Analysing Complexity of Algorithms

Basic Concepts and Definitions

Lecturer: Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

1. Definitions
2. Examples
   - One simple loop
   - Two nested loops
   - Two simple loops
3. Exercises

Informal Definition of an Algorithm

A list of unambiguous rules that specify successive steps to solve a problem.

More definitions:

- **Wikipedia:** "*. . . a step-by-step procedure for calculations / an effective method expressed as a finite list of well-defined instructions for calculating a function.*"
- **MathWorld:** "*. . . a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminates at some point.*"
- The abstract idea behind a computer program, i.e. the way of arranging the sequence of computational steps, so that the program works.

A computer program – a clearly specified sequence of computer instructions implementing the algorithm.

Examples of Algorithms

Sorting a database:

- Explicit and precise computational steps required to place all its entries in a certain order.

Searching for a certain entry in a database:

- Explicit and precise computational steps required to find whether a given entry is or is not in the database.

Finding the mean μ of n numbers $\{a_0, a_1, \ldots, a_{n-1}\}$:

- Summing all the numbers and dividing the sum by $n \longrightarrow$ $\mu = \frac{1}{n}(a_0 + a_1 + \ldots + a_{n-1})$.

Baking a cake(not a computational algorithm):

- Explicit and precise step-by-step instructions on how to bake the cake from given ingredients.

What is a Data Structure?

- **Wikipedia:** *"... a data structure is a particular way of storing and organising data in a computer so it can be used efficiently.*
- **Encyclopedia Britannica:** *"Way in which data are stored for efficient search and retrieval."*
- The simplest data structure – an one-dimensional array:

$$\{\text{element}_1, \text{element}_2, \ldots, \text{element}_n\}$$

- More complex data structures you might have already met:
  - Multi-dimensional arrays (each element is also an array).
  - Objects: a mix of different data with algorithms attached.
  - Arrays of objects.
  - Linked lists, stacks, and queues.
  - Trees.

What is a Data Structure?

- **Wikipedia:** *"... a data structure is a particular way of storing and organising data in a computer so it can be used efficiently.*
- **Encyclopedia Britannica:** *"Way in which data are stored for efficient search and retrieval."*
- The simplest data structure – an one-dimensional array:

$$\{\text{element}_1, \text{element}_2, \ldots, \text{element}_n\}$$

- More complex data structures you might have already met:
  - Multi-dimensional arrays (each element is also an array).
  - Objects: a mix of different data with algorithms attached.
  - Arrays of objects.
  - Linked lists, stacks, and queues.
  - Trees.

What is a Data Structure?

- **Wikipedia:** "*. . . a data structure is a particular way of storing and organising data in a computer so it can be used efficiently.*
- **Encyclopedia Britannica:** "*Way in which data are stored for efficient search and retrieval.*"
- The simplest data structure – an one-dimensional array:

$$\{\text{element}_1, \text{element}_2, \ldots, \text{element}_n\}$$

- More complex data structures you might have already met:
  - Multi-dimensional arrays (each element is also an array).
  - Objects: a mix of different data with algorithms attached.
  - Arrays of objects.
  - Linked lists, stacks, and queues.
  - Trees.

What is Not an Algorithm?

- A list of ingredients for a cake (no instructions how to bake it).
- An example of calculating the mean, $(5 + 13 + 6)/3 = 8$ (not a set of instructions).
- A data structure (e.g. a stack or a queue) by itself.
  - But the sets of instructions on how to `push` / `pop` or `queue` / `dequeue` / `insert` are algorithms.

It is not easy to define what is and what is not an algorithm...

> *"Just exactly what is and what is not an algorithm is in fact a fairly deep philosophical question. Intuitively, we are talking about a "recipe" that, given the input, will yield an answer in an automatic manner, following certain pre-set rules and procedures."*
>
> A.Magidin: `http://math.stackexchange.com/questions/21933/does-algorithmic-unsolvability-imply-unsolvability-in-general` [on-line: 14.02.2011]

What is Not an Algorithm?

- A list of ingredients for a cake (no instructions how to bake it).
- An example of calculating the mean, $(5 + 13 + 6)/3 = 8$ (not a set of instructions).
- A data structure (e.g. a stack or a queue) by itself.
  - But the sets of instructions on how to `push` / `pop` or `queue` / `dequeue` / `insert` are algorithms.

It is not easy to define what is and what is not an algorithm...

> *"Just exactly what is and what is not an algorithm is in fact a fairly deep philosophical question. Intuitively, we are talking about a "recipe" that, given the input, will yield an answer in an automatic manner, following certain pre-set rules and procedures."*
>
> A.Magidin: http://math.stackexchange.com/questions/21933/does-algorithmic-unsolvability-imply-unsolvability-in-general [on-line: 14.02.2011]

What is Not an Algorithm?

