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

1. Introduction : The principle of invariance
2. Asymptotic notations : definitions, examples, properties
   - Big O-notation
   - Ω-notation
   - Θ-notation
3. Cookbook for asymptotic notations : Limit rule
4. Exercises

thro

Running time

In the previous lecture, the running time of an algorithm was obtained by figuring out the worst case instance and then counting the number of times the most frequent basic operations is executed

We did agree this was not an exact measurement of the execution time of an algorithm because

1. we count as "basic" pseudo-code operations that may take quite different numbers of machine operations
2. we don't count all the basic operations of an algorithm but only the one that is executed the most often

From running time to growth rate to "orders"

Here we move even further in abstracting measurements from real execution time

The number of basic operations executed in term of n (the input size) is called the "growth rate" of an algorithm

The growth rate of different algorithms is in the same "order" if bound by a same function which differs from the growth rates by only a multiplicative constant

In other words, two algorithms for which the growth rate differs only by a multiplicative constant are considered to be in the same "order"

The Principle of Invariance

We know that if we use two different computers to execute an algorithm, one may run faster than the other

The principle of invariance states that the time used to execute the algorithm on two different computers will not differ by more than some multiplicative constant

For example, if the constant happen to be 2, than we know that the slower execution will not take more than 2 times the time needed for the other execution. If the fastest execution takes 1 second, the other one takes 2 seconds

The Principle of Invariance : Formally

Assume the fastest execution takes $t_1(n)$ seconds for a problem instance of size n and the slowest execution takes $t_2(n)$ seconds.

The principle of invariance is based on the following observation : Given what is stated in (1), there always exist positive constants c and d such that $t_1(n) \leq ct_2(n)$ and such that $t_2(n) \leq dt_1(n)$ for n sufficient large.

The running time of either execution is bounded by a constant multiple of the running time of the other, which one we should call the first is irrelevant !

Principle of Invariance : "orders"

One of the most important implication of the principle of invariance is that *we don't need to use any time unit* to express the efficiency of an algorithm.

For example, if an algorithm takes $t_1(n)$ seconds, than using a constant b, c or d we can say that it takes $bt_1(n)\mu$s, $ct_1(n)$ ns or $dt_1(n)$ year.

Rather we choose to express the time taken by an algorithm *within a multiplicative constant*, i.e. a time *in the order of* $t(n)$ for a given function t, a positive constant c and an implementation of the algorithm capable to solve every instance of size n in no more than $ct(n)$ seconds (the use of seconds here is completely arbitrary).

Frequent "orders"

Some orders occur so frequently that we give them a name.

- Logarithmic algorithm : If an algorithm never takes more than $c \log n$ basic operations (therefore the algorithm takes a time in the order of $\log n$ or logarithmic time).
- Linear algorithm : If an algorithm never takes more than cn basic operations (therefore the algorithm takes a time in the order of n or linear time).
- Quadratic algorithm : If an algorithm never takes more than cn^2 basic operations (therefore the algorithm takes a time in the order of n^2 or quadratic time).
- Cubic, polynomial or *exponential algorithms :* For algorithms that can take a time in the order of n^3, n^k or c^n.

O-Notation : An Asymptotic Upper Bound I

Definition (Big Oh 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.

O-Notation : An Asymptotic Upper Bound II

Example : Let $f(n) = 27n^2 + \frac{355}{113}n + 12$ be the number of basic operations 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,

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

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 $O(n^2)$.

