Data Structures and Algorithms

Algorithm Analysis

Amilcar Aponte
amilcar@cct.ie

Today's Plan

- The concept of algorithm analysis
  - Big-Oh notation
- Theoretical analysis of algorithms
- Experimental analysis of algorithms in Java

Asymptotic Complexity

- Computational Complexity measures the degree of difficulty of an algorithm.
  - Not for the programmer, but for the computer to process it.
- Indicates how much effort is needed to apply an algorithm or how costly it is.
- To evaluate an algorithm's efficiency, use logical units that express a relationship such as:
  - The size n of a file or an array
  - The amount of time t required to process the data

Analysis of Algorithms

- Method 1:
  - Transform the algorithm into some programming language and measure the execution time for a set of input values.
- Method 2:
  - Use the Big O notation (mathematical notation) to analyse the behaviour of the algorithms. (Independent of software and hardware).

Complexity of Algorithms

- To understand the performance of an algorithm, it is best to observe how well or poorly it does for different size of inputs.
- This indicates that the selection of the algorithm for a particular problem demands analysis based on the input size.

Complexity of Algorithms

- Let us consider two algorithms to solve a problem:
  - First algorithm spends 100 milliseconds times “n”. Where “n” is the size of the input.
    - t = 100 ms * n
  - Second algorithm spends 5000 milliseconds regardless of the input size
    - t = 5000 ms

Complexity of Algorithms

- Which one is better?
  - Algorithm X is better if our problem size is small, that is, if $x < 50$
  - Algorithm Y is better for larger problems, with $x > 50$

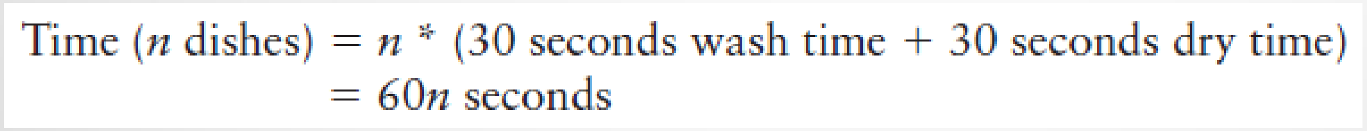
Running Time

- Most algorithms transform input objects into output objects.
- The running time of an algorithm typically grows with the input size.
- Average case time is often difficult to determine.
- We focus on the worst-case running time.
  - Easier to analyse.
  - Crucial to applications such as games, finance and robotics.

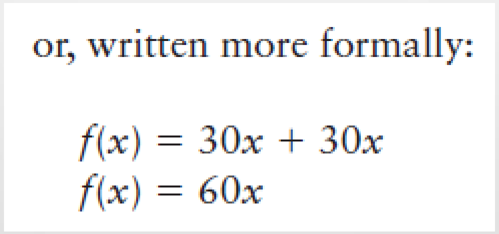
Algorithm Efficiency

- The efficiency of an algorithm is usually expressed in terms of its use of CPU time.
- The analysis of algorithms involves categorizing an algorithm in terms of efficiency.
- An everyday example: washing dishes.
  - Suppose washing a dish takes 30 seconds and drying a dish takes an additional 30 seconds.
  - Therefore, n dishes require n minutes to wash and dry.

Algorithm Efficiency



Time (n dishes) = n * (30 seconds wash time + 30 seconds dry time)
= $60n$ seconds



or, written more formally:

$$f(x) = 30x + 30x$$
$$f(x) = 60x$$

Algorithm Efficiency

- Now consider a less efficient approach
  - You must re-dry all previously washed dishes after washing another one

$$\text{time}\,(n\ \text{dishes}) = n * (30\ \text{seconds wash time}) + \sum_{\text{i}=1}^{\text{n}} (i * 30)$$

$$= 30n + \frac{30n(n+1)}{2}$$

$$= 15n^2 + 45n\ \text{seconds}$$

Problem Size

- For every algorithm we want to analyse, we need to define the size of the problem
- The dishwashing problem has a size *n* – number of dishes to be washed/dried
- For a search algorithm, the size of the problem is the size of the search pool
- For a sorting algorithm, the size of the program is the number of elements to be sorted

