Algorithms and Complexity

Introduction to complexity analysis

Paolo Camurati and Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Complexity Analysis

- Target
  - Forecast of resources (memory, time) needed by the algorithm for execution
- To really understand programs behavior we have to develop a mathematical model
  - Machine-independent
  - Assumption
    - Sequential single-processor model (traditional architecture)
  - Independent of the input data of a particular instance of the problem

Complexity Analysis

- It depends on the size n of the problem
- A lower complexity may compensate hardware efficiency
- Examples
  - Integer multiplication: Number of bits of the operands
  - Sorting algorithm: Number of data to sort
  - …
- Output
  - T(n): execution time
  - S(n): memory occupation

A Simple Counting Problem

- Write a program able to
  - Read an integer value N
  - Print-out the number M of ordered couples (i, j) such that
    - i and j are integer values
    - And $1 \leq i \leq j \leq N$
- Example
  - Input: N=4
  - Generated couples
    - (1,1)(1,2)(1,3)(1,4) (2,2)(2,3)(2,4) (3,3)(3,4) (4,4)
  - Output: M = 10

Algorithm 1

```
int count_ver1 (int N) {
  int i, j, sum;

  sum = 0;

  for (i=1; i<=N; i++) {

    for (j=i; j<=N; j++) {

      sum++;

    }
  }

  return sum;
}
```

It generates all pairs:
$1 \leq i \leq j \leq N$

It counts-them up

It returns the result

Algorithm 1

```
int count_ver1 (int N) {
  int i, j, sum;

  sum = 0;

  for (i=1; i<=N; i++) {

    for (j=i; j<=N; j++) {

      sum++;

    }
  }

  return sum;
}
```

1

$1 + (N + 1) + N$

$$\sum_{i=1}^{N} [1 + (N - i + 2) + (N - i + 1])$$

$$\sum_{i=1}^{N} (N - i + 1)$$

1

Algorithm 1

$$T(N) = 4 + 2N + \sum_{i=1}^{N}(5 + 3N - 3i)$$

$$T(N) = 4 + 2N + \underbrace{\sum_{i=1}^{N}(5)}_{5N} + \underbrace{\sum_{i=1}^{N}(3N)}_{3N^2} - \sum_{i=1}^{N}(3i)$$

1

$1 + (N + 1) + N$

$$\sum_{i=1}^{N}[1 + (N - i + 2) + (N - i + 1])$$

$$\sum_{i=1}^{N}(N - i + 1)$$

1

$$T(N) = 4 + 7N + 3N^2 - 3\sum_{i=1}^{N} i$$

Finite arithmetic progression

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$T(N) = 4 + 7N + 3N^2 - 3\frac{N(N+1)}{2}$$

$$\boldsymbol{T(N) = 1.5N^2 + 5.5N + 4}$$

Algorithm 2

```
int count_ver1 (int N) {
  int i, j, sum;
  sum = 0;
  for (i=1; i<=N; i++) {
    for (j=i; j<=N; j++) {
      sum++;
    }
  }
  return sum;
}
```

It generates all pairs:
$1 \leq i \leq j \leq N$

```
int count_ver2 (int N) {
  int i, sum;

  sum = 0;

  for (i=1; i<=N; i++) {
    sum = sum + (N-i+1);
  }

  return sum;
}
```

Algorithm 2

```
int count_ver2 (int N) {
  int i, sum;

  sum = 0;

  for (i=1; i<=N; i++) {
    sum = sum + (N-i+1);
  }

  return sum;
}
```

1

$1 + (N+1) + N$

$$\sum_{i=1}^{N} (4)$$

1

Algorithm 2

1

$1 + (N + 1) + N$

$$\sum_{i=1}^{N} (4) = 4N$$

1

$\boldsymbol{T(N) = 6N + 4}$

Algorithm 3

```
int count_ver2 (int N) {
  int i, sum;

  sum = 0;

  for (i=1; i<=N; i++) {
    sum = sum + (N-i+1);
  }

  return sum;
}
```

- The for cycle computes
  - $\sum_{i=1}^{N}(N-i+1) =$
    $= N^2 + N - \sum_{i=1}^{N} i =$
    $= \mathrm{N}(\mathrm{N}+1) - \frac{N(N+1)}{2}$
    $= \frac{N(N+1)}{2}$
  - which can be used to substitute the entire cycle

Algorithm 3

- The for cycle computes
  - $\sum_{i=1}^{N}(N - i + 1) = \frac{N(N+1)}{2}$
  - which can be used to substitute the entire cycle

```
int count_ver3 (int N) {
  return N * (N+1) / 2;
}
```

```
int count_ver2 (int N) {
  int i, sum;
  sum = 0;
  for (i=1; i<=N; i++) {
    sum = sum + (N-i+1);
  }
  return sum;
}
```

It generates all pairs:
$1 \le i \le j \le N$

Algorithm 3

- The for cycle computes
  - $\sum_{i=1}^{N}(N-i+1) = \frac{N(N+1)}{2}$
  - which can be used to substitute the entire cycle

```
int count_ver3 (int N) {
  return N * (N+1) / 2;
}
```

