Algorithms and Data Structures
Lecture slides: Asymptotic notations and growth rate of functions, Brassard Chap. 3

Lecturer: Michel Toulouse

Hanoi University of Science & Technology
michel.toulouse@soict.hust.edu.vn

Topics outline

Introduction

Big O-Notation

Omega-Notation

Theta-Notation

Some exercises

Algorithms performance in terms of growth rate ("orders")

The growth rate measures by how much the number of basic operations increases for a given increase in the input size

Growth rate is not strictly a measure of the number of basic operations for a given input size n, rather it is a bound on the rate of change in the running time over the continuum for all input sizes

For example, the growth rate of an expression like $an^2 + bn + c$ is n^2.

This is because we can bound $an^2 + bn + c$ by another expression ln^2 which is larger than $an^2 + bn + c$ for a sufficiently large constant l (such as $l = a + b + c$)

Algorithms performance in terms of growth rate (continue)

Given an input size n, the growth rate is a function that bound above (below, exactly) the number of basic operations executed by an algorithm

The constant l is only needed to prove the correctness of a bound, in the expression of the growth rate this constant is removed, in the previous example the growth rate is just n^2

In final analysis, the performance of algorithms is compared according to their respective growth rate.

Frequent “orders” or ”growth rates”

Some growth rates occur so frequently that we give them a name. Let c be some arbitrary constant.

- Logarithmic : An algorithm never executes more than $c \log n$ basic operations.
- Linear : An algorithm never executes more than cn basic operations.
- Quadratic : An algorithm never executes more than cn^2 basic operations.
- Cubic, polynomial or exponential : Algorithms that never execute more than cn^3, cn^k or ck^n basic operations.

Order notations : big O

The Order notations are O, Ω, Θ. They identify the type of bounds that is placed on the number of basic operations (running time) of an algorithm

When the running time of an algorithm is bound above by a function like $\log n$, $n \log n$, n^2, etc, we denote the order of the corresponding algorithm as $O(\log n)$, $O(n \log n)$ or $O(n^2)$ which we pronounce as big O of some function.

Order notations : Ω

The running time of an algorithm can be bound below by some function $g(n)$, i.e. the number of times the basic operation is executed is at least $g(n)$

We denote the running time of the algorithm as *omega* (Ω) of $g(n)$ such as $\Omega(\log n)$, $\Omega(n \log n)$ or $\Omega(n^2)$

Order notations : Θ

Finally, if the running time of an algorithm can be bound above and below by a same function $g(n)$, we have an exact bound on the running time, this is denoted as *theta* (Θ) of that function $g(n)$ such as $\Theta(\log n)$, $\Theta(n \log n)$ or $\Theta(n^2)$

Order notations

There is another characterization that is imposed on the usage of the notations O, Ω and Θ.

These notations are called asymptotic notations because the bounds on the running of algorithms must be verified as the input size increases to infinity. Often the bounds are not verified for small input sizes

The next slides define these notations in mathematical terms.

O-Notation : An Asymptotic Upper Bound I

Definition (Big O notation)

Let $g(n)$ be a function from $\mathbb{N}$ to $\mathbb{R}$. Denote

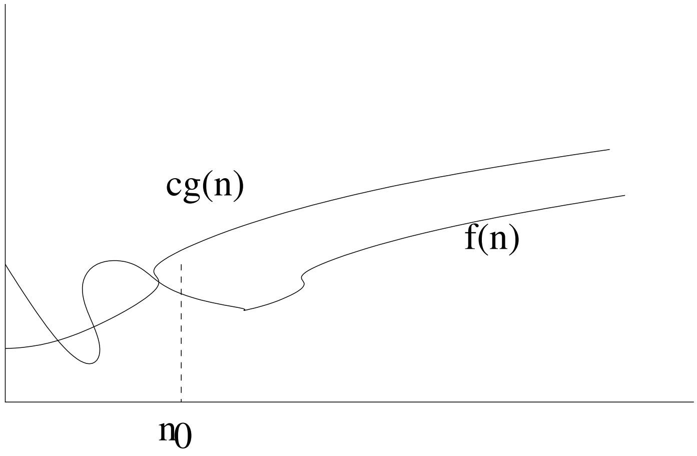
$$O(g(n)) = \{f(n) : \text{there exist positive constant } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c\,g(n) \text{ for all } n \geq n_0\}$$

the set of functions defined on natural numbers which are bounded above by a positive real multiple of $g(n)$ for sufficiently large n.

