University of Veszprem
Hungary

DATA STRUCTURES

AND

ALGORITHM ANALYSIS

lecture notes

Sandor Dominich
Adrienn Skrop
Maria Horvath

2001.

Mathematics Review

1. Exponents

$$a^b a^c = a^{b+c}$$

$$\frac{a^b}{a^c} = a^{b-c}$$

$$\left(a^b\right)^c = a^{bc}$$

$$a^n + a^n = 2a^n \neq a^{2n}$$

$$2^n + 2^n = 2^{n+1}$$

2. Logarithms

typically $\log_2 = \log$

$\log_a b = x \Leftrightarrow a^x = b$

properties: $\log ab = \log a + \log b$

$$\log \frac{a}{b} = \log a - \log b$$

$$\log_a b = \frac{\log_c b}{\log_c a}$$

$$\log 1 = 0$$

$$\log 2 = 1$$

Note: ln, lg

$\log_2$ is just a compromise

3. Series

Geometric series

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$a = 2, \ \sum_{i=0}^{n} 2^i = \frac{2^{n+1} - 1}{1}$$

$$0 < a < 1, \ \sum_{i=0}^{n} a^i \leq \frac{1}{1-a} \qquad \text{'='} \ n \to \infty$$

$$\left.\begin{aligned} 1 + a + a^2 + a^3 + \ldots &= S \quad | \cdot a \\ a + a^2 + a^3 + a^4 + \ldots &= aS \end{aligned}\right\} -$$

Note: only valid if convergence

$$S = \frac{1}{1-a}$$

$$\text{HW}: \quad \sum_{i=1}^{\infty} \frac{i}{2^i} = 2$$

Arithmetic series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Harmonic number

$$Hn = \sum_{i=1}^{n} \frac{1}{i} = \log_e n + \gamma_n$$
$$\gamma_n : const$$
$$\gamma_n = 0.57721566 \quad \text{Euler's constant}$$
$$Hn \approx \ln n$$

4. Proofs

1. Proof by induction

We want to prove p(n)

1. Prove that p(n) holds for a base case p(n_0)
2. Assume p(n) holds, n > n_0
3. Prove p(n) $\Rightarrow$ p(n+1)

Fibonacci numbers

$F_0 = 1$, $F_1 = 1$ (by definition)

$F_2 = F_1 + F_0 = 2$

$F_i = F_{i-1} + F_{i-2}$, $i \geq 2$

$$F_i < \left(\frac{5}{3}\right)^i, i \geq 1$$

2. Proof by contradiction (indirect, *reductio ad absurdum*)

To prove: P

1. Assume that ¬P holds
2. If we contradict some known property, then:
3. P (must be true)

Ex.:

P: there are infinitely many primes

¬P: there are a finite number of primes

$P_1, P_2, \ldots, P_n$

$P_1 + P_2 + \ldots + P_n$

$N = P_1 \cdot P_2 \cdot \ldots \cdot P_n + 1$

$N > P_i$, $i = 1,\ldots,n$

N is not prime } contradict s

rem. $(N : P_i) = 1 \neq 0$ } Fundamental Theorem of Arithmetic

MUST conclude: ¬P is false ⇔ P is true

Ex.: $\sqrt{2}$ irrational , $\sqrt{2} \in I = R \setminus Q$

P: $\sqrt{2}$ irrational

¬P: $\sqrt{2}$ rational

$\sqrt{2} = a / b$, $(a, b) = 1$

$2b^2 = a^2$ $\quad a = 2c$

$b^2 = 2c^2$ } contradiction

$b = 2d \Rightarrow (a, b) \neq 1$

Note: ¬P

P: The sun is shining

¬P: The sun is NOT shining

<u>It is not the sun that is shining</u>

P: The rain, whose mean value in South Africa exceeds that of Central Europe in august, is only half welcome in parts of a rain forest.

$\forall \varepsilon \; \exists \; S_\varepsilon \quad | x_n - x_m | < S_\varepsilon \, , \; \forall \; n > m$

3. Evaluate

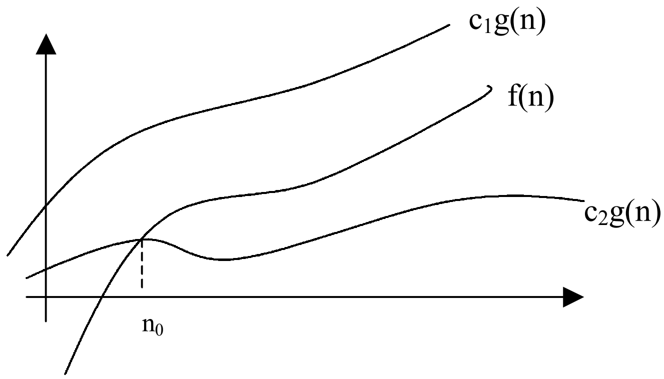
$$\sum_{i=0}^{\infty} \frac{1}{4^i} \, ; \sum_{i=0}^{\infty} \frac{i}{4^i}$$

Prove $\sum_{i=1}^{n} (2i - 1) = n^2$

Asymptotic notation

Θ - notation

$$\Theta(g(n)) \overset{\text{def}}{=} \{ f(n) | \exists c_1, c_2, n_0 : 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n > n_0 \}$$



$$f(n) \in \Theta(g(n))$$
$$f(n) = \Theta(g(n))$$

Θ: asymptotically tight bound
Ex.:

$$f(n) = \frac{n^2}{2} - 3n$$
$$c_1 = \frac{1}{6}, c_2 = \frac{1}{2}, n_0 = 3$$
$$c_1 n^2 \le f(n) \le c_2 n^2$$
$$f(n) = \Theta(n^2)$$

HW: $6n^3 \neq \Theta(n^2)$

Property:

$$p(n) = \sum_{i=0}^{d} a_i n^i$$
$$p(n) = \Theta(n^d)$$

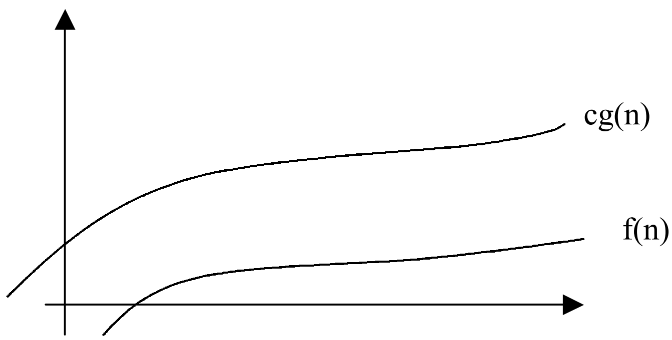
c = const = $\Theta(n^0) = \Theta(1)$

O notation

$$O(g(n)) \overset{\text{def}}{=} \{ f(n) | \exists c, n_0 : 0 \le f(n) \le cg(n), \forall n \ge n_0 \}$$

