cn^2 \leq 5n^2 \Rightarrow cn^2 \leq 5n^2 \Rightarrow c = 5$ and $n0 = 1$

$\therefore 5n^2 = \Omega(n^2)$ with $c = 5$ and $n0 = 1$

Are they true?

- **1,000,000 $n^2 \in \Omega(n^2)$ why /why not?**
  - **true**
- **$(n - 1)n / 2 \in \Omega(n^2)$ why /why not?**
  - **true**

- **$n / 2 \in \Omega(n^2)$ why /why not?**
  - **(false)**
- **$\lg(n^2) \in \Omega(\lg n)$ why /why not?**
  - **(true)**
- **$n^2 \in \Omega(n)$ why /why not?**
  - **(true)**

Theta notation (Θ)

- Provides both an upper and lower bound on the growth rate of an algorithm's running time or space usage.
- Represents the average-case scenario, i.e., the amount of time or space an algorithm typically needs to solve a problem
- Eg: What does it mean to say Θ(n) ?

Then it means that the running time of the algorithm increases linearly with the input size n.

Example 4 :

- Find Θ bound for f(n) = $\frac{n^2}{2} - \frac{n}{2}$

we have $c_1 n^2 \leq f(n) \leq c_2 n^2$ for all $n \geq n_0$,so $f(n) = \Theta(n^2)$

For $c_1 = \frac{1}{4},\ c_2 = \frac{1}{2},\ n_0 = 2$

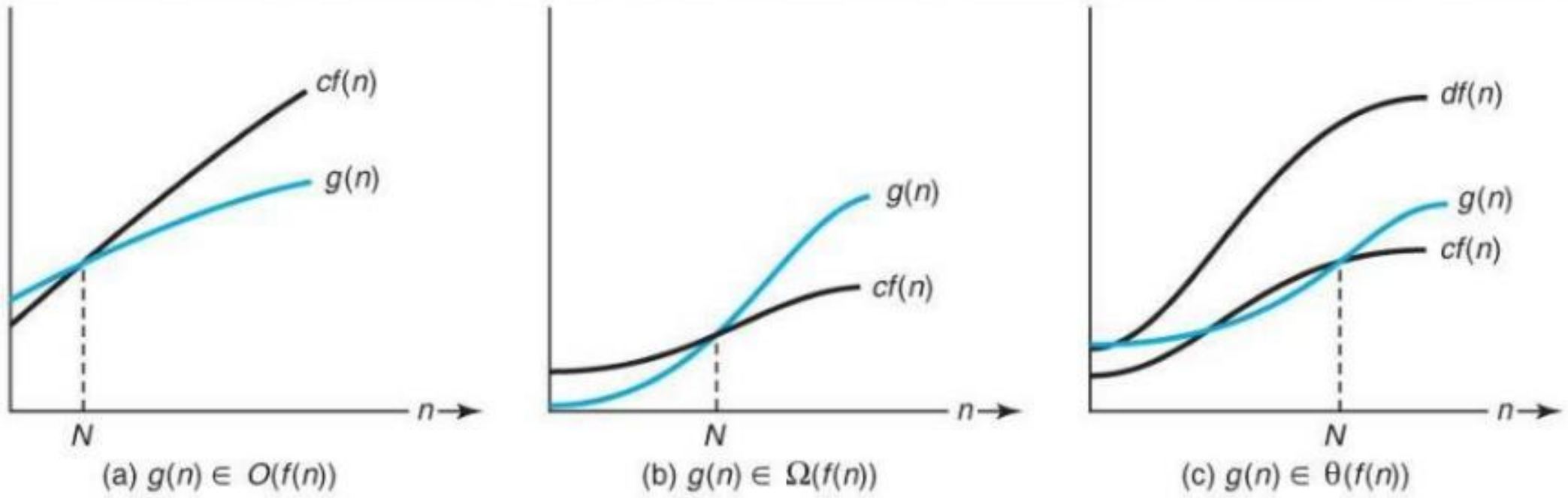


Illustration : Big O, Omega and Theta

small o

- $o(f(n))$ is the set of functions $g(n)$ which satisfy the following condition:

  For *every* positive real constant c, there exists a positive integer N, for which

  $$g(n) \leq cf(n) \text{ for all } n \geq N$$

Big O vs. Small o

Big O

- f(n)=O(g(n)) means f(n) grows at most as fast as g(n)

$$f(n) \leq C \cdot g(n) \quad \text{for all } n \geq n_0,$$

- Big O gives an upper bound on f(n)
- f(n) grow as fast as g(n) but not faster.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq C, \quad C > 0.$$

Small o

- f(n) = o(g(n)) means f(n) grows strictly slower than g(n)

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

- Small o gives a strict upper bound
- f(n) is negligible compared to g(n) as n → ∞.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

small omega

- $\omega(f(n))$ is the set of functions $g(n)$ which satisfy the following condition:

  For *every* positive real constant c, there exists a positive integer N, for which

  $g(n) \geq cf(n)$ for all $n \geq N$

Omega Ω vs. Small ω

Omega Ω

- f(n)=Ω (g(n)) means f(n) grows at least as fast as g(n)

$$f(n) \geq C \cdot g(n) \quad \text{for all } n \geq n_0,$$

- Omega gives an lower bound on f(n)
- f(n) grow as fast or at the same rate as g(n) but not slower.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq C,$$
$$C > 0.$$

Small ω

- f(n) = ω(g(n) means f(n) grows strictly faster than g(n)

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

- Small ω gives a strict lower bound
- f(n) must grow strictly faster than g(n), g(n) becomes negligible compared to g(n) as n→ ∞.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Analogy between asymptotic comparison of functions and comparison of real numbers.

$$f(n) = O(g(n)) \quad \approx \quad a \leq b$$
$$f(n) = \Omega(g(n)) \quad \approx \quad a \geq b$$
$$f(n) = \Theta(g(n)) \quad \approx \quad a = b$$
$$f(n) = o(g(n)) \quad \approx \quad a < b$$
$$f(n) = \omega(g(n)) \quad \approx \quad a > b$$

f(n) is asymptotically smaller than g(n) if $f(n) = o(g(n))$

f(n) is asymptotically larger than g(n) if $f(n) = \omega(g(n))$

Order of Algorithm

- Property
  - Complexity Categories:

$\theta(\lg n)$ $\theta(n)$ $\theta(n \lg n)$ $\theta(n^2)$ $\theta(n^j)$ $\theta(n^k)$ $\theta(a^n)$ $\theta(b^n)$ $\theta(n!)$

Where $k>j>2$ and $b>a>1$. If a complexity function g(n) is in a category that is to the left of the category containing f(n), then $g(n) \in o(f(n))$

Thank you