O-Notation : An Asymptotic Upper Bound III

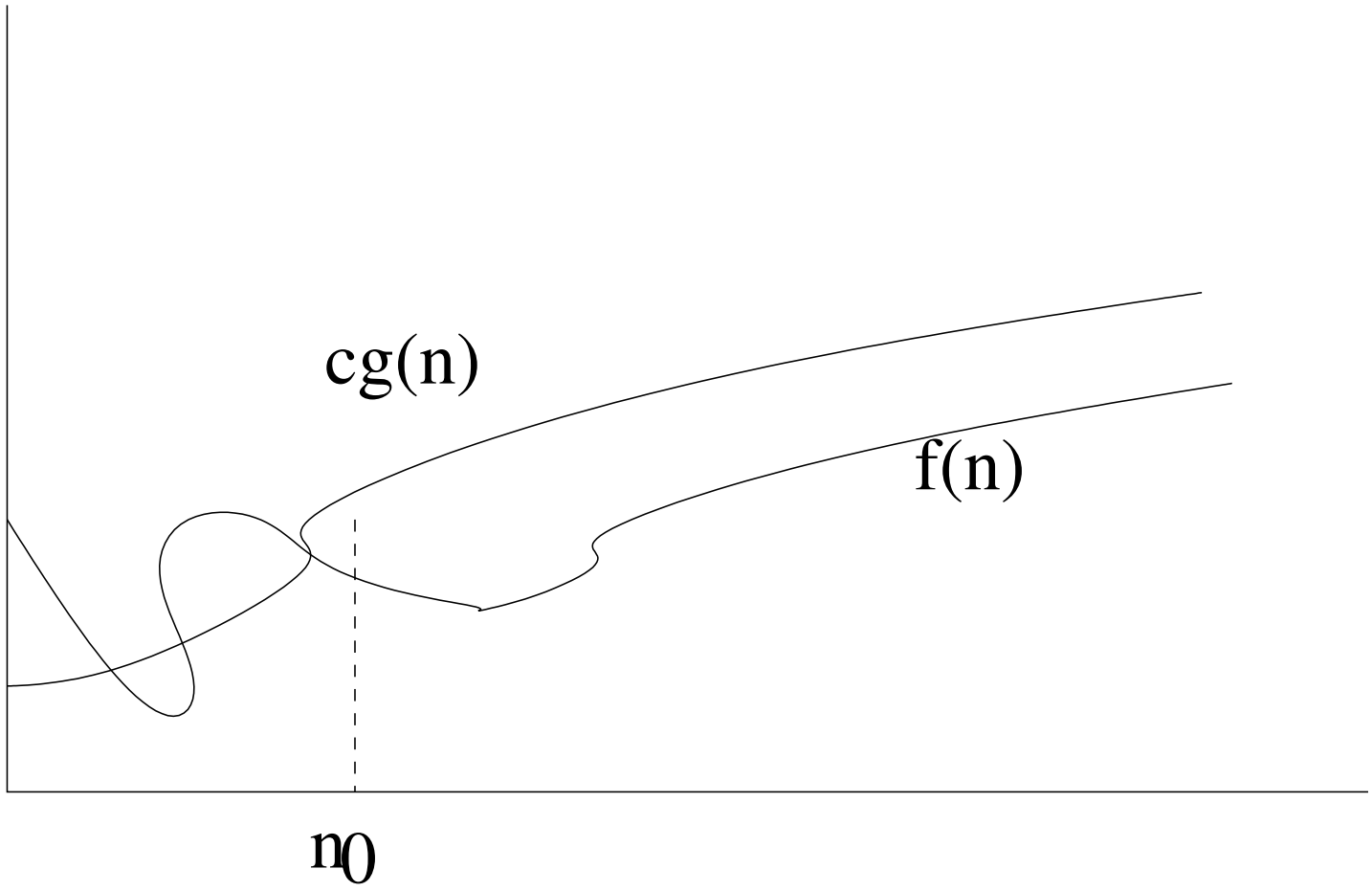
Terminologies : Let f and g be non-negative valued functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$:

1. We say that $f(n)$ is in the order of $g(n)$ if $f(n) \in O(g(n))$.
2. We say that "$f(n)$ is big-O of $g(n)$". For convenience, we also write $f(n) = O(g(n))$.
3. As n increases, $f(n)$ grows no faster than $g(n)$. In other words, $g(n)$ is an asymptotic upper bound of $f(n)$.

Graphic Example of O-notation

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

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



How to find $g(n)$

Given that we have a function f that gives the exact number of elementary operations performed by an algorithm in the worst case, we need to find :

1. the simplest and slowest-growing function g such that $f(n) \in O(g(n))$
2. prove the relation $f(n) \in O(g(n))$ is true, i.e. show $\exists\ c$ and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Strategy for finding $g(n)$

Assume $f(n) = 3n^2 + 2n$:

- 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 your $g(n)$.
- $f(n) = 3n^2 + 2n = O(n^2)$.