Growth Functions

- We must also decide what we are trying to efficiently optimize
  - time complexity – CPU time
  - space complexity – memory space
- CPU time is generally the focus
- A growth function shows the relationship between the size of the problem (n) and the value optimized (time)

Asymptotic Complexity

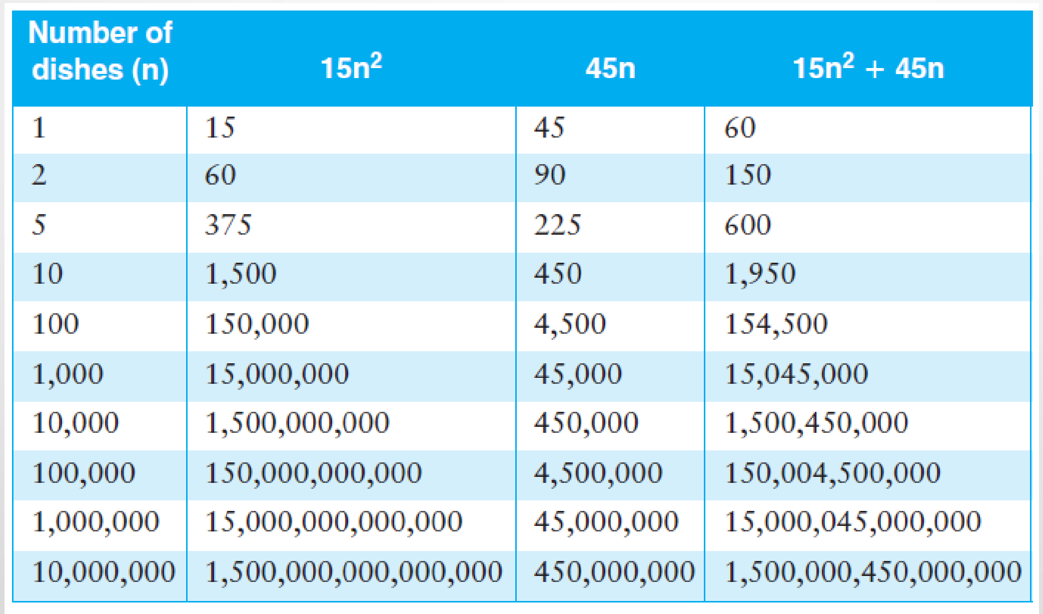
- The growth function of the second dishwashing algorithm is
  - $t(n) = 15n^2 + 45n$
- It is not typically necessary to know the exact growth function for an algorithm
- We are mainly interested in the asymptotic complexity of an algorithm – the general nature of the algorithm as n increases
- Asymptotic complexity helps to explore the performance of the algorithms

Asymptotic Complexity

- Asymptotic complexity is based on the dominant term of the growth function – the term that increases most quickly as n increases
- The dominant term for the second dishwashing algorithm is n^2.