- ☺ A list of ingredients for a cake (no instructions how to bake it).
- An example of calculating the mean, $(5 + 13 + 6)/3 = 8$ (not a set of instructions).
- A data structure (e.g. a stack or a queue) by itself.
  - But the sets of instructions on how to `push` / `pop` or `queue` / `dequeue` / `insert` are algorithms.

It is not easy to define what is and what is not an algorithm...

> *"Just exactly what is and what is not an algorithm is in fact a fairly deep philosophical question. Intuitively, we are talking about a "recipe" that, given the input, will yield an answer in an automatic manner, following certain pre-set rules and procedures."*
>
> A.Magidin: `http://math.stackexchange.com/questions/21933/does-algorithmic-unsolvability-imply-unsolvability-in-general` [on-line: 14.02.2011]

Efficiency of Algorithms

How to compare algorithms / programs:

- By domain of definition – what inputs are legal?
- By correctness – does it solve the problem for all legal inputs? (**in fact, you need a formal proof!**)
- By *efficiency* – its **maximum** or **average** requirements to basic resources:
  - Computing time
  - Memory space
  - Other resources

Different implementations of the same algorithm: different programs, programming languages, computer platforms, operating systems...

> In searching for the best algorithm, general features of algorithms must be isolated from peculiarities of particular implementations.

Informal Definitions

An elementary operation is a computer instruction executed in a single time unit.

- Typically, a standard unary or binary arithmetic operation:
  - Negation (-5)
  - Addition / subtraction $(5 + 37;\ 350 - 75)$
  - Multiplication / division / modulo $(67 \times 89;\ 399/54;\ 399\%54)$
  - Boolean operations (x AND y; x OR y, etc.)
  - Binary comparisons ($x == y$; $x \leq y$; $x < y$; $x \geq y$; etc.)
  - Branching operations, etc.

The running time (or computing time) of an algorithm is the number of its elementary operations.

Informal Definitions

An elementary operation is a computer instruction executed in a single time unit.

- Typically, a standard unary or binary arithmetic operation:
  - Negation (-5)
  - Addition / subtraction $(5 + 37;\ 350 - 75)$
  - Multiplication / division / modulo $(67 \times 89;\ 399/54;\ 399\%54)$
  - Boolean operations $(x \text{ AND } y;\ x \text{ OR } y$, etc.)
  - Binary comparisons $(x == y;\ x \leq y;\ x < y;\ x \geq y$; etc.)
  - Branching operations, etc.

The running time (or computing time) of an algorithm is the number of its elementary operations.

Informal Definitions

An elementary operation is a computer instruction executed in a single time unit.

- Typically, a standard unary or binary arithmetic operation:
  - Negation (-5)
  - Addition / subtraction $(5 + 37;\ 350 - 75)$
  - Multiplication / division / modulo $(67 \times 89;\ 399/54;\ 399\%54)$
  - Boolean operations (x AND y; x OR y, etc.)
  - Binary comparisons ($x == y$; $x \leq y$; $x < y$; $x \geq y$; etc.)
  - Branching operations, etc.

The running time (or computing time) of an algorithm is the number of its elementary operations.

Example 1: $s = \sum_{i=0}^{n-1} a[i]$ – Linear Time Complexity

Algorithm 1: Summing n elements of a linear array $a[0..n-1]$.

Input: array $a[0..n-1]$; **Output**: sum s
begin
  $s \leftarrow 0$
  for $i \leftarrow 0$ **step** $i \leftarrow i+1$ **until** $n-1$ **do**
    $s \leftarrow s + a[i]$
  end for
  return s
end

Summing n elements of the array a repeats elementary fetch/add operations n times.

Therefore, running time is linear in n, i.e., $T(n) = cn$.

Example 2: Sums of Subarrays, or a "Moving Window"

For an array $\mathbf{a} = \{a[i] : i = 0, 1, \ldots, n-1\}$ of size n, compute $n - m + 1$ sums of the m successive elements:

$$s[j] = \sum_{k=0}^{m-1} a[j+k]; \quad j = 0, \ldots, n-m$$

at each of the possible $n - m + 1$ positions of the window supporting each current subarray.

Brute force computation:

- $\Rightarrow$ cm elementary operations per subarray
- $\Rightarrow$ $n - m + 1$ subarrays
- $\Rightarrow$ In total: $cm(n - m + 1)$ operations

Time is **linear** in n if m is fixed and is **quadratic** if m is growing with n (e.g. if $m = 0.5\ n$).