$n^2 = O(n^2)$

$n = O(n^2)$

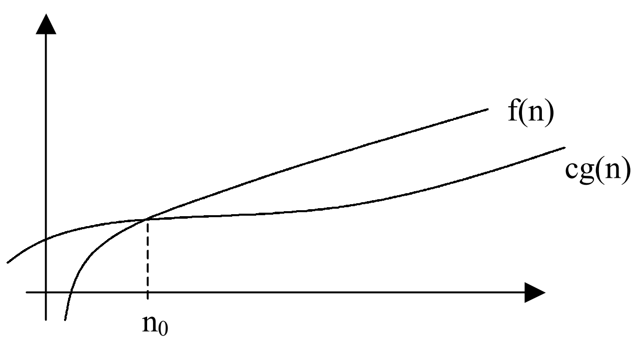


$$f(n) = O(g(n))$$

O upper bound for f(n)

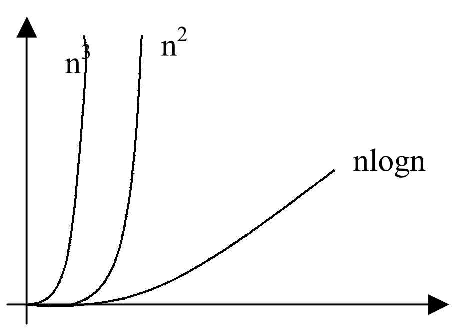
Ω - notation

$$\Omega(g(n)) \overset{\text{def}}{=} \{ f(n) | \exists c, n_0 : 0 \le cg(n) \le f(n), \forall n \ge n_0 \}$$



$$f(n) = \Omega(g(n))$$

Ω lower bound for f(n)



Ex.:

i, S: integers — running time

S: = 0 — measure effective time

FOR i = 1 to n — algorithm analysis

S = S + i * i

1 1 1 — declaration takes no time

assignment: 1 unit

FOR: 1 unit n + 1 unit

$1 + 1 + n + 1 + 3 = n + 6 = \Theta(n) = O(n)$

Similarly (loop in loop):

FOR

….

FOR

….

$\Theta(n^2)$

1. Calculate:

$$\sum_{i=1}^{\infty} \frac{i}{2^i} =$$

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \ldots \Big| \cdot \frac{1}{2}$$

$$\left.\begin{aligned} \frac{S}{2} &= \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \ldots \\ &\quad \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots \end{aligned}\right\} +$$

$$= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \ldots$$

$$S - \left(\frac{1}{2} S + \underset{\substack{\uparrow \\ \frac{1}{2}}}{\sum_{i=2}^{\infty} \frac{1}{2^i}} \right) = \frac{1}{2} \Rightarrow S = 2$$

2. Prove:

a,

$$F_i < \left(\frac{5}{3} \right)^i, i \geq 1$$

$$i = 1, F_i = 1 < \frac{5}{3}$$

$$i = 2, F_i = 2 < \frac{25}{9}$$

$$\mathrm{F}_{\mathrm{i+1}} < \left(\frac{5}{3} \right)^{i+1}$$

$$\mathrm{F}_{\mathrm{i+1}} = \mathrm{F}_{\mathrm{i}} + \mathrm{F}_{\mathrm{i-1}} < \left(\frac{5}{3} \right)^i + \left(\frac{5}{3} \right)^{i-1} = \left(\frac{5}{3} \right)^{i+1} \left(\frac{3}{5} + \left(\frac{3}{5} \right)^2 \right) = \left(\frac{5}{3} \right)^{i+1} \cdot \frac{24}{25} < \left(\frac{5}{3} \right)^{i+1}$$

b,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$n = 1 \Rightarrow 1 = 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} = \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$$

c,

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$n = 1: \ 1 = \frac{1 \cdot 2 \cdot 3}{6}$$

$$\mathrm{n+1}: \ \sum_{i=1}^{n+1} i^2 = \frac{(n+1)(n+2)(2n+3)}{6}$$

$$\sum_{i=1}^{n+1} i^2 = \sum_{i=1}^{n} i^2 + (n+1)^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2 =$$

$$= (n+1)\left(\frac{n(2n+1)}{6} + n + 1\right) = (n+1)\frac{n(2n+1) + 6n + 6}{6} =$$

$$= (n+1)\frac{2n^2 + 7n + 6}{6}$$

$$(n+2)(2n+3) = 2n^2 + 7n + 6 \Rightarrow \text{igaz}$$

3.

$$\sum_{i=1}^{n} (2i-1) = n^2$$

$$\sum_{i=1}^{n} (2i-1) = \sum_{i=1}^{n} 2^i - \sum_{i=1}^{n} 1 = 2\frac{n(n+1)}{2} - n = n^2 + n - n = n^2$$

4.

$$\sum_{i=1}^{n-2} F_i = F_n - 2$$

$$n = 3: \quad \sum_{i=1}^{1} F_i = 1 = F_3 - 2 = 1$$

$$\sum_{i=1}^{n-2} F_i = F_n - 2$$

$$\sum_{i=1}^{n-1} F_i = F_{n+1} - 2$$

$$\sum_{i=1}^{n-1} F_i = F_1 + \sum_{i=2}^{n-1} \left(F_{i-1} + F_{i-2}\right) = F_1 + \sum_{i=2}^{n-1} F_{i-1} + \sum_{i=2}^{n-1} F_{i-2}$$

$$F_{n+1} - 2 = F_n + F_{n-1} - 2 = \left(F_n - 2\right) + \left(F_{n-1} - 2\right) + 2 = \sum_{i=1}^{n-2} F_i + \sum_{i=1}^{n-3} F_i + 2$$

Asymptotic notation in equations

$n = \Theta(n^2)$ $n = O(n^2)$

$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ There exists some functions such that the equation holds.

$2n^2 + \Theta(n) = \Theta(n^2)$

O, Ω, Θ	upper bound: O tight, o loose
o, ω	lower bound: Ω tight, ω loose

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$ g(n) grows much rapidly than f(n)

$$f \in o(g)$$
$$=$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$ f(n) grows much rapidly than g(n)

$$f \in \omega(g)$$
$$=$$

<u>Properties</u>

Transitivity

$(f(n) = .\ (g(n)) \wedge g(n) = .\ (h(n))) \Rightarrow f(n) = .\ (h(n))$

where $. \in \{O, \Omega, \Theta, o, \omega\}$

Reflexivity

$f(n) = .\ (f(n))$

Symmetry

$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Analogy between asymptotic notation and real numbers

$f(n) = O(g(n))$ $a \leq b$

$f(n) = \Omega(g(n))$ $a \geq b$

$f(n) = \Theta(g(n))$ $a = b$

$f(n) = o(g(n))$ $a < b$

$f(n) = \omega(g(n))$ $a > b$

Trichotomy

Dichotomy: Boolean logic

Trichotomic: three different values

Trichotomy: $\forall a, b \in R$: exactly one of the followings holds:

$a < b$ $a = b$ $a > b$

Note: for every real number holds, and we can prove it.

Asymptotic notation: trichotomy doesn't hold

Not any two functions can be compared in an asymptotic sense.

Ex.: $f(n) = n$

$g(n) = n^{1+\sin n}$

They can't be compared.

$f(n) = n$

$g(n) = n^{\sin n}$

They can be compared.

f(n) is a tight upper bound for g(n).