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. $\frac{1}{2}n^2 + 3n \in O(n^2)$.

Some properties for Big O notation

1. Reflexivity : $f(n) \in O(f(n))$.
2. Scalar rule : Let f be a non-negative valued functions defined on $\mathbb{N}$ and c be a positive constant. Then

$$O(cf(n)) \in O(f(n)).$$

   Example : $6n^2 \in O(n^2)$
3. Maximum rule : Let f, g be non-negative functions. Then

$$O(f(n) + g(n)) \in O(\max\{f(n), g(n)\}).$$

Exercises I

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

Exercise solutions I

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

1. Which instruction can be used as elementary operation ? Answer : $t := t + i + j$
2. Express the running time of this algorithm in terms of the number of times your selected elementary operation is executed ? Answer : n^2
3. Give (without proof) a big $\mathcal{O}$ estimate for the running time of the algorithm. $O(n^2)$

Exercise solutions I

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

What is computed and returned by this algorithm ?

The number of $i =$
$n + 2n + 3n + \cdots + n - 1(n) + n \times n = n(1 + 2 + 3 + \cdots + n - 1 + n) =$
$n(\sum_{i=1}^{n} i) = n \times \frac{n(n+1)}{2} = n \times \frac{n^2+n}{2} = \frac{n^3+n^2}{2}$

The number of $j = n(1 + 2 + 3 + \cdots + n - 1 + n) = \frac{n^3+n^2}{2}$

Answer : $t = 2(\frac{n^3+n^2}{2}) = n^3 + n^2$

Exercises I

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

     $1,000n^2 + 2^n + 36n \log n + \frac{3}{2}\left(\frac{3}{2}\right)^n$

Exercises II

2. 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 III

3. 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 IV

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

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

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

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

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

Exercises V

5. Find the appropriate "Big-Oh" relationship between the functions $n \log n$ and $5n$ and find the constants c and n_0
   Answer : $5n \in O(n \log n)$. Looking for c and n_0 such that $0 \leq 5n \leq cn \log n$

$$\begin{aligned} 5n &\leq cn \log n \\ &\leq c \log n \\ 5 &\leq \log 32 \end{aligned}$$

