Computer Science Department
College of Computer and Information Sciences
King Saud University.

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

3.1 Growth of functions and asymptotic notations

Big Oh:
$O(g(n))$ is the set of all functions with a smaller or same order of growth as $g(n)$.

$t(n) \in O(g(n)) \Leftrightarrow \exists c > 0, n_0 \geq 0 | t(n) \leq cg(n)$ for all $n \geq n_0$.

Big Omega:
$\Omega(g(n))$ is the set of all functions with a larger or same order of growth as $g(n)$.

$t(n) \in \Omega(g(n)) \Leftrightarrow \exists c > 0, n_0 \geq 0 | t(n) \geq cg(n)$ for all $n \geq n_0$.

Big Theta:
$\Theta(g(n))$ is the set of all functions with the same order of growth as $g(n)$.

$t(n) \in \Theta(g(n)) \Leftrightarrow \exists c_1, c_2 > 0, n_0 \geq 0 | c_2 g(n) \leq t(n) \leq c_1 g(n)$ for all $n \geq n_0$.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Useful theorems

- $\sum_{i=0}^{m} a_i n^i \in O(n^m)$.
- if $a_m > 0$, $\sum_{i=0}^{m} a_i n^i \in \Theta(n^m)$.
- $f(n) \in O(g_1(n))$ and $h(n) \in O(g_2(n)) \Rightarrow f(n) + h(n) \in O(MAX(g_1(n), g_2(n)))$.
- $f(n) \in O(g_1(n))$ and $h(n) \in O(g_2(n)) \Rightarrow f(n) \cdot h(n) \in O(g_1(n) \cdot g_2(n))$.

Eaxample:
Give a formal proof to show that $300n + 100 \in O(n)$.
$300n + 100 \leq 300n + n$ for all $n \geq 100$.
$300n + 100 \leq 301n$ for all $n \geq 100$.
Therefore, $n_0 = 100$ and $c = 301$.

Eaxample:
Give a formal proof to show that $3n^4 - 2n^3 + 20 \in \Omega(n^4)$.
$3n^4 - 2n^3 + 20 \geq 2n^4 + 20$ for all $n^4 \geq 2n^3$.
$3n^4 - 2n^3 + 20 \geq 2n^4 + 20$ for all $n \geq 2$.
$3n^4 - 2n^3 + 20 \geq 2n^4$ for all $n \geq 2$.

Therefore, $n_0 = 2$ and $c = 2$.

Eaxample:
Give a formal proof to show that $300n + 100 \in \Theta(n)$.
The right part:
$300n + 100 \leq 300n + n$ for all $n \geq 100$.
$300n + 100 \leq 301n$ for all $n \geq 100$.
Therefore, $n_{r0} = 100$ and $c_1 = 301$.
The left part:
$300n + 100 \geq 300n$ for all $n \geq 0$.

Therefore, $n_{l0} = 0$ and $c_2 = 300$.
Therefore, $c_1 = 301$, $c_2 = 300$, and $n_0 = MAX(n_{l0}, n_{r0}) = 100$.

Example: Prove that if $a_m > 0$, $\sum_{i=0}^{m} a_i n^i \in \Theta(n^m)$.

The right part:
$\sum_{i=0}^{m} a_i n^i \leq (\sum_{i=0, a_i>0}^{m} a_i) n^m$ for all $n \geq 1$.

Therefore, $n_{r0} = 1$ and $c_1 = \sum_{i=0, a_i>0}^{m} a_i$.

The left part:
$\sum_{i=0}^{m} a_i n^i \geq \frac{a_m}{2} n^m$, whenever $\frac{a_m}{2} n^m \geq -\sum_{i=0, a_i<0}^{m-1} a_i n^i$.
$\sum_{i=0}^{m} a_i n^i \geq \frac{a_m}{2} n^m$, whenever $\frac{a_m}{2} n^m \geq -\sum_{i=0, a_i<0}^{m-1} a_i n^{m-1}$.
$\sum_{i=0}^{m} a_i n^i \geq \frac{a_m}{2} n^m$, whenever $\frac{a_m}{2} n \geq -\sum_{i=0, a_i<0}^{m-1} a_i$.
$\sum_{i=0}^{m} a_i n^i \geq \frac{a_m}{2} n^m$, whenever $n \geq -\frac{2}{a_m} \sum_{i=0, a_i<0}^{m-1} a_i$.