Floor, ceiling

Floor of x ∈ R: $\lfloor x \rfloor$: the greatest integer less than or equal to x

Ceiling of x ∈ R: $\lceil x \rceil$: the least integer greater than or equal to x.

$x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$

$\forall\, n \ni Z: \lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

Rate of growth of polynomials and exponentials

$$\lim_{x\to\infty} \frac{x^n}{a^x} = 0$$
$$n \in \aleph \setminus \{0\}, a \in (1, \infty)$$

Any exponential grows faster than any polynomial!

Series

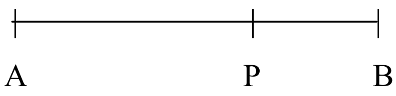
$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$
$$e^x \ge 1 + x$$
$$1 + x \le e^x \le 1 + x + x^2, |x| \le 1$$
$$x \to 0, e^x = 1 + x + \Theta(x^2)$$
$$e^x \approx 1 + x$$
$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n} \right)^n$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots, |x| < 1$$

Stirling approximation:

$$n! = \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$
$$\sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n \le n! \le \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^{n + \frac{1}{12n}}$$

Fibonacci numbers and the Golden ratio

$F_i = F_{i-1} + F_{i-2}$, $i \ge 2$, $F_0 = 0, F_1 = 1$



$$\frac{AB}{AP} = \frac{AP}{PB}$$

$$AP = \frac{AB}{2}\left(\sqrt{5} - 1\right)$$

$$AB = 1, AP = \frac{\sqrt{5} - 1}{2} \cong 0.61803$$

$$\Phi = \frac{1 + \sqrt{5}}{2}, \hat{\Phi} = \frac{1 - \sqrt{5}}{2}$$

$$F_i = \frac{\Phi^i - \hat{\Phi}^i}{2}$$

The relation between the Fibonacci numbers and the Golden ratio.

Fibonacci numbers grow exponentially.

Summations

Sequence: $a_1, \ldots, a_n$ (finite)

Series: $a_1 + \ldots + a_n + \ldots = \sum_{i=1}^{\infty} a_i$ (infinite)

$\lim_{n \to \infty} \sum_{k=1}^{n} a_k$ If it exists we have convergent series (else divergent).

Linearity

$$\sum_{k=1}^{n} (ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

$$\sum_{k=1}^{n} \Theta(f(k)) = \Theta\left(\sum_{k=1}^{n} f(k)\right)$$

Differentiation

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}, |x| < 1$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1 - x)^2}$$

Telescoping

Telescoping series (sums)

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = \cancel{a_1} - a_0 + \cancel{a_2} - \cancel{a_1} + \cancel{a_3} - \cancel{a_2} + \ldots + a_n - \cancel{a_{n-1}} = a_n - a_0$$

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{n}$$

Products

$$\lg\left(\prod_{k=1}^{n} a_k \right) = \sum_{k=1}^{n} \lg a_k$$

Bounding summations

By induction

Prove: $\sum_{k=0}^{n} 3^k = O(3^k)$

$$\exists c : \sum_{k=0}^{n} 3^k \le c \cdot 3$$

$$\text{Initial value} : n = 0$$

$$1 \le c \cdot 1, c \ge 1$$

Assume:

$$\sum_{k=0}^{n+1} 3^k = \sum_{k=0}^{n} 3^k + 3^{n+1}$$

$$\le c \cdot 3^n + 3^{n+1}$$

$$= \left(\frac{1}{3} + \frac{1}{c} \right) c \cdot 3^{n+1}$$

$$\le c \cdot 3^{n+1}$$

By bounding terms

$$\sum_{k=1}^{n} a^k, a_{\max} = \max_k a_k$$

$$\sum_{k=1}^{n} a^k \le n a_{\max}$$

Ex. $\sum_{k=1}^{n} k \le n^2$

By splitting

n even

$$\sum_{k=1}^{n} k = \sum_{k=1}^{\frac{n}{2}} k + \sum_{k=\frac{n}{2}+1}^{n} k$$

$$\geq \sum_{k=1}^{\frac{n}{2}} 0 + \sum_{k=\frac{n}{2}+1}^{n} \frac{n}{2}$$

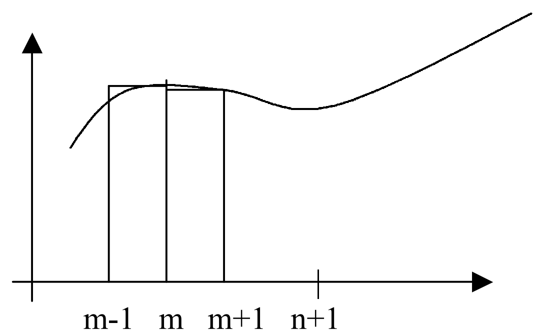
$$\geq 0 + \frac{n^2}{4}$$

$$= \Omega\left(n^2\right)$$

By integrals

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$

Ex. $\displaystyle\sum_{k=1}^{n} \frac{1}{k} \geq \int_{1}^{n+1} \frac{dx}{x}$



Elementary Data Structures

Data should be organised into stuctures so that it can be processed as required by the problem

Elementary (Basic or fundamental):

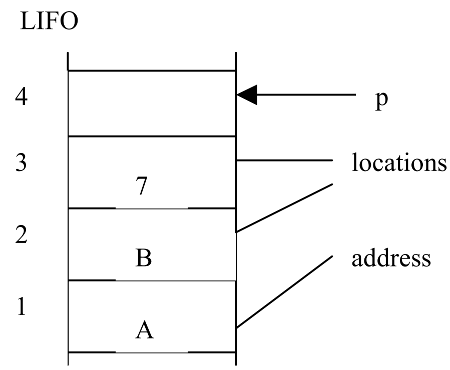
There are just a few elementary data structure. All the other rely on the elementary data structures.

STACK

a) LIFO (Last in, first out)
b) FIFO (First in, first out)

The stack is organised into locations: address
p: pointers
stack pointer: the address of the next available locations

Note:
p: the address of the top which isn't empty (another view)



Two basic operations:

- write: PUSH: D (new data) → p (location) ; p → p+1
- read: POP: p → p-1; p(location) → D

Note:

In theory : the stack is infinite
In practice: finite
it's impossible to increment the stack pointer: overflow: the stack is full

before writing overflow check is needed

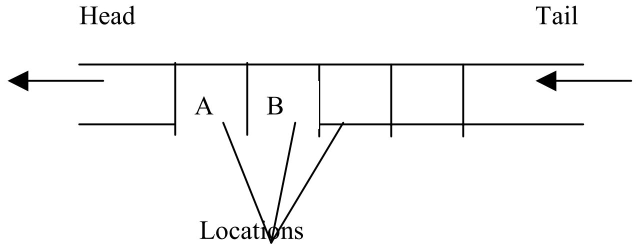
Note:
before read: check whether the stack is empty (underflow check) (we want to extract a data from the empty stack)

Application:

- management of teh main memory
  when the OS is loaded
- compilers: evaluating expressions

FIFO

it's called a QUEUE



It is infinite at both ends. Tha data is written only from one direction at one end.
Tail: where we write the data in
Head: where we read the data out

The space we can allocate for a QUEUE is finite.