Graphic Example of O-notation

- $f(n) \in O(g(n))$ if there are constants c and n_0 such that

$$0 \leq f(n) \leq c\,g(n) \text{ for all } n_0 \leq n$$



Example : Let $f(n) = 27n^2 + \frac{355}{113}n + 12$ be the number of times a basic operation is performed by an algorithm in the worst case, giving an input of size n. We would like to find a simple function $g(n)$ such that $f(x) \in O(g(n))$.

We can guess $g(n) = n^2$. Thus,

$$\begin{aligned} f(n) &= 27n^2 + \tfrac{355}{113}n + 12 \\ &\leq 27n^2 + \tfrac{355}{113}n^2 + 12n^2 \\ &\leq 42\tfrac{16}{113}n^2 \end{aligned}$$

So instead of saying that an algorithm takes $27n^2 + \frac{355}{113}n + 12$ elementary operations to solve an instance of size n, we can say that the time of the algorithm is in order of n^2, or write the algorithm is in $O(n^2)$.

How to find the growth rate $g(n)$ of an algorithm

Assume the running time is $f(n) = 3n^2 + 2n$:

- Throw away the multiplicative constants : $3n^2 + 2n$ is replaced by $n^2 + n$
  - Also if you have $2^{n+1} = 2 \times 2^n$ can be replaced by 2^n.
  - If you have logs, throw away the bases since the log properties says that for any two bases a and b, $\log_b n = c \times \log_a n$ for some multiplicative constant c.
- Once $f(n)$ has been simplified, the fastest growing term in $f(n)$ is the growth rate $g(n)$.
- $f(n) = 3n^2 + 2n = O(n^2)$.

OK, this is a ways to find $g(n)$, but this is not a proof that $f(n) \in O(g(n))$

Prove that $f(n) \in O(g(n))$

Often the easiest way to prove that $f(n) \in O(g(n))$ is to take c to be the sum of the positive coefficients of $f(n)$.

Example : Prove $5n^2 + 3n + 20 \in O(n^2)$

- We pick $c = 5 + 3 + 20 = 28$. Then if $n \geq n_0 = 1$,

  $$5\,n^2 + 3\,n + 20 \leq 5\,n^2 + 3\,n^2 + 20\,n^2 = 28\,n^2,$$

  thus $5n^2 + 3n + 20 \in O(n^2)$.
- We can also guess other values for c and then find n_0 that work.

Prove that $f(n) \in O(g(n))$

Another way is to assume $c = 1$ and find for which n_0 $f(n) \leq g(n)$

Example : Show that $\frac{1}{2}n^2 + 3n \in O(n^2)$

Proof : The dominant term is $\frac{1}{2}n^2$, so $g(n) = n^2$. Therefore we need to find c and n_0 such that

$$0 \leq \frac{1}{2}n^2 + 3n \leq c\, n^2 \text{ for all } n \geq n_0.$$

Since we decided to fix $c = 1$, we have

$$\frac{1}{2}n^2 + 3n \leq n^2 \Leftrightarrow 3n \leq \frac{1}{2}n^2 \Leftrightarrow 6 \leq n$$

Thus, we pick $n_0 = 6$.

We have just shown that if $c = 1$, and $n_0 = 6$, then $0 \leq \frac{1}{2}n^2 + 3n \leq c\, n^2$ for all $n \geq n_0$, i.e. $\frac{1}{2}n^2 + 3n \in O(n^2)$.

Exercises on Big O I

1. Given the following algorithm written in pseudo code :

```
$t := 0$;
**for** $i := 1$ **to** $n$ **do**
    **for** $j := 1$ **to** $n$ **do**
        $t := t + i + j$;
**return** $t$.
```

   1.1 Which instruction can be used as elementary operation ?
   1.2 Express the running time of this algorithm in terms of the number of times your selected elementary operation is executed ?
   1.3 Give (without proof) a big $\mathcal{O}$ estimate for the running time of the algorithm.
   1.4 What is computed and returned by this algorithm ?