As $2^5 = 32$ and for $c = 1$, we have $5n \leq cn \log n$

Exercises VI

6. 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]
}
```

Answer : The first inner loop performs $2n^2$ operations while the second one performs n^3 operations. This iterative procedure performs $2n^2 + n^3$ operations. $2n^2 + n^3 \in O(n^3)$

Properties of O-Notation

These are properties we have been using informally. Now we state them explicitly and prove them.

Constant factors can be ignored in O-notation calculations.

Theorem 1 : For all $k > 0$, $kf(n)$ is $O(f(n))$.

Proof : For $n > 0$, $kf(n) \leq cf(n)$ for $c = k$.

Example : $f(n) = 3n^2$ implies that $3n^2 \in O(n^2)$.

Example : Any constant function is $O(1)$ since let $f(n) = C$ then $f(n) \leq C \cdot 1$ for all $n > N$.

Remark : 2 is not a constant factor in 2^n. Functions 2^n and 3^n have different growth rates, so don't throw away the 2 or 3 in these cases.

Theorem 2 : If $f(n)$ is $O(h(n))$ and $g(n)$ is $O(h(n))$, then $f(n) + g(n)$ is $O(h(n))$.

Proof : By definition,

- $f(n)$ **is** $O(h(n))$ means that for $n > n_1$, $f(n) \leq c_1 h(n)$ and
- $g(n)$ **is** $O(h(n))$ means that for $n > n_2$, $g(n) \leq c_2 h(n)$

where c_1, c_2, n_1, and n_2 are positive constants.

We want two positive constants c_3 and n_3 such that $f(n) + g(n) \leq c_3 h(n)$ for all $n > n_3$.

We compute c_3 by adding the two inequalities :

$$\begin{array}{rcl} f(n) & \leq & c_1 h(n) \\ g(n) & \leq & c_2 h(n) \\ \hline f(n)+g(n) & \leq & (c_1+c_2)h(n) \end{array}$$

So $c_3 = c_1 + c_2$, which is positive since c_1 and c_2 are.

What should n_3 be? We need $f(n) \leq c_1 h(n)$ and $g(n) \leq c_2 h(n)$ at the same time, so we can only consider values of n such that $n > n_1$ and $n > n_2$ at the same time. If $n > \max\{n_1, n_2\}$, then $n > n_1$ and $n > n_2$ at the same time, so choose $n_3 = \max\{n_1, n_2\}$, which is positive since n_1 and n_2 are.

Then, for $n > n_3$, we have $f(n) + g(n) \leq c_3 h(n)$, where $c_3 = c_1 + c_2$ and $n_3 = \max\{n_1, n_2\}$. Therefore, $f(n) + g(n)$ is $O(h(n))$.

Corollary to Theorem 2 : If $f(n)$ is $O(g(n))$, then $f(n) + g(n)$ is $O(g(n))$.

Proof : Since $g(n)$ is $O(g(n))$, we can put $h(n) = g(n)$ in Theorem 2 to get this result.

Example : $f(n) = n + n^2 \log_2 n - \log_b n$ is $O(n^2 \log_2 n)$.

If $h(n)$ grows faster than $g(n)$ and $g(n)$ grows faster than $f(n)$, then $h(n)$ grows faster than $f(n)$.

Theorem 3 : If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.