- overflow check: ENQUEUE
- underflow check: DEQUEUE

write: ENQUEUE
read: DEQUEUE

Application:

- modelling the dataflow in online processing (pressure, temperature)

ARRAY

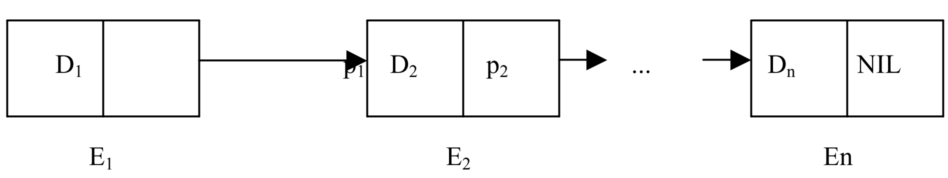
1 dimensional – sequence: $a_1, a_2, a_3, \dots, a_n$
2 dimensional – matrix: $(a_{i,j})_{n \times n}$
higher dimensional – matrix of matrices

Usage: represent stack, queue, list…

LIST

L : $E_1, E_2, \dots, E_i, \dots E_n$ the elements of the list
$\forall E_i$: D_i p_i data and pointer fields
p_i can contain several pointers: p_i^j, $j \in \{1, \dots\}$

j=1 single linked list



NIL: no more list elements
p_1: the address of E_2
p_i: the address of E_{i+1}
Important: the order in wich the elements are linked through the pointers (the logical order) not necessarily the same as the physical order.

Example: Application:

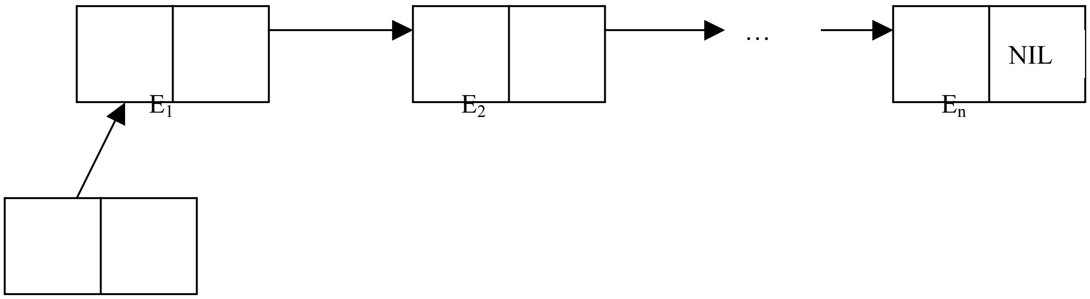
to store data on disk (files)

element = track : the elements are stored on available tracks, ??????????

Operations:

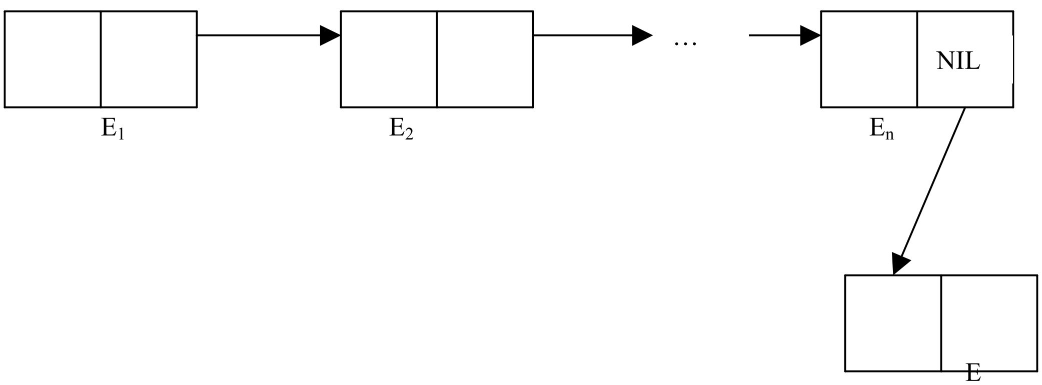
INSERT: write a new element into the list

a. front of the first element of the list



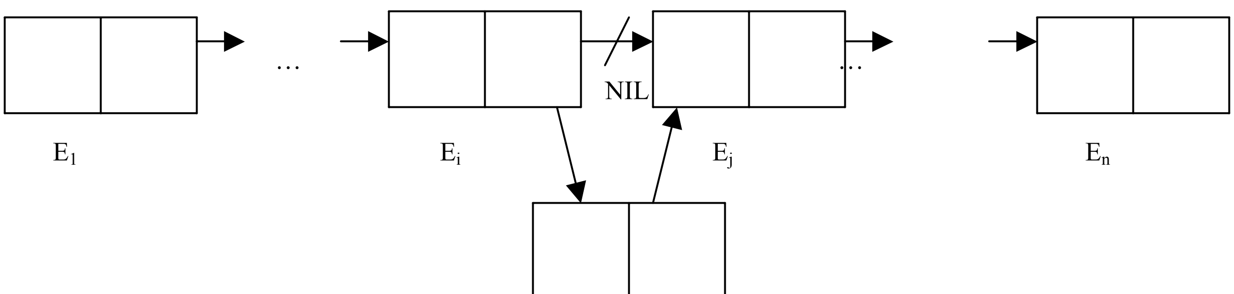
pointer of E points towards E_1

b. after the last element



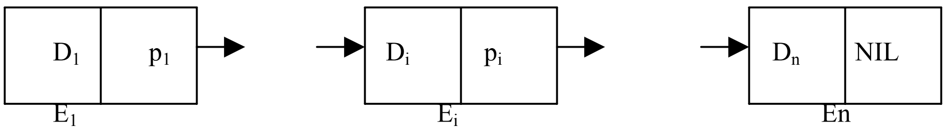
Replacing NIL with a pointer to E

c. between the existing elements



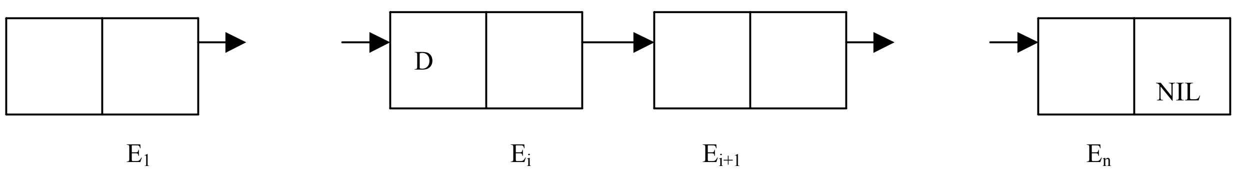
Rearraging the pointers.