Exercises on Big O I

2. Find $g(n)$ for each of the following functions $f_i(n)$ such that $f_i(n) \in O(g(n))$.
   - $f_1 = 3n\log_2 n + 9n - 3\log_2 n - 3$
   - $f_2 = 2n^2 + n\log_3 n - 15$
   - $f_3 = 100n + (n+1)(n+5)/2 + n^{3/2}$
   - $f_4 = 1,000n^2 + 2^n + 36n\log n + \left(\frac{3}{2}\right)^{n+1}$

Exercises on Big O II

3. Which of the following statements are true ?

- $n^2 \in O(n^3)$
- $2^n \in O(3^n)$
- $3^n \in O(2^n)$
- $n \log n \in O(n^{\frac{3}{2}})$
- $n^{\frac{3}{2}} \in O(n \log n)$
- $2^{n+1} \in O(2^n)$
- $O(2^{n+1}) = O(2^n)$
- $O(2^n) = O(3^n)$

Exercises on Big O III

4. Give an upper bound on the worst-case asymptotic time complexity of the following function used to find the Kth smallest integer in an unordered array of integers. Justify your result. You do not need to find the closed form of summations.

```
int selectkth( int A[ ], int k, int n )
  int i, j, mini, tmp ;
  for ( i = 0 ; i < k ; i++ )
    mini = i ;
    for ( j = i + 1 ; j < n ; j++ )
      if ( A[j] < A[mini] )
        mini = j ;
        tmp = A[i] ;
        A[i] = A[mini] ;
        A[mini] = tmp ;
  return A[k-1] ;
```

Exercises on Big O IV

5. How $n \log n$ compares with $n^{1.\epsilon}$ for $0 < \epsilon < 1$?

   Answer : Note that $n \log n = n \times (\log n)$ and $n^{1.\epsilon} = n \times n^{\epsilon}$.

   The grow rate of $\log n$ is slower than n^{ϵ} for any value of $\epsilon > 0$

   Eventually n^{ϵ} catch up with $\log n$ for some value of $n > n_0$, depending on how small is ϵ.

   Therefore, $n \log n \in O(n^{1.\epsilon})$ for $0 < \epsilon$.

Exercises on Big O V

6. Find the appropriate "Big-Oh" relationship between the functions $n \log n$ and $5n$ and find the constants c and n_0

Exercises on Big O VI

7. Give the polynomial expression describing the running time of the code below. Provide the asymptotic time complexity of this code using the "Big-Oh" notation.

```
for ($i = 0; i < n; i++$){
    for ($j = 0; j < 2 * n; j++$)
        $sum = sum + A[i] * A[j]$
    for ($j = 0; j < n * n; j++$)
        $sum = sum + A[i] + A[j]$
}
```

Big Omega (Ω) : An Asymptotic Lower Bound

Given a non-negative valued function $g(n)$. Denote

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constant } c \text{ and } n_0 \text{ such that } f(n) \geq c\, g(n) \text{ for all } n \geq n_0\}$$

Definition

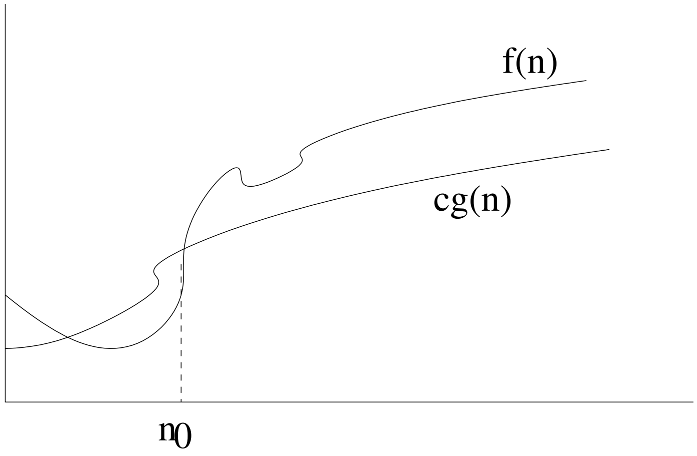
Let f and g be non-negative valued functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$:

1. We say that $f(n)$ is in omega of $g(n)$ if $f(n) \in \Omega(g(n))$.
2. As n increases, $f(n)$ grows no slower than $g(n)$. In other words, $g(n)$ is an asymptotic lower bound of $f(n)$.

Graphic Example of Ω-notation

- $f(n) = \Omega(g(n))$ if there are constants c and n_0 such that

$$0 \le c\,g(n) \le f(n) \text{ for all } n \ge n_0.$$



Big Omega : Examples

1. $f(n) = 3n^2 + n + 12$ is $\Omega(n^2)$ and also $\Omega(n)$, but not $\Omega(n^3)$.

2. $n^3 - 4n^2 \in \Omega(n^2)$.