Therefore, $c_1 = \sum_{i=0, a_i>0}^{m} a_i$, $c_2 = \frac{a_m}{2}$, and $n_0 = MAX(1, -\frac{2}{a_m} \sum_{i=0, a_i<0}^{m-1} a_i)$.

Useful theorems

- $f(n) \in O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) \in O(h(n))$.
- $f(n) \in \Omega(g(n))$ and $g(n) = \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$.
- $f(n) \in \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$.
- $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$.

3. Growth of functions and asymptotic notations

Basic asymptotic efficiency classes:

1 (constant).

$\log n$ (logarithmic).

n (linear).

$n \log n$ (n-log-n).

n^2 (quadratic).

n^3 (cubic).

2^n (exponential).

$n!$ (factorial).

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

3.2 Comparing the growth of functions

To compare two functions $f(n)$ and $g(n)$ we need to tell whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, or $f(n) \in \Theta(g(n))$.
The growth of $f(n)$ is the same as that of $g(n)$ if $f(n) \in \Theta(g(n))$.
The growth of $f(n)$ is lower than that of $g(n)$ if $f(n) \in O(g(n))$ and $g(n) \notin O(f(n))$

Eaxample:
Compare the growth of $f(n) = 10^6$ and $g(n) = \log n$.
The growth of $f(n)$ is lower than that of $g(n)$.
$f(n) \in O(g(n))$ and $g(n) \notin O(f(n))$
$10^6 \leq \log n$, whenever $n \geq 2^{(10^6)}$. $c = 1$ and $n_0 = 2^{(10^6)}$.

$\log n \notin O(10^6)$.
Proof:
It is impossible to find c and n_0 such that $\log n \leq c10^6$, whenever $n \geq n_0$.
Assume for the sake of contradiction that such c and n_0 exist.
$\log n \leq c10^6$, for any $n \geq n_0$. Therefore, $\log m \leq c10^6$, where $m = MAX(2^{c10^6}, n_0) + 1$.
But we know that $\log m > c10^6$ which makes a contradiction.
So it is not true that such c and n_0 exist.

Eaxample:
Compare the growth of $f(n) = 2^n$ and $g(n) = n^2$.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

3. Growth of functions and asymptotic notations

The growth of $f(n)$ is higher than that of $g(n)$.
$g(n) \in O(f(n))$ and $f(n) \notin O(g(n))$.
Can you prove this?
Note that $n^2 \leq 2^n$, whenever $n \geq 4$, which can be proven using mathematical induction.

Eaxample:
Compare the growth of $f(n) = n!$ and $g(n) = 3^n$.
The growth of $f(n)$ is higher than that of $g(n)$.
$g(n) \in O(f(n))$ and $f(n) \notin O(g(n))$.
$3^n \leq n!$, whenever $n \geq 3^3$. Why?

Basic asymptotic efficiency classes ordered based on their order of growth:
Note that c is a constant.

c (constant).

$\log n$ (logarithmic).

n (linear).

$n \log n$ (n-log-n).

n^2 (quadratic).

n^3 (cubic).

c^n (exponential).

$n!$ (factorial).

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

3.3 Using limits to compare orders of growth

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\begin{cases}0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n).\\ c>0 & \text{implies that } t(n) \text{ has the same order of growth than } g(n).\\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n).\end{cases}$$

L'Hopital's rule:

$$\lim_{n\to\infty}\frac{t(n)}{g(n)}=\lim_{n\to\infty}\frac{t'(n)}{g'(n)}$$

Stirling's formula:

$$n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \text{ for large values of } n.$$

Example:
Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

Example:
Compare the orders of growth of $\log_2 n$ and $\sqrt{n}$.

Example:
Compare the orders of growth of $n!$ and 2^n.
See the textbook for solutions to these examples.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Useful derivative rules

$f(x) = x^r \Rightarrow f'(x) = rx^{r-1}$.

$f(x) = a^x \Rightarrow f'(x) = (\ln a)a^x$.

$f(x) = \ln x \Rightarrow f'(x) = \dfrac{1}{x}$, $x > 0$.

$f(x) = \log_a x \Rightarrow f'(x) = \dfrac{1}{x \ln a}$.

