D. Algorithms

V. Blobel
Universität Hamburg

April 2001

Robert Sedgewick, Algorithms, Addison-Wesley (various editions)
Sara Base, Computer Algorithms, Addison-Wesley
William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes in . . . (various progr. lang.), Cambridge University Press

1. Analysis of algorithms

Complex problems are divided into smaller problems, which hopefully can be solved by standard algorithms.

Important standard problems in data organisation

- sorting of items according to a key (ascending, descending)
- search in a list

Algorithms usually apply to a set of objects of a certain size n. The dependence $f(n)$ of the computer time or the number of operations on n for larger values of n (leading dependence) is often an essential property of an algorithm.

Sometimes important:

$A(n) =$ average behaviour $\qquad W(n) =$ worst case behaviour

Typical leading dependences

Typical leading dependences on the number n of objects in algorithms with examples of algorithms for sorting and searching.

function	sorting	searching
$f(n) = \text{const.}$		hash search
$f(n) = \text{const.} \cdot \log n$		binary search
$f(n) = \text{const.} \cdot n$		sequential search
$f(n) = \text{const.} \cdot n \log n$	quicksort, heapsort	
$f(n) = \text{const.} \cdot n^2$	quicksort with sorted input	
$f(n) = \text{const.} \cdot 2^n$		

4. Searching

Sequential search. Simple sequential search in a list for a given key requires a time proportional to the number of elements:

$$f(n) \propto n$$

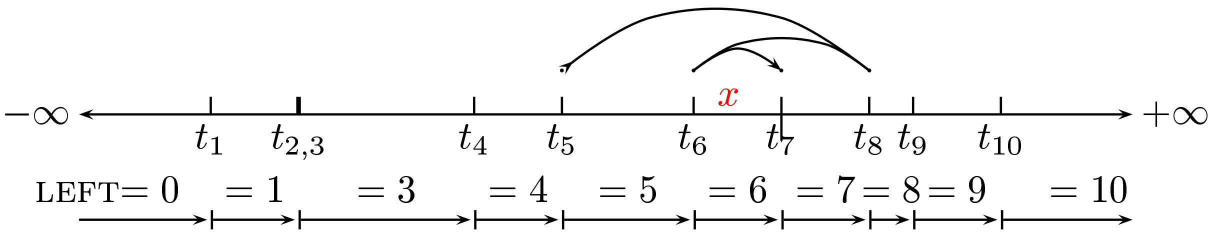
Binary search in sorted lists. Key x compared to the element of the table with index $(n+1)/2$. For $x < t_{(n+1)/2}$ further search is in this part of the table, otherwise in the second part of the table. This method is repeated for the reduced lists, dividing the length of the list each time by two

$$f(n) \propto 1 + \lceil \lg n \rceil$$

comparisons.

Binary search

Binary search for the interval with $t_\ell \leq x < t_{\ell+1}$



Hash search

Simple adressing method: if the keys are integers in the range 0 to $(p-1)$ a table of length p can be used, where the key is directly taken as index. Large values of p require a lot of space.

Generalization: define hash function

$$\begin{array}{lrcl} & \text{hash(key)} & \rightarrow & \text{integer } \; 0 \ldots (p-1) \\ \text{Example:} & \text{hash}(m) & = & m \mod p \end{array}$$

(with large value of p, preferentially a prime number) – the function value is used as index: $f(n) = \text{const.}$

Collisions (where different keys are transformed to the same index) have to be resolved.

Often keys are long alphanumerical strings, which can be represented by an integer. Taking advantage of the properties of the modulo function the following code can be used to avoid very large integers:

```
j = ch(1); for i=2 to k do j = j*256+ch(i) mod p
```