   Proof : Let $c = 1$. Then we must have

$$\begin{aligned} cn^2 &\leq n^3 - 4n^2 \\ 1 &\leq n - 4 \end{aligned}$$

   which is true when $n \geq 5$, therefore $n_0 = 5$
   so

$$0 \leq n^2 \leq n^2\,(n-4) = n^3 - 4\,n^2$$

Ω Proofs : How to choose c and n_0

To prove that $f(n) \in \Omega(g(n))$, we must find positive values of c and n_0 that make $c \cdot g(n) \leq f(n)$ for all $n > n_0$.

- You can assume that $c < 1$, pick a n_0 such that $f(n)$ is larger than $c \cdot g(n)$ and then find the exact constant c for n_0, OR
- Choose c to be some positive constant less than the multiplicative constant of the fastest growing term of $f(n)$, then find n_0 that works with the chosen c.

Example 1

For this example we assume that $c < 1$ and find an appropriate n_0

Show that $(n \log n - 2\,n + 13) \in \Omega(n \log n)$

Proof : We need to show that there exist positive constants c and n_0 such that

$$0 \leq c\,n \log n \leq n \log n - 2\,n + 13 \text{ for all } n \geq n_0.$$

Since $\quad n \log n - 2\,n \leq n \log n - 2\,n + 13,$

we will instead show that

$$c\,n \log n \leq n \log n - 2\,n,$$

Example 1 continue

$$\begin{aligned} c\, n \log n &\leq n \log n - 2\, n \\ c &\leq \frac{n \log n}{n \log n} - \frac{2n}{n \log n} \\ c &\leq 1 - \frac{2}{\log n} \end{aligned}$$

so, $c \leq 1 - \frac{2}{\log n}$, when $n > 1$.

If $n \geq 8$, then $2/(\log n) \leq 2/3$, and picking $c = 1/3$ suffices.

Thus if $c = 1/3$ and $n_0 = 8$, then for all $n \geq n_0$, we have

$$0 \leq c\, n \log n \leq n \log n - 2\, n \leq n \log n - 2\, n + 13.$$

Thus $(n \log n - 2\, n + 13) \in \Omega(n \log n)$.

Example 2

For this example we select c to be smaller than the constant of the fastest growing term in the expression describing the running time.

Prove that $f(n) = 3n^2 - 2n - 7 \in \Omega(n^2)$.

Proof : The fastest growing term of $f(n)$ is $3n^2$. Try $c = 1$, since $1 < 3$.

Then

$$1 \cdot n^2 \leq 3n^2 - 2n - 7 \quad \text{for all } n > n_0$$

is true only if (subtracting n^2 from both sides)

$$0 \leq 2n^2 - 2n - 7 \quad \text{for all } n > n_0$$

is also true.

Choose $n_0 = 3$, then the inequality above hold for any $n \geq 3$.

An Asymptotic Tight Bound Θ-notation

Let $g(n)$ be a non-negative valued function. Denote

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ s.t. } c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ for all } n \geq n_0\}$$

Definition

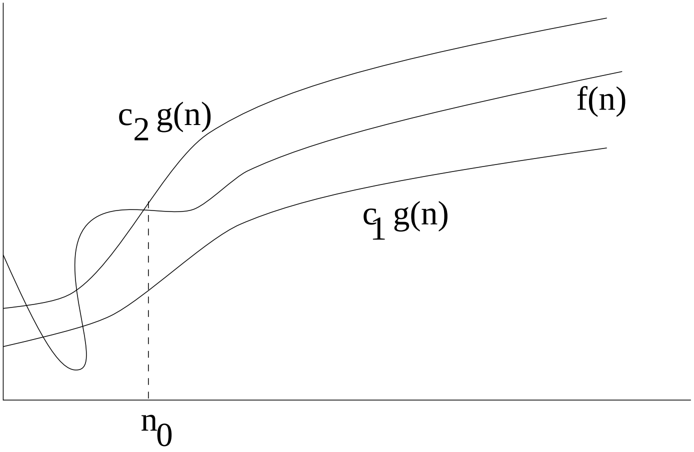
Let f and g be non-negative valued functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$:

1. We say that $f(n)$ is in theta of $g(n)$ if $f(n) \in \Theta(g(n))$.
2. As n increases, $f(n)$ grows at the same rate as $g(n)$. In other words, $g(n)$ is an asymptotic tight bound of $f(n)$.

Graphic Example of Θ-notation

- $f(n) = \Theta(g(n))$ if there are constants c_1, c_2 and n_0 such that

$$0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n) \text{ for all } n_0 \leq n$$



Θ-notation : Example

Prove that $n^2 - 5n + 7 \in \Theta(n^2)$.