4

$\boldsymbol{T(N) = 4}$

Summary

Algorithm	T(N)	Order of T(N)
Version 1	$1.5N^2 + 5.5N + 4$	N^2
Version 2	$6N + 4$	N
Version 3	4	constant

Algorithm Classification

Asymptotic behavior	Algorithm Class
1	Constant
log n	Logarithmic
n	Linear
n log n	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential

Complexity grows much faster than the input size

Summary

- Hypothesis
  - 1 operation = 1 nsec = 10^{-9} sec

Wall-clock (elapsed) time

Asymptotic behavior	10^3	10^4	10^5	10^6	10^7
N	1µs	10µs	100µs	1ms	10ms
20 n	20µs	200µs	2ms	20ms	200ms
n log n	9.96µs	132µs	1.66ms	19.9ms	232ms
20 n log n	199µs	2.7ms	32ms	398ms	4.6sec
n^2	1ms	100ms	10s	17min	1.2day
20 n^2	20ms	2s	3.3min	5.6h	23day
N^3	1s	17min	12day	32years	32 millenium

Some more examples

- Discrete Fourier Transform
  - Decomposition of a N-sample waveform into periodic components
  - Applications: DVD, JPEG, astrophysics, ….
  - Trivial algorithm: Quadratic (N^2)
  - FFT (Fast Fourier Transform): Linearitmic (N log N)
- Simulation of N bodies
  - Simulates gravity interaction among N bodies
  - Trivial algorithm: Quadratic (N^2)
  - Barnes-Hut algorithm: Linearitmic (N log N)

Asymptotic Analysis

- Goal
  - Guess an upper-bound for T(n) for an algorithm on n data in the worst possible case
  - Asymptotic
    - For small n, complexity is irrelevant
    - Understand behaviour for $n \rightarrow \infty$

Asymptotic Analysis

- Three main analysis
  - Worst case
  - Average case
  - Best case
- Why worst-case analysis?
  - Conservative guess
    - Avoid complex hyphoteses on data
  - Worst case is very frequent
  - Average (and best) case
    - Either it coincides with the worst case
    - It is not definable, unless we resort to complex hypotheses on data

Tilde Notation

- Estimate running time (or memory) as a function of input size N
- Ignore lower order terms
  - When N is large, terms are negligible
  - When N is small, terms are not negligible but we do not care about them
- Definition
  - $f(N) \sim g(N)$ means $\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

Tilde Notation

- Examples
  - $\frac{1}{6}N^3 + 20N + 16 \sim \frac{1}{6}N^3$
  - $\frac{1}{6}N^3 + 100N^{4/3} + 16 \sim \frac{1}{6}N^3$
  - $\frac{1}{6}N^3 + \frac{5}{12}N^2 + 16 \sim \frac{1}{6}N^3$

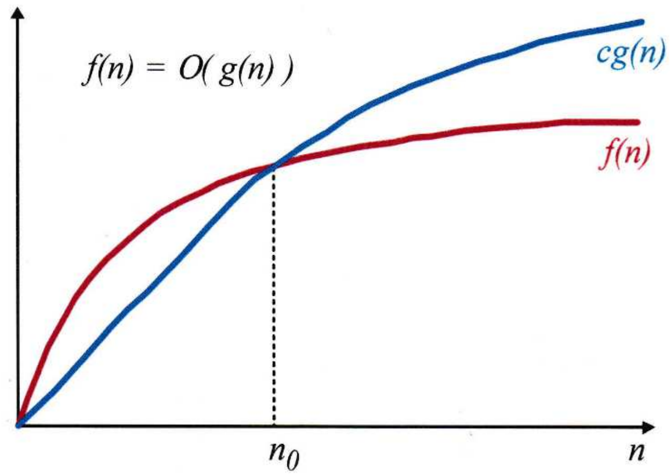
O Asymptotic Notation

- Definition

$$T(n) = O(g(n)) \iff \exists\, c>0,\ \exists\, n_0>0 \text{ such that } \forall n \geq n_0$$
$$0 \leq T(n) \leq cg(n)$$

g(n) = loose upper bound

Big-Oh Notation Develops upper bounds



O Asymptotic Notation

- Examples
  - $T(n) = 3n+2 = O(n)$
    - $c=4$ and $n_0=2$
  - $T(n) = 10n^2+4n+2 = O(n^2)$
    - $c=11$ and $n_0=5$
  - $T(n) = 3n+3 = O(n^2)$
    - $c=3$ and $n_0=2$
- Theorem
  - If $T(n) = a_m n^m + \dots + a_1 n + a_0$
    - Then $T(n) = O(n^m)$