Proof : By the definition of O-notation, for $n > n_1$, $f(n) \leq c_1 g(n)$ and for $n > n_2$, $g(n) \leq c_2 h(n)$. Therefore, for $n > \max\{n_1, n_2\}$, $f(n) \leq c_3 h(n)$, where $c_3 = c_1 \cdot c_2$.

Example : If you calculate that $f(n)$ is $O($ something awful $)$ and read in a book that something awful is $O(n^2)$, then you can conclude that $f(n)$ is $O(n^2)$.

The product of upper bounds for functions gives an upper bound for the product of functions.

Theorem 4 : If $f(n)$ is $O(h(n))$ and $g(n)$ is $O(l(n))$, then $f(n) \cdot g(n)$ is $O(h(n) \cdot l(n))$.

Proof of Theorem 4 :

By definition, $f(n) \leq c_1 h(n)$ for all $n > n_1$ and $g(n) \leq c_2 l(n)$ for all $n > n_2$, for some positive constants c_1, c_2, n_1 and n_2.

Then for $n > n_3$

$$\begin{aligned} f(n) \cdot g(n) &\leq (c_1 h(n))(c_2 l(n)) \\ &= c_3 h(n) l(n) \end{aligned}$$

where $n_3 = \max\{n_1, n_2\}$ and $c_3 = c_1 \cdot c_2$.

Example : If you have a running time that is the product of two awful looking functions $f(n)$ and $g(n)$, it may be quite difficult to multiply them out, but you don't have to work so hard if you happen to know what O-notation these functions have.

Exponentials grow faster than powers.

Theorem 5 : n^k is $O(b^n)$, for all $b > 1$ and $k \geq 0$ and b^n is **NOT** $O(n^k)$.

Proof : Consider n^k/b^n where b and k are constants.

$$\lim_{n\to\infty} \frac{n^k}{b^n} = \lim_{n\to\infty} \frac{k!}{b^n(\log_e b)^k} = 0 < 1$$

by k applications of l'Hopital's rule.

This means that for $n > N =$ some integer, $n^k/b^n < 1$ or $n^k \leq cb^n$ for $c = 1$.

Logarithms grow more slowly than powers.

Theorem 6 : $\log_b n$ is $O(n^k)$ for any positive integers b and k.

Proof : By Theorem 5, for $n > N$, $n^k \leq b^n$.

Taking logs we get : for $n > N$, $k \log_b n \leq n$ or for $n > N$, $\log_b n \leq cn$, for $c = 1/k$.

Generalization : $(\log_b n)^r$ is $O(n^k)$ for any integer r and any positive integers k and b.

All logarithm functions grow at the same rate.

Theorem 7 : $\log_b n$ is $O(\log_c n)$

Proof : Property 5 of logs says $\log_b x = (\log_b c) \cdot (\log_c x)$.

The sum of the rth powers of the first n numbers grows at the same rate as the $(r+1)$st power of n.

Theorem 8 : $f(n) = \sum_{k=1}^{n} k^r$ is $O(n^{r+1})$.

Proof :

$$\begin{aligned}
\sum_{k=1}^{n} k^r &= 1^r + 2^r + \cdots + n^r \\
&\leq n^r + n^r + \cdots + n^r \\
&= n \cdot n^r \\
&= n^{r+1}
\end{aligned}$$

so for $n > 0$, $f(n) \leq cn^{r+1}$ where $c = 1$.

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