Proof : Let $c_1 = \frac{1}{2}$, $c_2 = 1$, and $n_0 = 10$. Then $\frac{1}{2}n^2 \geq 5n$ and $-5n + 7 \leq 0$. Thus,

$$0 \leq \frac{1}{2}n^2 \leq n^2 - \frac{1}{2}n^2 \leq n^2 - 5n \leq n^2 - 5n + 7 \leq n^2$$

If $f(n)$ is $\Theta(g(n))$, then

- $f(n)$ is “sandwiched” between $c_1 g(n)$ and $c_2 g(n)$ for sufficiently large n;
- $g(n)$ is an asymptotically tight bound for $f(n)$;

Big Theta Proofs

The following theorem shows us that proving $f(n) \in \Theta(g(n))$ is nothing new :

- **Theorem :** $f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.
- Thus, we just apply the previous two strategies.

Example

Show that $\frac{1}{2}n^2 - 3n \in \Theta(n^2)$

Proof :

- Find positive constants c_1, c_2, and n_0 such that

$$0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3\,n \leq c_2 n^2 \text{ for all } n \geq n_0$$

- Dividing by n^2, we get $0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$
- $c_1 \leq \frac{1}{2} - \frac{3}{n}$ holds for $n \geq 10$ and $c_1 = 1/5$
- $\frac{1}{2} - \frac{3}{n} \leq c_2$ holds for $n \geq 10$ and $c_2 = 1$.
- Thus, if $c_1 = 1/5$, $c_2 = 1$, and $n_0 = 10$, then for all $n \geq n_0$,

$$0 \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3\,n \leq c_2 n^2 \text{ for all } n \geq n_0.$$

Thus we have shown that $\frac{1}{2}n^2 - 3n \in \Theta(n^2)$.

Cookbook for asymptotic notations

Theorem (Limit rule)

Given non-negative valued functions f and $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$. Then the following statements are true

1. *if* $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} = L < \infty$, *then* $f(n) \in \Theta(g(n))$ *and consequently* $f(n) \in O(g(n))$ *and* $f(n) \in \Omega(g(n))$.
2. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *then* $f(n) \in O(g(n))$.
3. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$, *then* $f(n) \notin O(g(n))$ *but* $g(n) \in O(f(n))$ *and* $f(n) \in \Omega(g(n))$.

Exercises

1. Prove that $f(n) = n^3 + 20n + 1 \in O(n^3)$
2. Prove that $f(n) = n^3 + 20n + 1 \notin O(n^2)$
3. Prove that $f(n) = n^3 + 20n + 1 \in O(n^4)$.
4. Prove $f(n) = n^3 + 20n \in \Omega(n^2)$.
5. Prove $f(n) = \frac{1}{2}n^2 - 3n \in \Omega(n^2)$.
6. Prove that $f(n) = 5n^2 - 7n \in \Theta(n^2)$.
7. Prove that $f(n) = 23n^3 - 10n^2 \log n + 7n + 6 \in \Theta(n^3)$.
8. Find the appropriate Ω relationship between the functions n^3 and $3n^3 - 2n^2 + 2$ and find the constants c and n_0.

Exercises (continue)

9. Consider the following iterative procedure :

```
for ($i = 0; i < n; i++$){
    for ($j = 0; j < 2 * n; j++$)
        $sum = sum + A[i] * A[j]$
    for ($j = 0; j < n * n; j++$)
        $sum = sum + A[i] + A[j]$
}
```

9.1 Give a function f describing the computing time of this procedure in terms of the input size n.

9.2 Bound above the running time of this code using the "Big-Oh" notation. Prove your result.

9.3 Give a lower bound on the running time of this code using the "Ω" notation. Prove your result. Then argue, based on your two previous results about an exact time complexity of f

Exercises (continue)

10. To illustrate how the asymptotic notation can be used to rank the efficiency of algorithms, use the relation $\subset$ and $=$ to put the orders of the following functions into a sequence.

$$n^2,\ 1,\ n^{3/2},\ 2^n,\ \log n,\ n^n,\ 3^n,\ n,\ n^3,\ n\log n,\ \sqrt{n}, \log\log n, n!$$

11. Similar to previous question, order the following growth rate functions

$$n! \quad (n+1)! \quad 2^n \quad 2^{n+1} \quad 2^{2n} \quad n^n \quad n^{\sqrt{n}} \quad n^{\log n}.$$