SEARCHING: Θ (n)



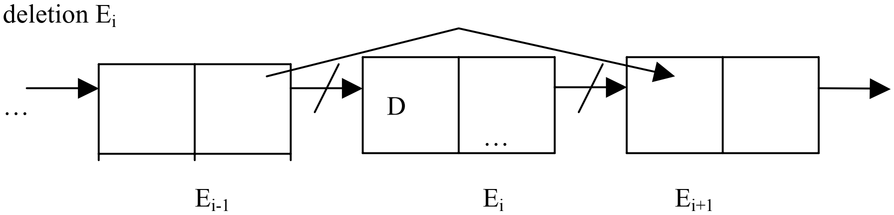
D : to be searched sequentially: compare D with D_i i=1,… we either find it or not

DELETION: delete an existing element



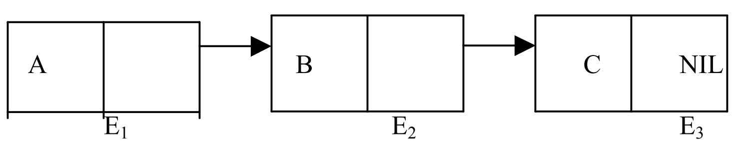
D: find the element contains D and take it out from the list

a) searching
b) deletion E_i



Logical deletion: just the pointers are rearranged, not physical deletion

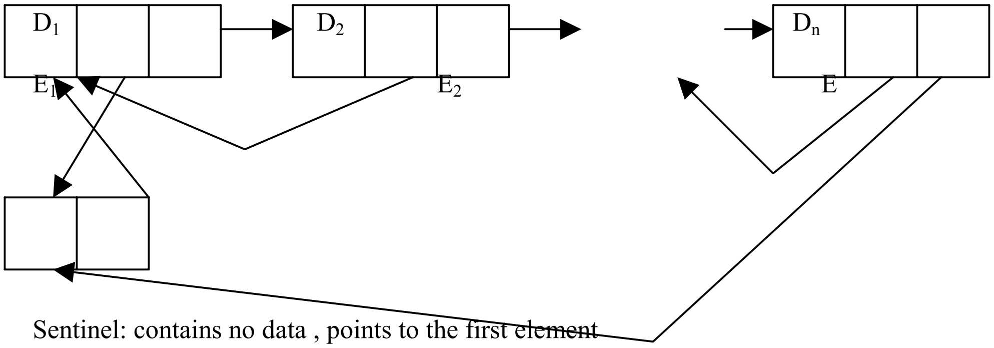
Example: array representation of a list



two dimensional array

	1	2	3	
1	A	B	C	← Data
2	(1,2)	(1,3)		← Pointers

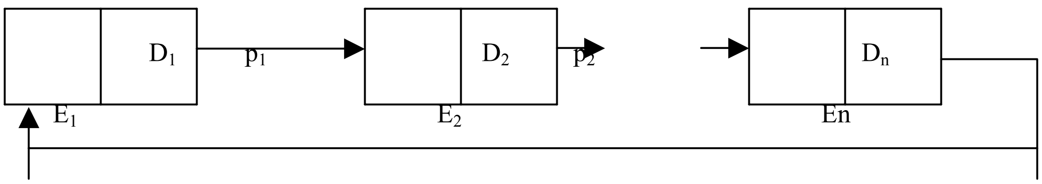
Doubly linked list



Sentinel: contains no data , points to the first element

Circular list

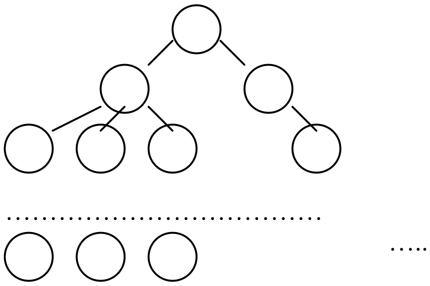
Circular singly linked list



The sentinel always tells us which is the first element of the list.

TREE

Mathematically: acyclic connected graph



ROOT level 0: parent of its children, has no parent

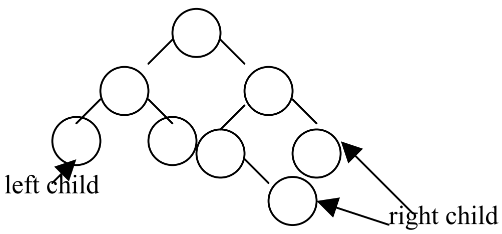
level 1: children of the root, parents of level 2

level 2: children of the nodes of level 1, parents of level 3

…

last level: leaves: have no children

Binary tree: at most two children (except the leaves)

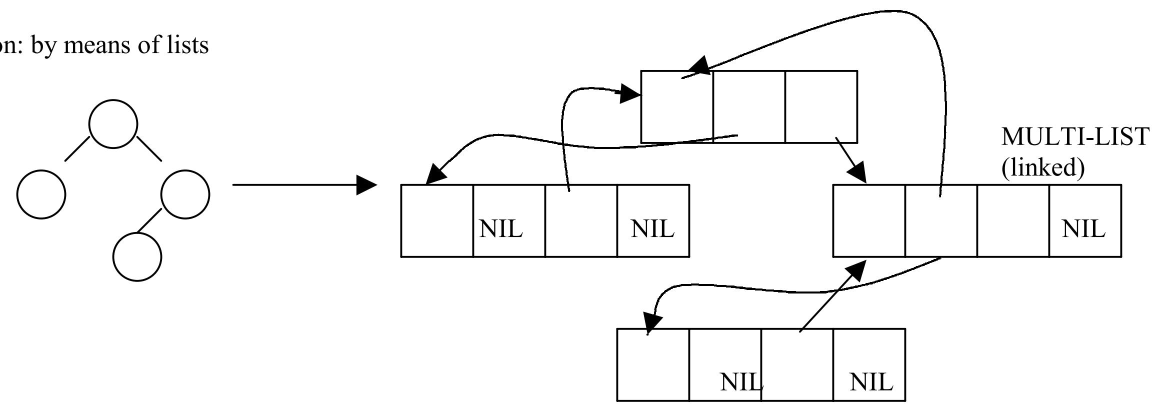


ROOT number of levels: HIGHT (h) of the tree

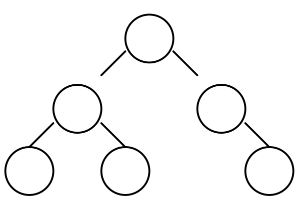
number of leaves $\leq 2^h$

Implementation: by means of lists

Example:



Binary search tree

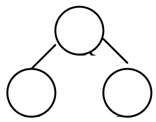


Every key in every left subtree is at most the key of its root.
Every key in every righta subtree is at most the value of its root.