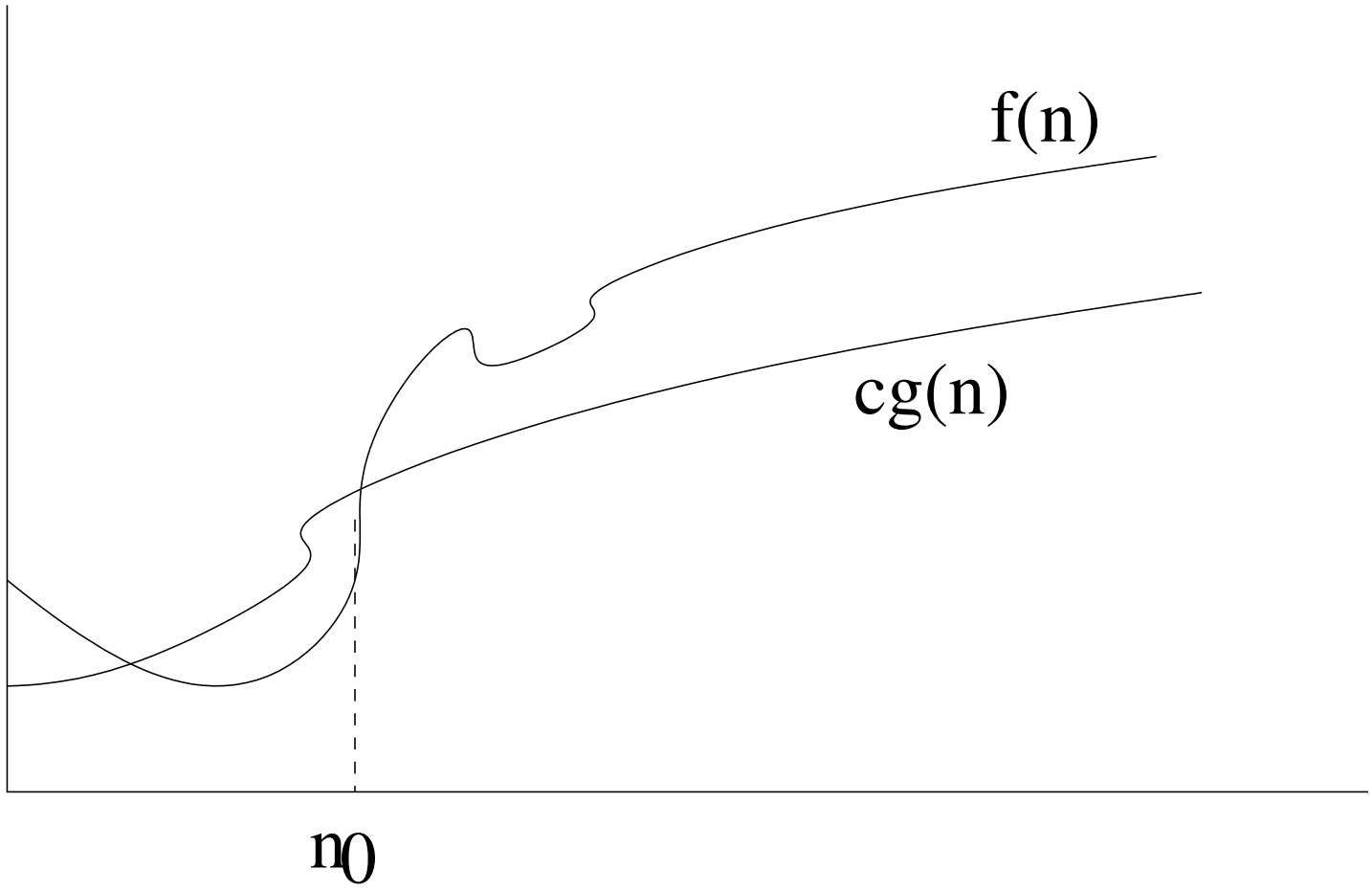
Let f and g be non-negative valued functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$:

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

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



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

$$cn^2 \leq n^3 - 4n^2$$
$$1 \leq n - 4$$

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

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

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

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

we will instead show that

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

Example 1 continue

$$c\,n\log n \le n\log n - 2\,n$$
$$c \le \frac{n\log n}{n\log n} - \frac{2n}{n\log n}$$
$$c \le 1 - \frac{2}{\log n}$$

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

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

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

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

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

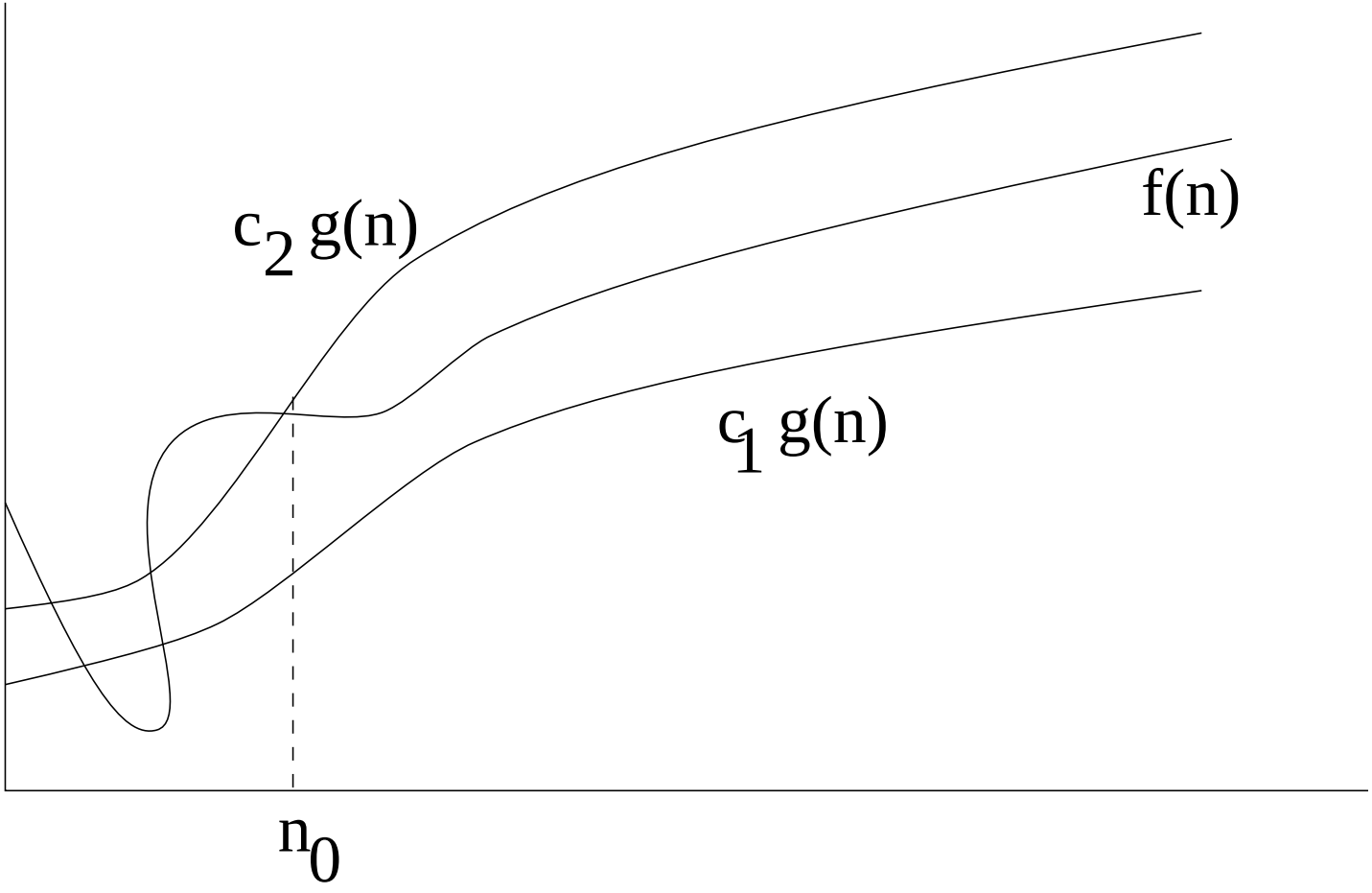
Let f and g be non-negative valued functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$:

1. We say that $f(n)$ is in the 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

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

- 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 $n \geq n_0$,

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

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* $f(n) \in \mathcal{O}(g(n))$ *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

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

Exercises with solutions

Prove that $f(n) = n^3 + 20n + 1 \in O(n^3)$

By the definition of big O, we must show $\exists$ some constants c and n_0 such $f(n) \leq cn^3$ for all values of $n \geq n_0$.

$$\begin{aligned} n^3 + 20n + 1 &\leq cn^3 \\ \frac{n^3}{n^3} + \frac{20n}{n^3} + \frac{1}{n^3} &\leq c \\ 1 + \frac{20}{n^2} + \frac{1}{n^3} &\leq c \end{aligned}$$

Therefore $f(n) \leq cn^3$ for $n_0 = 1$ and $c = 22$

We could have found this solution by simply adding the coefficients of all the terms in $f(n)$.

Larger values of n_0 will reduce the value of c. For example, if we select $n_0 = 5$,

$1 + \frac{20}{25} + \frac{1}{125} \leq c$ will work for $c > 1.808$

Exercises with solutions

Prove that $f(n) = n^3 + 20n + 1 \notin O(n^2)$