$f(x) = h(g(x)) \Rightarrow f'(x) = h'(g(x)) \cdot g'(x)$.

$(f + g)' = f' + g'$.

$(f \cdot g)' = f' \cdot g + f \cdot g'$.

$\left(\dfrac{f}{g}\right) = \dfrac{f' \cdot g - f \cdot g'}{g^2}$, where $g = 0$.

$\log_a x = \dfrac{\log_k x}{\log_k a}$.

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

3.4 Algorithm analysis

1. Time efficiency: how fast is the algorithm?
2. Space efficiency: how much memory space does it need?

Time and space complexities are expressed as functions of the input size.
We focus more on time complexity.
What is our time unit?
It is the time it takes a computer to perform a simple operation (i.e., a simple operation is executed in one time unit). Simple operations are +, -, *, ÷, mod, div, comparing two atomic values, assignment, ...etc. The number of operations to be performed by an algorithm is usually expressed as a function of the input size n.
Operations in an algorithm can be divided into two types:

1. Minor operations are those that are performed a constant number of times regardless of the input size.
2. Major operations are those that are performed a number of times that depends on the input size.

We focus on major operations. This is because we try to understand the performance of the algorithm when the input size grows.
Major operations are usually the comparisons in loop conditions; let's call them major comparisons.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Algorithm 1: Searching for a given item in an array of items.

```
Algorithm Search(A[0..n − 1],K)
i:=0;
while i < n and A[i] ≠ K do
    i:=i+1;
end
if i < n then
    return i ;
else
    return −1 ;
end
```

Algorithm 2: An algorithm that sorts a list (array) of values.

```
Algorithm InsertionSort(A[0..n − 1])
for i:=1..n-1 do
    v:=A[i];
    j:=i − 1;
    while j ≥ 0 and A[j] > v do
        A[j + 1]:=A[j];
        j:=j-1;
    end
    A[j + 1]:=v;
end
```

Example:
An algorithm that searches for a given item in an array of items.

Worst-case time complexity analysis of the search algorithm:
Number of major comparisons= $n + 1 \in O(n)$.
Note that the major comparisons are those in the while loop condition.
Example:
An algorithm that sorts a list (array) of values.

Worst-case time complexity analysis of the InsertionSort algorithm:
Number of major comparisons=

$$n + 2 + 3 + 4 + ... + n = n + (\sum_{i=2}^{n} i) = n + \frac{n(n+1)}{2} - 1 \in O(n^2).$$

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

3.5 Time analysis of nonrecursive algorithms[Section 2.3]

How to analyze the time efficiency of an algorithm?
This can be done through two steps:

1. Assuming the worst case, count the number of major comparisons as a function f of the input size.
2. Find the order of growth of f using O and/or Θ.

Example:
Consider the problem of finding the largest value in a list.
Number of major comparisons= $n-1 \in \Theta(n)$.

Algorithm 1: An algorithm that finds the largest value in a list.

```
Algorithm MaxElement(A[0..n − 1])
maxval:=A[0];
for i:=1..n-1 do
    if A[i] > maxval then
        maxval:=A[i];
    end
end
return maxval;
```

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Example:
Consider an algorithm that finds the product of two n-by-n matrices A and B.
Note that it suffices to focus on the most inner loops in counting major comparisons. Why?

Algorithm 2: An algorithm that finds the product of two n-by-n matrices.

Algorithm MatrixMultiplication$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$
for *i:=0..n-1* **do**
  for *j:=0..n-1* **do**
    $C[i, j]$:=0;
    for *k:=0..n-1* **do**
      $C[i, j]$:=$C[i, j] + A[i, k] * B[k, j]$;
    end
  end
end
return C;

Number of major comparisons= $\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = n^3 \in \Theta(n)$.

Example:
Consider an algorithm that finds the number of binary digits in the binary representation of a positive decimal integer.
Number of major comparisons= $\lfloor \log_2 n \rfloor + 1 \in \log_2 n$.

Algorithm 3: Finding the number of binary digits in the binary representation of a positive decimal integer.

Algorithm Binary(n)
count:=1;
while $n > 1$ **do**
  count:=*count* + 1;
  n:=$\lfloor n/2 \rfloor$;
end
return *count*;

To be precise, the input size here is the number of digits representing n, and the algorithm is linear in the input size.