Some Informal Definitions

- **algorithm** - a system of uniquely determined rules that specify successive steps in solving a problem
- **program** - a clearly specified series of computer instructions implementing the algorithm
- **elementary operation** - a computer instruction executed in a single time unit (computing step)
- **running** (computing) **time** of an algorithm - a number of its computing steps (elementary operations)

Efficiency of Algorithms: How to compare algorithms / programs

- by **domain of definition** – what inputs are legal?
- by **correctness** – is output correct for each legal input? (in fact, you need a **formal proof**!)
- by **basic resources** – ***maximum*** or ***average*** requirements:
  - **computing time**
  - **memory space**

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

Algorithm linear sum (**input**: array $a[n]$)

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

To sum elements of an array $a[n]$, elementary fetch–add operations are repeated n times $\Rightarrow$

Running time $T(n) = cn$ is **linear** in n

Example 2: GCD

- The **greatest common divisor**, $k = \mathrm{GCD}(n, m)$ is the greatest positive integer such that it divides both two positive integers m and n
- A ***"brute-force"*** **linear** solution: to exhaust all integers from 1 to the minimum of m and n
- Is it practicable to use such an algorithm to find **GCD(3 787 776 332, 3 555 684 776)** or even **GCD(9245,7515)?**

Euclid's GCD Algorithm

- ***<u>Euclid's analysis</u>***: if k divides both m and n, then it divides their difference ($n - m$ if $n > m$):

$$\mathbf{GCD}(n, m) = \mathbf{GCD}(n-m, m)$$

- k divides every difference when the subtraction is repeated up to λ times until $n - \lambda m < m$:

$$\mathbf{GCD}(n, m) = \mathbf{GCD}(\, n \bmod m, m\,)$$

where n **mod m**, or n **modulo** m is the ***remainder*** of division of n by m (in Java/C: $n\%m$, e.g. 13%5 = 3)

Euclid's GCD $\approx c \log(n+m)$ time

GCD(9245,7515) = 5

9245 **mod** 7515 = 1730	7515 **mod** 1730 = 595
1730 **mod** 595 = 540	595 **mod** 540 = 55
540 **mod** 55 = 45	55 **mod** 45 = 10
45 **mod** 10 = 5	10 **mod** 5 = **0** ⇒**GCD=5**

8 steps vs **7515** steps of the brute-force algorithm!

Example 3: Sums of Subarrays

Given an array $(a[i]: i = 0,1, \ldots, n - 1)$ of size n, compute $n - m + 1$ sums:

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

of all contiguous subarrays of size m

- **Brute force computation**: cm operations per subarray; in total: $cm(n - m + 1)$ operations
- Time is **linear** if m is fixed and **quadratic** if m is growing with n, such as $m = 0.5n$

Quadratic time (2 *nested loops*)

```
Algorithm slowsum (input: array a[2m])
  begin array s[m + 1]
     for j ← 0 to m do
        s[j] ← 0
        for k ← 0 to m−1 do
           s[j] ← s[j] + a[k + j]
        end for
     end for
     return s
  end
```

$$T(n) = c\frac{n}{2}\left(\frac{n}{2}+1\right) \cong c'\cdot n^2 = n^2T(1)$$

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:

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

Linear time (2 *simple loops*)

Algorithm fastsum (**input**: array $a[2m]$)
  begin array $s[m + 1]$
    $s[0] \leftarrow 0$
    for $k \leftarrow 0$ **to** $m-1$ **do** $s[0] \leftarrow s[0] + a[k]$
    end for
    for $j \leftarrow 1$ **to** m **do**
      $s[j] \leftarrow s[j-1] + a[j + m - 1] - a[j - 1]$
    end for
    return s
  end

Computing Time for T(1)=1μs

Array size	n	2,000	2,000,000
Size / number of subarrays	m / $m + 1$	1,000 / 1,001	1,000,000 / 1,000,001
Brute-force (***quadratic***) algorithm	$T(n)$	2 *sec*	> 23 *days*
Efficient (***linear***) algorithm	$T(n)$	1.5 *msec*	1.5 *sec*

Exercises: Textbook, p.12

1.1.1: Quadratic algorithm with processing time $T(n)=cn^2$ spends 500μ *sec* on 10 data items. What time will be spent on 1000 data items?

Solution: $T(10) = c\cdot 10^2 = 500 \rightarrow c = 500/100 = 5$ $\mu sec/item$
$\rightarrow T(1000) = 5\cdot 1000^2 = 5\cdot 10^6$ μsec or $T(1000) = 5$ *sec*

1.1.2: Algorithms **A** and **B** use $T_A(n) = c_A n \log_2 n$ and $T_B(n) = c_B n^2$ elementary operations for a problem of size n. Find the fastest algorithm for processing $n = 2^{20}$ data items if **A** and **B** spend 10 and 1 operations, respectively, to process $2^{10}=1024$ items.

Solution: $T_A(2^{10}) = 10 \rightarrow \boldsymbol{c_A} = 10/(10\cdot 2^{10}) = \mathbf{2^{-10}}$;
$T_B(2^{10}) = 1 \rightarrow \boldsymbol{c_B} = 1/2^{20} = \mathbf{2^{-20}}$
$\rightarrow \boldsymbol{T_A(2^{20})} = 2^{-10}\cdot 20 \cdot 2^{20} = \mathbf{20\cdot 2^{10}} \ll \boldsymbol{T_B(2^{20})} = 2^{-20} \cdot 2^{40} = \mathbf{2^{20}}$
$\rightarrow$ Algorithm **A** is the fastest for $n = 2^{20}$