$$\begin{aligned} n^3 + 20n + 1 &\leq cn^2 \\ \frac{n^3}{n^2} + \frac{20n}{n^2} + \frac{1}{n^2} &\leq c \\ n + \frac{20}{n} + \frac{1}{n^2} &\leq c \end{aligned}$$

Here we cannot find a n_0 and a c such that $f(n) \leq cn^2$ for any value of $n \geq n_0$.

Exercises with solutions

Prove that $f(n) = n^3 + 20n + 1 \in O(n^4)$. By the definition of big O, we must show $\exists$ some constants c and n_0 such $f(n) \leq cn^4$ for all values of $n \geq n_0$.

$$\begin{aligned} n^3 + 20n + 1 &\leq cn^4 \\ \frac{n^3}{n^4} + \frac{20n}{n^4} + \frac{1}{n^4} &\leq c \\ \frac{1}{n} + \frac{20}{n^3} + \frac{1}{n^4} &\leq c \end{aligned}$$

This condition holds for $n_0 = 1$, $1 + 20 + 1 \leq c$ for $c \geq 22$

Exercises with solutions

Prove $f(n) = n^3 + 20n \in \Omega(n^2)$.

By the definition of big Omega, $f(n) \in \Omega(n^2)$ if $f(n) \geq cn^2$ for all $n \geq n_0$ and some constant c.

$$\begin{aligned} cn^2 &\leq n^3 + 20n \\ c &\leq n + \frac{20}{n} \end{aligned}$$

For $n_0 = 5$, the value of the right hand side of the equation is 9, if we set $c \leq 9$ then $f(n) \geq cn^2$

Since there is no negative term in $f(n)$, we could simply select $c < 3$, where 3 is the constant of the dominant term in $f(n)$. Setting $c = 2$, $f(n) \geq cn^2$ for all $n \geq n_0 = 1$.

Exercises with solutions

Prove $f(n) = \frac{1}{2}n^2 - 3n \in \Omega(n^2)$.

By the definition of big Omega, $f(n) \in \Omega(n^2)$ if $f(n) \geq cn^2$ for all $n \geq n_0$ and some constant c.

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

For $n \geq n_0 = 12$, the value of the right hand side of the equation is no smaller than $1/4$, therefore we can set $c = 1/4$. Then $\frac{1}{4}n^2 \leq \frac{1}{2}n^2 - 3n$ for all values of $n \geq 12$

Exercises with solutions

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

By the definition of Theta, $f(n) \in \Theta(n^2)$ if there exists c_1, c_2 and n_0 such that $c_1 n^2 \leq 5n^2 - 7n \leq c_2 n^2$ for all $n \geq n_0$.

$$\begin{aligned} c_1 n^2 &\leq 5n^2 - 7n \\ c_1 &\leq 5 - \frac{7}{n} \end{aligned}$$

which is true for $c_1 = 1$ and $n_0 = 2$

$$\begin{aligned} 5n^2 - 7n &\leq c_2 n^2 \\ 5 - \frac{7}{n} &\leq c_2 \end{aligned}$$

which is true for $c_2 = 5$ and $n_0 = 2$

Exercises with solutions

Prove that $f(n) = 23n^3 - 10n^2 \log n + 7n + 6 \in \Theta(n^3)$.

By the definition of Theta, $f(n) \in \Theta(n^3)$ if there exists c_1, c_2 and n_0 such that $c_1 n^3 \leq 23n^3 - 10n^2 \log n + 7n + 6 \leq c_2 n^3$ for all $n \geq n_0$.

We first prove that $f(n) \in \Omega(n^3)$