Asymptotic Complexity

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

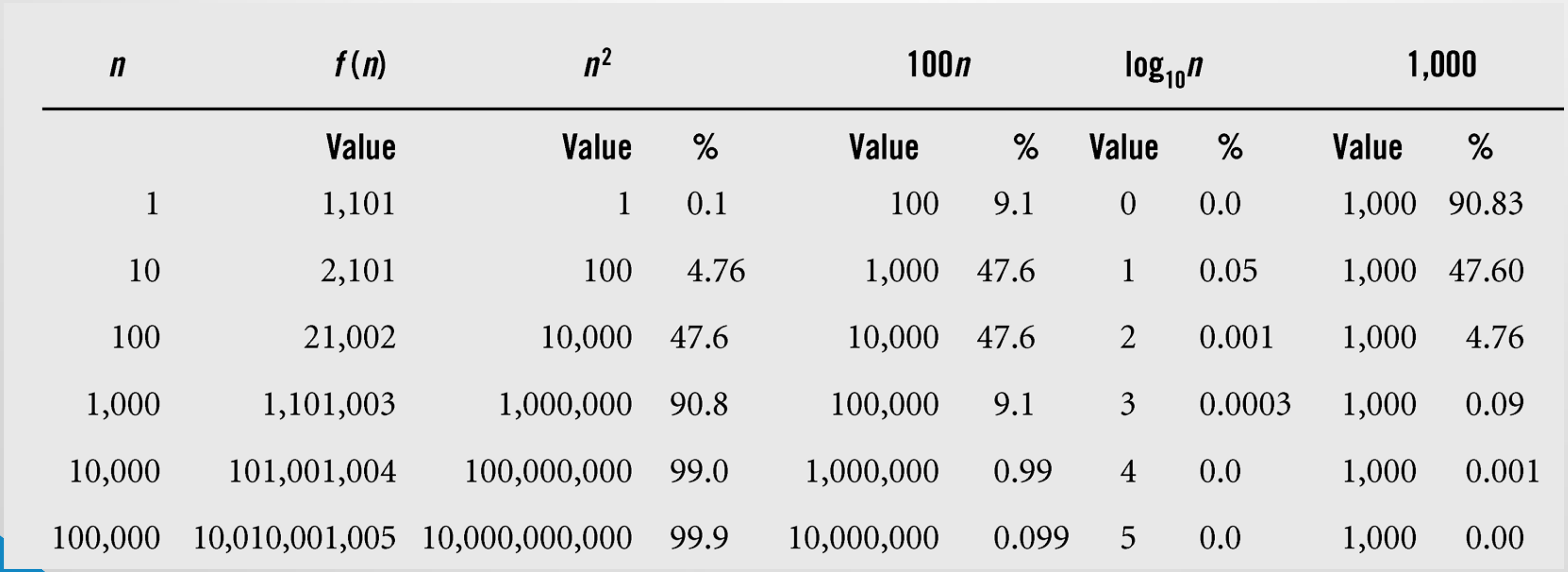


Number of dishes (n)	$15n^2$	$45n$	$15n^2 + 45n$
1	15	45	60
2	60	90	150
5	375	225	600
10	1,500	450	1,950
100	150,000	4,500	154,500
1,000	15,000,000	45,000	15,045,000
10,000	1,500,000,000	450,000	1,500,450,000
100,000	150,000,000,000	4,500,000	150,004,500,000
1,000,000	15,000,000,000,000	45,000,000	15,000,045,000,000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

FIGURE 12.1 Comparison of terms in growth function

Asymptotic Complexity

- $f(n) = n^2 + 100n + \log_{10}n + 1{,}000$



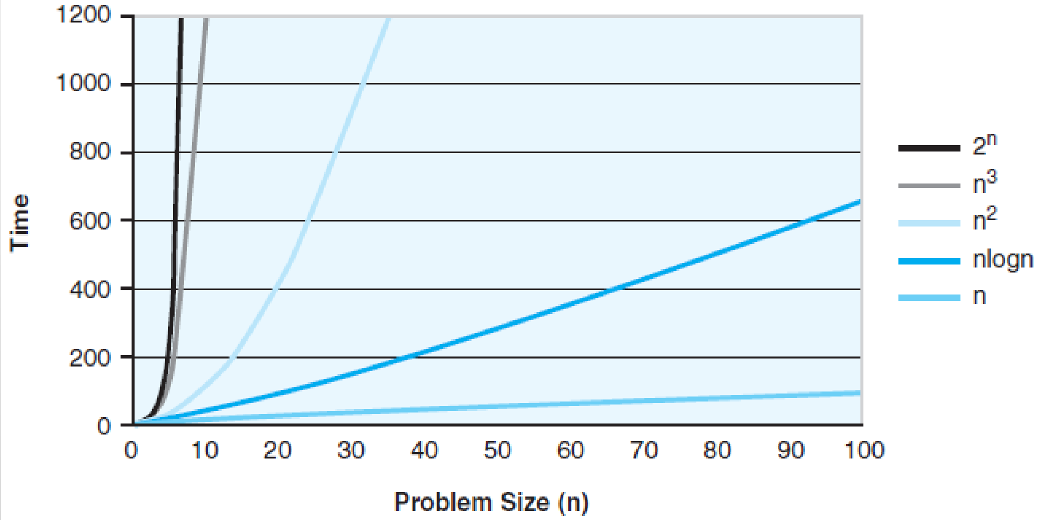
n	$f(n)$	n^2		$100n$		$\log_{10}n$		1,000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1,000	90.83
10	2,101	100	4.76	1,000	47.6	1	0.05	1,000	47.60
100	21,002	10,000	47.6	10,000	47.6	2	0.001	1,000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1,000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1,000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1,000	0.00