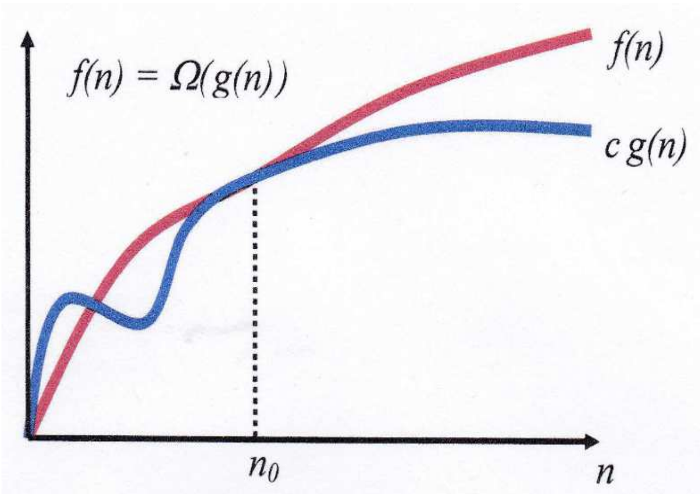
Ω Asymptotic Notation

- Definition

$$T(n) = \Omega(g(n)) \Leftrightarrow \exists\ c>0,\ \exists\ n_0>0 \text{ such that } \forall n \geq n_0$$
$$0 \leq c\ g(n) \leq T(n)$$

g(n) = loose lower bound for T(n)

Big-Omega Notation Develops lower bounds



Ω Asymptotic Notation

- Examples
  - $T(n) = 3n+3 = \Omega(n)$
    - $c=3$ and $n_0=1$
  - $T(n) = 10n^2+4n+2 = \Omega(n^2)$
    - $c=1$ and $n_0=1$

- Theorem
  - If $T(n) = a_m n^m + \ldots. + a_1 n + a_0$
    - Then $T(n) = \Omega(n^m)$

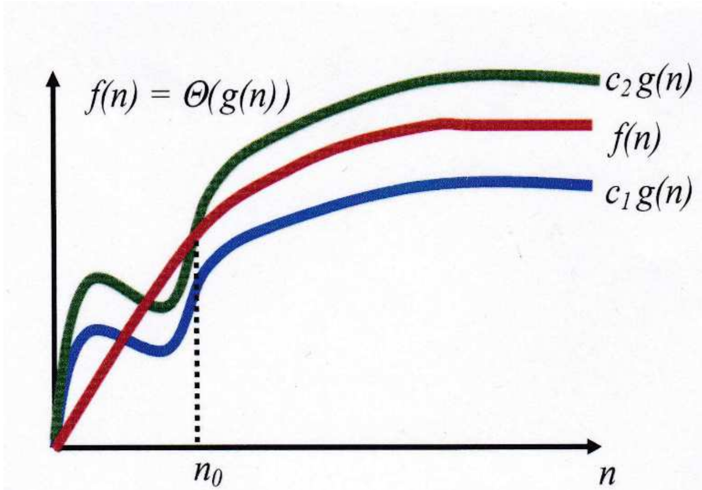
Θ Asymptotic Notation

- Definition

$$T(n) = \Theta(g(n)) \Leftrightarrow \exists\ c_1, c_2 > 0,\ \exists\ n_0 > 0 \text{ such that } \forall n \geq n_0$$
$$0 \leq c_1\, g(n) \leq T(n) \leq c_2\, g(n)$$

g(n) = tight asymptotic bound for T(n)

Big-Theta Notation
Classify algorithms
Asymptotic order of growth



Θ Asymptotic Notation

- Examples
  - $T(n) = 3n+2 = \Theta(n)$
    - c1=3, c2=4 and n0=2
  - $T(n) = 3n+2 \neq \Theta(n^2)$
  - $T(n) = 10n^2+4n+2 \neq \Theta(n)$

- Theorem
  - If $T(n) = a_m n^m + \ldots. + a_1 n + a_0$
    - Then $T(n) = \Theta(n^m)$

Theorems

- Given two functions f(n) and g(n)
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow$ f(n) = O(g(n))
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow$ f(n) = Ω(g(n))
  - $\lim_{n\to\infty} \frac{f(n)}{g(n)} = const \Rightarrow$ f(n) = Θ(g(n))
  - f(n) = Θ(g(n)) ⇔ g(n) = Θ(f(n))
  - f(n) = Θ(g(n)) ⇔
    f(n) = O(g(n)) and f(n) = Ω(g(n))
  - etc.

Memory Occupation

Basics Objects	Size
Bit	0 or 1
Byte	8 bits
1KByte	2^{10} Byt2 (1 thousand)
1MByte	2^{20} Bytes (1 million)
n1GByte	2^{30} Byte (1 billion)

C scalar Type	sizeof(type)
char	1 byte
int	4 Bytes
float	4 Bytes
double	8 Bytes
etc.	etc.

Padding may be used, i.e., each object uses a multiple of 4/8 bytes

Memory Occupation

- Memory occupation from aggregate types may be computed starting from scalar types

```
int vet[N];

struct type {
  char id[N];
  int i;
  float x;
};
```

N * sizeof (int)

N · sizeof (char) + sizeof (int) + sizeof (float) plus padding

- Total memory S(N) usage can be computed based on those considerations