5.2 Concepts of efficiency

So far the efficiency of algorithms has been studied from the point of view of the running-time. However, there are other alternatives:

- We can measure the memory consumption or bandwidth usage

In addition, in practise at least the following need to be taken into account:

- The unit used to measure the consumption of resources
- The definition of the size of the input
- Whether we're measuring the best-case, worst-case or average-case resource consumption
- What kind of input sizes come in question
- Are the asymptotic notations precise enough or do we need more detailed information

Units of the running-time

When choosing the unit of the running-time, the "step", usually a solution as independent of the machine architecture as possible is aimed for.

- Real units like a second cannot be used
- The constant coefficients become unimportant
  $\Rightarrow$ We're left with the precision of the asymptotic notations.

$\Rightarrow$ any constant time operation can be used as the step

- Any operation whose time consumption is independent of the size of the input can be seen as constant time.
- This means that the execution time of the operation never exceeds a given time, independent of the size of the input
- Assignment of a single variable, testing the condition of a **if**-statement etc. are all examples of a single step.
- There is no need to be too precise with defining the step as $\Theta(1) + \Theta(1) = \Theta(1)$.

Units of memory use

A bit, a byte (8bits) and a word (if its length is known) are almost always exact units

The memory use of different types is often known, although it varies a little between different computers and languages.

- an integer is usually 16 or 32 bits
- a character is usually 1 byte = 8 bits
- a pointer is usually 4 bytes = 32 bits
- an array $A[1 \ldots n]$ is often $n \cdot$ <element size>

$\Rightarrow$ Estimating the exact memory use is often possible, although requires precision.

Asymptotic notations are usefull when calculating the exact number of bytes is not worth the effort.

If the algorithm stores the entire input simultaneously, it makes sense to separate the amount of memory used for the input from the rest of the memory consumption.

- $\Theta(1)$ extra memory vs. $\Theta(n)$ extra memory
- It's worth pointing out that searching for a character string from an input file doesn't store the entire input but scans through it.

5.3 Binary Search

An efficient search in sorted data

Works by comparing a search key to the middle element in the data

- divide the search area into two
- choose the half where the key must be, ignore the other
- continue until only one element left
  - it is the key
  - the element is not in the data set

```
BIN-SEARCH(A, 1, n, key)        ▷ requirement: n ≥ 1, the array is sorted
1   low := 1; hi := n           (initialize the search to cover the entire array)
2   while low < hi do           (search until there are no more elements to cover)
3       mid := ⌊(low + hi)/2⌋   (divide the search area in half)
4       if key ≤ A[mid] then    (If the key is in the bottom half...)
5           hi := mid           (...choose the bottom half as the new search area)
6       else                    (else...)
7           low := mid + 1      (...choose the upper half as the search area)
8   if A[low] = key then
9       return low              (the element is found)
10  else
11      return 0                (the element is not found)
```

- The running time of BIN-SEARCH is $\Theta(\lg n)$.