Big O Notation

- The coefficients and the lower order terms become increasingly less relevant as n increases
- So we say that the algorithm is order n, which is written $O(n^2)$
- This is called Big-Oh notation
- There are various Big-Oh categories
- Two algorithms in the same category are generally considered to have the same efficiency, but that doesn't mean they have equal growth functions or behave exactly the same for all values of n

Big O Categories

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



Comparison of typical growth functions for small values of n

Some examples

- I found this very good stackoverflow post
- https://stackoverflow.com/questions/2307283/what-does-olog-n-mean-exactly

- Also this post is kind of funny
- https://www.topcoder.com/blog/learning-understanding-big-o-notation/

Analysing Loop Execution

- First determine the order of the body of the loop, then multiply that by the number of times the loop will execute

```
for (int count = 0; count < n; count++)
    // some sequence of O(1) steps
```

- N loop executions times O(1) operations results in a O(n) efficiency

Analysing Loop Execution

- Consider the following loop:

```
count = 1;
while (count < n) {
    count *= 2;
    some sequence of O(1) steps
}
```

- The loop is executed $\log_2 n$ times, so the loop is O(log n)

Analising Loop Execution Nested Loops

- When the loops are nested, we multiply the complexity of the outer loop by the complexity of the inner loop

```
for (int count = 0; count < n; count++)
     for (int count2 = 0; count2 < n; count2++)
     {
          // some sequence of O(1) steps
     }
```

- Both the inner and outer loops have complexity of **O(n)**
- The overall efficiency is **O(n^2)**

Analysing Nested Methods

- The body of a loop may contain a call to a **method**
- To determine the order of the loop body, the order of the method must be taken into account

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, n
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/ software environment

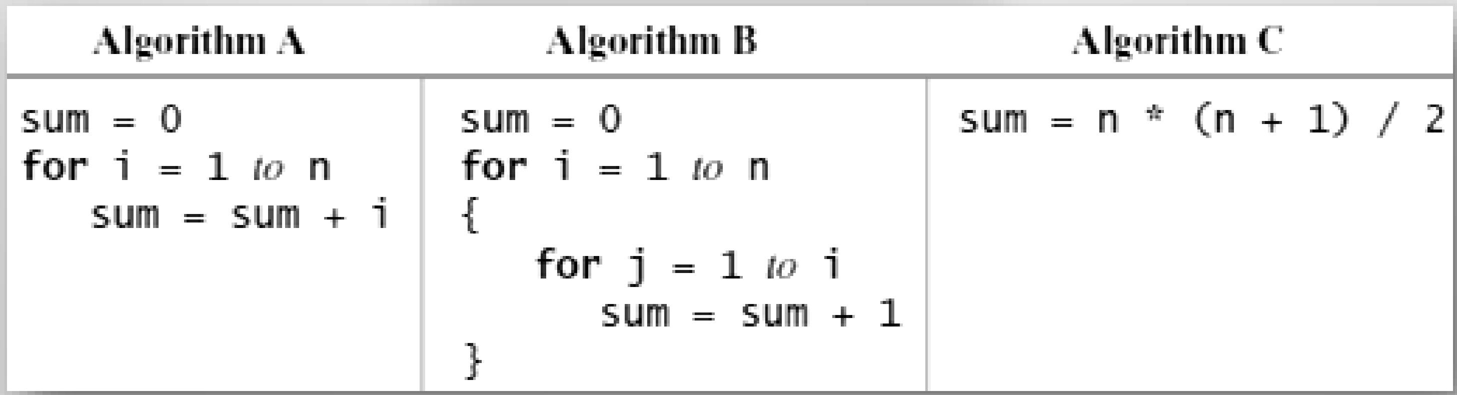
Experimental Studies

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition, noting the time needed:
- Plot the results

Complexity of Algorithms

- Consider the problem of summing up consecutive numbers

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \ldots + n$$



Algorithm A	Algorithm B	Algorithm C

`sum = 0`
`for i = 1 to n`
`    sum = sum + i`
	`sum = 0`
	`for i = 1 to n`
	`{`
	`    for j = 1 to i`
	`        sum = sum + 1`
	`}`
		`sum = n * (n + 1) / 2`

Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used

That’s all folks!

- Any questions?