CHAPTER ~~2~~ 8:
Analysis of Algorithms

Java Software Structures:

Designing and Using Data Structures

Third Edition

John Lewis & Joseph Chase

Addison Wesley
is an imprint of
PEARSON

Chapter Objectives

- Discuss the goals of software development with respect to efficiency
- Introduce the concept of algorithm analysis
- Explore the concept of asymptotic complexity
- Compare various growth functions

Analysis of Algorithms

- An aspect of software quality is the efficient use of resources, including the CPU
- Algorithm analysis is a core computing topic
- It gives us a basis to compare the efficiency of algorithms
- Example: which sorting algorithm is more efficient?

Growth Functions

- Analysis is defined in general terms, based on:
  - the problem size (ex: number of items to sort)
  - key operation (ex: comparison of two values)
- A *growth function* shows the relationship between the size of the problem (n) and the time it takes to solve the problem

$$t(n) = 15n^2 + 45\ n$$

Growth Functions

Number of dishes (n)	$15n^2$	45n	$15n^2 + 45n$
1	15	45	60
2	60	90	150
5	375	225	600
10	1,500	450	1,950
100	150,000	4,500	154,500
1,000	15,000,000	45,000	15,045,000
10,000	1,500,000,000	450,000	1,500,450,000
100,000	150,000,000,000	4,500,000	150,004,500,000
1,000,000	15,000,000,000,000	45,000,000	15,000,045,000,000
10,000,000	1,500,000,000,000,000	450,000,000	1,500,000,450,000,000

FIGURE 2.1 Comparison of terms in growth function

Growth Functions

- It's not usually necessary to know the exact growth function
- The key issue is the *asymptotic complexity* of the function – how it grows as n increases
- Determined by the dominant term in the growth function
- This is referred to as the *order* of the algorithm
- We often use *Big-Oh notation* to specify the order, such as $O(n^2)$

Some growth functions and their asymptotic complexity

Growth Function	Order	Label
$t(n) = 17$	$O(1)$	constant
$t(n) = 3\log n$	$O(\log n)$	logarithmic
$t(n) = 20n - 4$	$O(n)$	linear
$t(n) = 12n \log n + 100n$	$O(n \log n)$	n log n
$t(n) = 3n^2 + 5n - 2$	$O(n^2)$	quadratic
$t(n) = 8n^3 + 3n^2$	$O(n^3)$	cubic
$t(n) = 2^n + 18n^2 + 3n$	$O(2^n)$	exponential

FIGURE 2.2 Some growth functions and their asymptotic complexity

Increase in problem size with a ten-fold increase in processor speed

Algorithm	Time Complexity	Max Problem Size Before Speedup	Max Problem Size After Speedup
A	n	s_1	$10s_1$
B	n^2	s_2	$3.16s_2$
C	n^3	s_3	$2.15s_3$
D	2^n	s_4	$s_4 + 3.3$

FIGURE 2.3 Increase in problem size with a tenfold increase in processor speed

Comparison of typical growth functions for small values of N

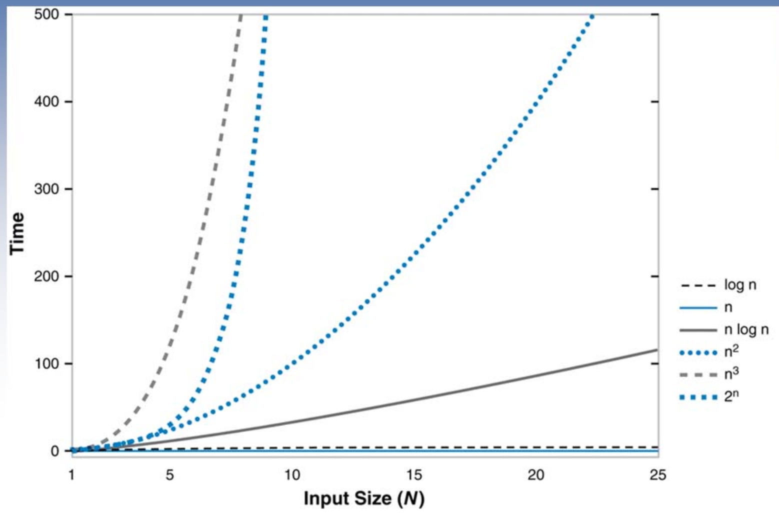

FIGURE 2.4 Comparison of typical growth functions for small values of n

Comparison of typical growth functions for large values of N

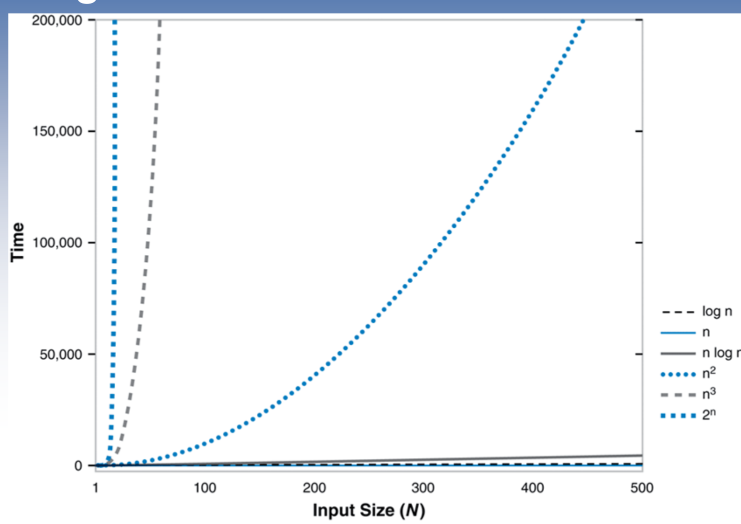

FIGURE 2.5 Comparison of typical growth functions for large values of n

Analyzing Loop Execution

- A loop executes a certain number of times (say n)
- Thus the complexity of a loop is n times the complexity of the body of the loop
- When loops are nested, the body of the outer loop includes the complexity of the inner loop

Analyzing Loop Execution

- The following loop is O(n) because the loop executes n times and the body of the loop is O(1):

```
for (int i=0; i<n; i++)
  {
     x = x + 1;
  }
```

Analyzing Loop Execution

- The following loop is $O(n^2)$ because the loop executes n times and the body of the loop, including a nested loop, is O(n):

```
for (int i=0; i<n; i++)
{
   x = x + 1;
   for (int j=0; j<n; j++)
   {
      y = y - 1;
   }
}
```


Analyzing Method Calls

- To analyze method calls, we simply replace the method call with the order of the body of the method
- A call to the following method is O(1)

```
public void printsum(int count)
{
     sum = count*(count+1)/2;
     System.out.println(sum);
}
```