↓

Binary search tree property

WALK:



INORDER: L Root R
PREORDER: Root L R
POSTORDER: L R Root

Example:

INORDER: 2 3 5 5 7 8 → ascending order of keys
PREORDER: 5 3 2 5 7 8
POSTORDER: 2 5 3 8 7 5

Application:

(A+B)*C-D mathematical form (usual)

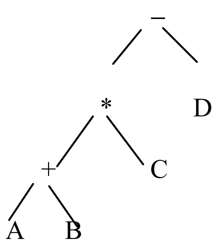
Evaluation of an arithmetic expression :

- parentheses
- operators

} precedence rules

form: Pohsh: no parentheses

no need the check the precedence of the operators

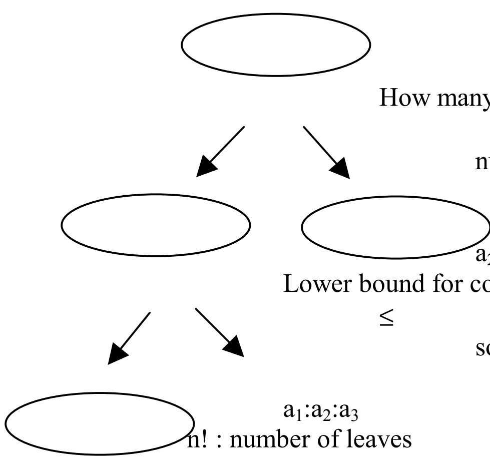


postorder walk: AB+C*D- POSTFIX POLISH FORM

Application:
SORTING: arrange data

numbers 1, 7, 6 → 1, 6, 7
Comparison

Decision tree (specialised binary tree)
a_1, a_2, a_3



$a_1:a_2$

How many comparisons needed to sat the numbers?

≤ >

$a_2:a_3$ $a_1:a_3$

Lower bound for comparison – bored sorting

≤ >

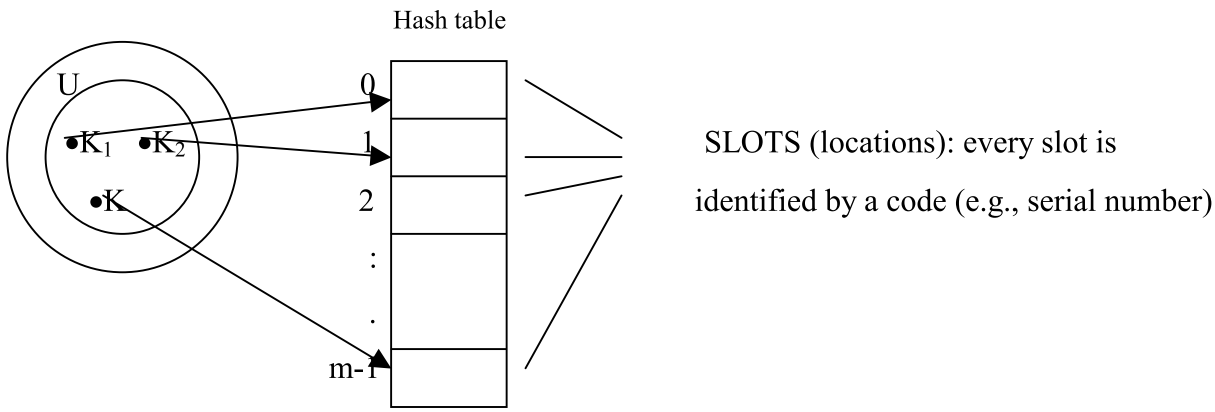
$a_1:a_2:a_3$ . . .

n! : number of leaves

$n! \leq 2^h$

$h \geq \log (n!)$

HASHING



U: Universe of all possible key values (from where the keys can take values).

K_1, K_2, K: keys, they are used to identify records.

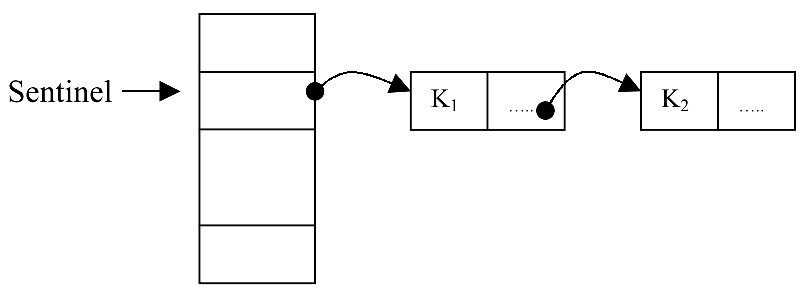
<u>Hash function</u>: h, meets the key values with the slots.

h: U $\rightarrow$ {0, 1, …, m-1}

h: K $\rightarrow$ h(K)

<u>Collision</u>: $h(K_1) = h(K_2)$ when $K_1 \neq K_2$

Resolve: chaining, data that collide are chained.



Complexity

Analysing hashing with chaining.

Search: $\Theta(1+\alpha)$

n: number of keys

m: number of slots

$\frac{n}{m}$: average number of elements in a list

$\alpha = \frac{n}{m}$: load factor (the analysis is made in term of α)

Search:

K

h(K) $\Theta(1)$

search in the list $\Theta(\alpha)$

$\Theta(1) + \Theta(\alpha) = \Theta(1 + \alpha)$

Assume: n = O(m) (as many keys as slots)

$$\alpha = \frac{n}{m} = \frac{O(m)}{m} = O(1)$$

The search takes constant time.

Uniform hashing: any given key is equally likely to hash into any of the slots.

Probability:1 / m

<u>Application</u>

- Spelling checker
- Compilers
- Game – playing
- Graphs

Hash function

What makes a <u>good</u> hash function? (Uniform hashing is ensured 1 / m)

Usually we do not know the distribution of the key values → difficult to design good hash function.

In practice: spread the key values as much as possible.

Division method

$h(K) = K \bmod m$

It does not work well in every case → $m = 2^p$

Reminder [ ] p bits

[|]

$p = 2 \quad m = 2^2$

0, 1, 2, 3 2 bits

$K = 5$ [1 | 0 | 1] 3 bits

M should be a prime near α

$n = 2000$, $\alpha = 2000 / 3$

Multiplication method

$h(K) = \lfloor m((KA) \bmod 1) \rfloor$

$0 < A < 1$, $A = 0.618033$ (Golden section) → good results

Uniform method

$h(K) = \lfloor Km \rfloor$

K – uniform distribution [0, 1]

Practically and theoretically good function

Interpreting strings as numbers

SOUNDEX CODING

RADIX 128

C1 C2 …Cn ASCII

C1 C2 …Cn number $0 \le Cj \le 127$

Ex. $p = 112$, $t = 116$, $pt = 116 + 128 \cdot 112 = 14452$

Universal hashing

There are h hashing functions s.t. there exist key values that more than one will be hashed into the same slot.

Any fixed hashing is valid.

$H = \{h_1, h_2, \ldots, h_r\}$ set of hash functions

For any random $h_i \rightarrow$ uniform hashing

Statement

If h is chosen from a universal collection H of hash functions and is used to hash n keys into a hash table of size m, where $n \le m$, then the expected number of collisions involving the particular key x is less than 1.