$$\begin{aligned} c_1 n^3 &\leq 23n^3 - 10n^2 \log n \\ c_1 &\leq 23 - \frac{10 \log n}{n} \end{aligned}$$

where $\frac{10 \log n}{n}$ decreases as n increases : $\frac{10 \log n}{n} = 5$ for $n = 2$, $\frac{10 \log n}{n} = 3.75$ for $n = 8$, $\frac{10 \log n}{n} = 2.5$ for $n = 16$. Therefore for $c_1 = 1$ and $n_0 = 2$, which is true for $c_1 = 1$ and $n_0 = 2$,
$c_1 n^3 \leq 23n^3 - 10n^2 \log n + 7n + 6$

Exercises with solutions

Prove that $f(n) = 23n^3 - 10n^2 \log n + 7n + 6 \in \Theta(n^3)$.

Next we prove that $f(n) \in O(n^3)$

$$\begin{aligned} 23n^3 + 7n + 6 &\leq c_2 n^3 \\ 23 + \frac{7}{n^2} + \frac{6}{n^3} &\leq c_2 \end{aligned}$$

where for $n_0 = 2$, $23 + \frac{7}{n^2} + \frac{6}{n^3} = 23 + 1.75 + 0.75 = 25.5$. There for $n_0 = 2$ and $c_2 = 26$, $23n^3 - 10n^2 \log n + 7n + 6 \leq 26n^3$

Exercises with solutions I

1. Find the appropriate Ω relationship between the functions n^3 and $3n^3 - 2n^2 + 2$ and find the constants c and n_0
   Answer : $3n^3 - 2n^2 + 2 \in \Omega(n^3)$. Looking for c and n_0 such that $0 \leq cn^3 \leq 3n^3 - 2n^2 + 2$. Choose $c = 1$ to be some positive constant less than the multiplicative constant of the fastest growing term of $3n^3 - 2n^2 + 2$.
2. 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 $(j = 0; j < n * n; j++)$
   $sum = sum + A[i] + A[j]$
   $\}$

   2.1 Give a polynomial expression describing the computing time of this procedure.
   Answer : $f(n) = n(2n + n^2) = 2n^2 + n^3$

Exercises with solutions II

2.2 Bound above the asymptotic time complexity of this code using the "Big-Oh" notation. Prove your result.
Answer : Dominant term is n^3. Defining the constant c for n^3 as the sum of the constants in the polynomial expression, then $2n^2 + n^3 \leq 3n^3$ for $c = 3$ and all value of $n \geq n_0 = 1$, therefore $f(n) \in O(n^3)$

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

Answer : For the lower bound, since the two terms in $f(n)$ are positive, choosing $c = 1$, we have $n^3 \leq 2n^2 + n^3$ for all $n \geq n_0 = 1$, therefore $f(n) \in \Omega(n^3) \Rightarrow f(n) \in \Theta(n^3)$, given that $f(n) \in O(n^3)$ as well

Exercises with solutions III

3. 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.

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

Answer : $1 < \log\log n < \log n < \sqrt{n} < n < n\log n < n^{3/2} < n^2 < n^3 < 2^n < 3^n < n! < n^n$

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

Answer : $n^{\log n} < n^{\sqrt{n}} < 2^n = 2^{n+1} < 2^{2n} < n! = n+1! < n^n$