Quadratic Time (Two Nested Loops; $n = 2m$)

Algorithm 2 (`slowsum`):
Getting $m + 1$ sums of subarrays $a[j..(m + j - 1)]$; $j = 0, \ldots, m$.

```
Input: array a[0..2m − 1]; Output: array of sums s[0..m]
begin
    for j ← 0 step j ← j + 1 until m do
        s[j] ← 0
        for i ← 0 step i ← i + 1 until m − 1 do
            s[j] ← s[j] + a[j + i]
        end for
    end for
    return s
end
```

$$T(2m) = cm(m + 1) \Rightarrow T(n) = c\frac{n}{2}\left(\frac{n}{2} + 1\right) \approx c'n^2 = n^2T(1)$$

Getting Linear Computing Time

Quadratic time due to reiterated innermost computations:

$$s[j] = a[j] + \underline{a[j+1] + \ldots + a[j+m-1]}$$

$$s[j+1] = \underline{a[j+1] + \ldots + a[j+m-1]} + a[j+m]$$

Linear time $T(n) = c(m+2m) = 1.5cn$ after excluding reiterated computations:

$$s[0] = a[0] + a[1] + \ldots + a[m-1]$$

$$s[1] = s[0] - a[0] + a[m]$$

$$s[j+1] = s[j] - a[j] + a[m+j]; \quad j = 1, \ldots, m-1$$

Linear Time (Two Simple Loops)

Algorithm 3 (`fastsum`):
Getting $m+1$ sums of subarrays $a[j..(m+j-1)]$; $j=0,\ldots,m$.

Input: array $a[0..2m-1]$; **Output**: array of sums $s[0..m]$

begin
  $s[0] \leftarrow 0$
  for $i \leftarrow 0$ **step** $i \leftarrow i+1$ **until** $m-1$ **do**
    $s[0] \leftarrow s[0] + a[i]$
  end for
  for $j \leftarrow 0$ **step** $j \leftarrow j+1$ **until** $m-1$ **do**
    $s[j+1] \leftarrow s[j] + a[m+j] - a[j]$
  end for
  return s
end

Linear Vs. Quadratic Complexity

Relative growth of linear and quadratic terms in the expression $T(n) = c\frac{n}{2}\left(\frac{n}{2}+1\right) = c\left(0.25n^2 + 0.5n\right)$:

n	$T(n)$	$0.25n^2$	$0.5n$	
			value	% of quadratic term
10	30	25	5	20.0
100	2550	2500	50	2.0
1000	250500	250000	500	0.2
5000	6252500	6250000	2500	0.04

Computing time for $T(1) = 1\ \mu sec$:

Size of array	n	$2,000$	$2,000,000$
Size of subarray	m	$1,000$	$1,000,000$
Number of subarrays	$m+1$	$1,001$	$1,000,001$
Brute-force (**quadratic**) algorithm	$T(n)$	$2\ sec$	$> 23\ days$
Efficient (**linear**) algorithm	$T(n)$	$1.5\ msec$	$1.5\ sec$

Exercise 1.1.1*

A quadratic algorithm with processing time $T(n) = cn^2$ uses 500 elementary operations for processing 10 data items. How many will it use for processing 1000 data items?

Solution:

$$\begin{array}{rlclcl} & T(10) & = & c \cdot 10^2 & = & 500, \text{ that is,} \\ \Rightarrow & c & = & 500/100 & = & 5 \\ \Rightarrow & T(1000) & = & 5 \cdot 1000^2 & = & 5 \cdot 10^6 \end{array}$$

that is, 5 million operations to process 1000 data items.

In fact, we need not compute c, because $\frac{T(1000)}{T(10)} = \frac{c10^6}{c10^2} = 10^4$, so that $T(1000) = 10^4 T(10) = 10^4 \cdot 500$, or 5 million.

* M.J.Dinneen, G. Gimel'farb, M. C. Wilson: Introduction to Algorithms and Data Structures. 4^{th} ed. (e-book), 2016, p.18/210.

Exercise 1.1.2*

Algorithms $\mathbf{A}$ and $\mathbf{B}$ use $T_{\mathbf{A}}(n) = c_{\mathbf{A}} n \log_2 n$ and $T_{\mathbf{B}}(n) = c_{\mathbf{B}} n^2$ elementary operations for a problem of size n. Find the fastest algorithm for processing $n = 2^{20}$ data items if $\mathbf{A}$ and $\mathbf{B}$ spend 10 and 1 operations, respectively, to process 2^{10} items.

Solution:

$$\begin{array}{lcccccccc} T_{\mathbf{A}}\left(2^{10}\right) & = & 10 & \Rightarrow & c_{\mathbf{A}} & = & \frac{10}{10 \cdot 2^{10}} & = & 2^{-10} \\ T_{\mathbf{B}}\left(2^{10}\right) & = & 1 & \Rightarrow & c_{\mathbf{B}} & = & \frac{1}{2^{20}} & = & 2^{-20} \end{array}$$

$$T_{\mathbf{A}}(2^{20}) = 2^{-10} \cdot 20 \cdot 2^{20} = 20 \cdot 2^{10} << T_{\mathbf{B}}(2^{20}) = 2^{-20} \cdot 2^{40} = 2^{20}$$

Therefore, algorithm $\mathbf{A}$ is the fastest for $n = 2^{20}$.

* M.J.Dinneen, G. Gimel'farb, M. C. Wilson: Introduction to Algorithms and Data Structures. 4^{th} ed. (e-book), 2016, p.19/210.