Proof

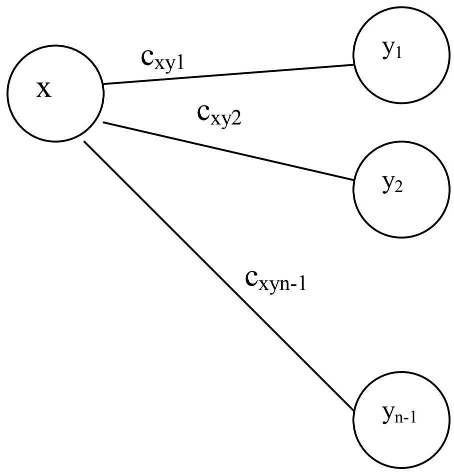
$$\text{Let } c_{yz} \text{ a random variable} : c_{yz} = \begin{cases} 1, h(y) = h(z), \forall y, z \\ 0, \text{otherwise} \end{cases}$$

H universal: probability for y and z to collide is 1 / m (by definition)

$$E[c_{y,z}] = \frac{1}{m}$$

$$\begin{pmatrix} \xi, P_n = P(\xi = x_n) \\ E[\xi] = \sum_n p_n x_n \end{pmatrix}$$

Let c_x be the number of collisions involving key x.



$$E[c_{xy1}] = \frac{1}{m}$$

$$E[c_{xy2}] = \frac{1}{m}$$

$$E[c_{xyn-1}] = \frac{1}{m}$$

$$E[c_x] = \sum_y E[c_{xy}] = \sum_y \frac{1}{m} = \frac{n-1}{m} < 1$$

$$\left(\begin{array}{l} c_x = c_{xy1} + \ldots + c_{xyn-1} \\ E\left[\sum_i \xi_i\right] = \sum_i E[\xi_i] \end{array} \right)$$

Construction of a universal set H.

m – prime

given key x

decompose $x = x_0\ x_1 \ldots x_r$, value $x_i < m$

{0, 1, …, m-1}, let a be a sequence of slots

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m$$

$$H = \bigcup_a \{h_a\} \text{ the universal set}$$

$a = \langle a_0\ a_1 \ldots a_r \rangle$, every a_i is chosen randomly from the set {0, 1, …, m-1}

Theorem

The set H is universal.

Proof

$$x \neq y$$

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m$$

$$h_a(y) = \sum_{i=0}^{r} a_i y_i \bmod m$$

$$y = y_0 y_1 \ldots y_r$$

$$x \neq y \quad x_0 \neq y_0$$

$$h_a(x) = h_a(y)$$

$$\sum_{i=0}^{r} a_i x_i \bmod m = \sum_{i=0}^{r} a_i y_i \bmod m$$

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \quad (\bmod m)$$

$$a_0(x_0 - y_0) \equiv \sum_{i=1}^{r} a_i(x_i - y_i) \quad [\bmod m]$$

As many collisions as the number of equations.

As many equations as different a_i.

$$m^r$$

$$|H| = m^{r+1}$$

$$\frac{m^r}{m^{r+1}} = \frac{1}{m}$$

H(n m) n, m should be comparable.

Choose at random any time when hashing should be done, apply h.

Application

Data bases

Hash table + B- tree

Not easy to estimate n

Not easy the choice of h.

ESTIMATION OF COMPLEXITY

Running time: the time required for a computer to execute a program.

Running time dependes on the following factors:

- CPU: the higher the speed, the quicker the computer.
- Memory: main and secondary memory available to execute the program.
- Input data: size, type, operations.
- Software: compiler, operating system, etc.
- Algorithm (based on which the particular program is written).

Example: $a_1 + a_2 + a_3 + a_4$

S = 0	S = 0
$S = S + a_1$	FOR i = 1 TO 4
$S = S + a_2$	$S = S + a_i$
$S = S + a_3$	
$S = S + a_4$	

The asymptotic notation is a way to express how bad (slow) or good (fast) an algorithm is.

Expression of complexity → we get a measure → to express the 'character' or behaviour of an algorthm

→ comparison of algorthms in term of complexity.

We can decide which algorithm is better or we should choose.

Note: the complexity will not guarantee that the algorithm will really be faster or that the computer will always execute the program faster, because physical running time, as seen above, depends on other factors, too.

Estimation of the complexity of an algorithm:

1. Assignment 1 umit Ex.: S = 0
   Operations 1 unit each
2. Consecutive statement sum of each
3. Loop: time required to evaluate the body multiplied by the number of iterations
4. Nested loop: inside multiplied by the product of iterations
5. IF: test + max (THEN, ELSE)

This technique can over–estimate (ex. 5), but it will never under–estimate the complexity.

SORTING

To arrange given data according to given criteria.

Ex.: 1. c_i, $i = 1, \dots, n$

α_j, $j = 1, \dots, m$ (criteria)

arrange c_i taking into account every α_j

2. List of names → alphabetic order
3. Temperature values → ascending or descending order
4. Sorting $a_1, a_2, \dots, a_n$ input data

criterion: ascending (descending)

BUBBLE SORTING

Main idea: find the smallest and put it on the top

find the second smallest and put it next to the top one …

Ex. 2, 1, 7, 6 — Fix the first number and compare it with all the other. If wrong order swap and change.

1, 2, 7, 6

(1) — Repeat from the second position.

2, 7, 6   7, 6   → 1, 2, 6, 7

(2)   (6) (7)

$a_1, a_2, \dots, a_i, a_j \dots, a_n$

```
FOR i = TO n – 1
    FOR j = i + 1 TO n
        IF ai > aj THEN swap (ai, aj)
```

$2 \cdot (n - 1)(n - 1) = O(n^2)$

Best case: the input is already in the right order (no swap) $\rightarrow O(n^2)$
Worst case: all the numbers are in the wrong order $\rightarrow O(n^2)$

Average case: the numbers are given at random (typical case)

Bubble sorting (comparison-based): $O(n^2)$

Comparison-based sorting: $\Omega(n \log n)$

$O(n \log n)$

QUICK SORTING

The fastest known method.

Divide and conquer philosophy

A(p…r) → $A(p, q), \forall x \le q$

A(p…r) → $A(q + 1, r), \forall y \ge q$

q: computed value

Ex. 9, 2, 11, 20, 7

Q: pivot = $\lfloor (9 + 7) / 2 \rfloor = 8 \rightarrow$ not necessarily belongs to the sequence

7, 2   11, 20, 9

q = 4   q = 10

2   7   9   11, 20

q = 15

2   7   9   11   20

```
QUICKSORT (A, p, r)
IF p < r THEN
    q ← PARTITION (A, p, r)
     QUICKSORT (A (p, q))
     QUICKSORT (A (q + 1, r))
Initial call: QUICKSORT (A, 1, length(A))
```

PARTITION (A, p, r)

x ← A(p), i ← p – 1, j ← r + 1

WHILE TRUE DO

REPEAT j ← j – 1 UNTIL A(j) ≤ x

REPEAT i ← i + 1 UNTIL A(i) ≥ x

IF i < j THEN SWAP (A(i), A(j))

ELSE RETURN j

Recurrence: resolved by telescoping

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad |: n$$

$$\left.\begin{aligned} \frac{T(n)}{n} &= \frac{T\left(\frac{n}{2}\right)}{\frac{n}{2}} + 1 \\ \frac{T(n)}{\frac{n}{2}} &= \frac{T\left(\frac{n}{4}\right)}{\frac{n}{4}} + 1 \\ &. \\ &. \\ &. \\ \frac{T(2)}{2} &= \frac{T(1)}{1} + 1 \end{aligned}\right\} + \qquad \frac{T(n)}{n} = \frac{T(1)}{1} + \log n$$

T(n) = nc + nlogn

T(n) = O(nlogn) average (and best) case

Worst case: $O(n^2)$

Worst case: one element / region

$$\left.\begin{aligned}
&T(n) = T(n-1) + \Theta(n)\\
&T(n-1) = T(n-2) + \Theta(n-1)\\
&.\\
&.\\
&.\\
&T(2) = T(1) + \Theta(2)
\end{aligned}\right\}+$$

$$T(n) = T(1) + \sum_{k=2}^{n} \Theta(k) =$$

$$T(n) = \Theta(n^2) = \sum \Theta(k) = \Theta\left(\sum k\right)$$

SEARCHING

There may be different situations where searching is performed

SEQUENTIAL SEARCH

Given $a_1, \ldots, a_i, \ldots, a_n$ (numbers / characters: objects to be searched)

Find: x

(naïve or brute force search or straight search)

It is in fact one loop:

FOR i = 1 TO n

IF $x = a_i$ THEN STOP

O(n)

Note: this kind of search is very simple (primitive),

but what is if a_i is a matrix?

records of a file?

The comparison is more complicated here. But in principle it is very simple.

RANDOM SEARCH

Introduce some sort of probability.

Given $a_1, \ldots, a_i, \ldots, a_n$ to be searched.

Find: x

Coin: probability element

Flip a coin
- If head: search from 1 to n.
- If tail: search from n to 1.

Two sequential searches are combined, applied together.

Assume: $x = a_i$ (i^{th} position)

The coin is fair: the head and the tail occur with equal probabilities.

If head: i comparisons to find x.

If tail: n – i + 1 comparisons to find x.

½ i + ½ (n – i + 1) = (n + 1) / 2 better than n. (Average the two case.)

In average we need (n + 1) / 2 comparisons rather than n.

The order of the elements is irrelevant (no need for pre-sorting) in

- Sequential search,
- Randomized search.

BINARY SEARCH

It is very quick and used almost everywhere.

The elements to be searched are sorted.

Given $a_1 \leq \ldots \leq a_i \leq \ldots \leq a_n$

Find: x

Idea: guess the number I am thinking at

Ex. 2 3 7 | 8 <u>9</u> | 10

Find: 9

- Half the sequence.
- Compare the last element with x.

$a_1, \ldots, a_n$

x

low: leftmost element in the half = 1

high: rightmost element in the half = n

```
REPEAT
        mid = (low + high) DIV 2
        IF low > high THEN mid = 0
        ELSE
        IF a(mid) < x THEN low = mid + 1
        ELSE high = mid – 1
UNTIL x = a(mid)
```

$\frac{n}{2^0}$

$\frac{n}{2^1}$

.

.

.

$\frac{n}{2^{\lceil \log n \rceil}}$

O(log n) , very fast.

Note: n finite

If n infinite then binary search: div 2^m

It may not happen in practice, just in theory.

Note:

1. Will the search work when all elements are given at once (at the same time)?
2. Will the search work when all elements are given on line (one by one)?

	1.	2.
Sequential search	YES	YES
Randomized search	YES	NO
Binary search	YES	NO

In every case we assume that we have just one processor to do the job.

PARALLEL BINARY SEARCH

It speeds up the binary search.

P processors with shared memory (PRAM parallel RAM).

CREW: Concurrent Read Exclusive Write

Given the elements $a_1, \ldots, a_i, \ldots, a_n$ sorted.

Find: x

Divide the sequence into p + 1 parts:

$a_1, \ldots, \underline{a_{i1}} \mid a_{i1+1}, \ldots, \underline{a_{i2}} \mid \ldots, a_{ij} \mid \ldots, \underline{a_n}$

x is compared with the boundary elements (or the leftmost or the rightmost)

in parallel: processor j compares x with the j^{th} boundary

processor j sets a variable c_j: — 0, if $x > a_{ij}$
— 1, otherwise

Thus: $\exists s$: $c_s = 0 \wedge c_{s+1} = 1$ (to locate part)

Repeat recursively until $x = a_{ij}$

Complexity:

BS	PBS
$\frac{n}{2^0}$	$\frac{n}{(p+1)^0}$
.	$\frac{n}{(p+1)^2}$
.	
.	
$\frac{n}{2^{\lceil \log n \rceil}}$	$\left.\frac{n}{(p+1)^n}\right\} = 1$

O ($\log_{p+1}$ n) better than O(log n), if p > 1

Note: PBS doesn't online

PBS works offline

STRING SEARCHING (straight search, naïve, brute force)

Idea:

text (string of characters) of length n, i (index) pointer

find: pattern of length m, j

Comparisons from left to right

- match: both i and j are incremented
- mismatch: reset j to the beginning of the pattern

  i set to the position corresponding to moving the pattern to the right one position

Ex. 3 2 4 5 6 text: n = 5

4 5 pattern: m = 2

3 2 4 5 6

4 5

| →

4 5

| →

4 5

O(n · m)

KNUTH–MORRIS–PRATT ALGORITHM

Given text: n, i

pattern: m, j

Both pointers are incremented by 1 as long as there is a match.

Mismatch at position k: j is reset and comparison is restarted.

I. 3 2 3 4 5 i = 1

3 4 j = 1

II. 3 2 3 4 5 i = 2

3 4 j = 2

III. 3 2 3 4 5 i = 2

3 4 j = 1

IV. 3 2 3 4 5 i = 3

3 4 j = 1

O(n + m)

BOYER–MOORE ALGORITHM

text: s1, m

pattern: s2, m

<u>Idea</u>: the pattern is searched from right to left, while it is moved from left to right an appropriate number of positions.

I. 3 2 7 4 5 6

~~4~~ 5 mismatch: no matching characters between pattern and text → we move the pattern m position to the right

II. 3 2 7 4 5 6

~~4~~ 5 mismatch: align the four

III. 3 2 7 4 5 6

<u>4</u> <u>5</u> stop

O(n/m) really very fast.

Application: word processing (spelling checker)

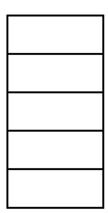
SIGNATURE FILES

Given text

Every word is hashed (transformed) into a string of bits. (Ex. radix 128, soundex coding)

synonym: in hashing sense, not grammatically

hash table:



The locations contains the possible bit representation

→ sector (block)

Find word x:

- x is transformed into a string of bits,
- the hash table is pre–sorted,
- x is searched using binary search in the hash table or better hash function,
- we get a block address on the disk, we find the word on that block somewhere,

– we apply string searching within the block

maximum–sized file
relatively constant
bibliographic DB searching

INVERTED FILE

Huge file (DB) such as in a library, WWW

Search engines work as this: locate a word (not when the Web page is returned)
Crawler (programme): scans the Web all the time. Builds (updates) an inverted file.

Inverted file:

w_1	URL_1
w_2	URL_2
.	.
.	.
w_n	URL_n

Records

w_i: words on the Web
URL–s: containing that word

- search (whether the word exists in the inverted file): B–tree (to implement the inverted file)
- sorting (the inverted file is updated every time a new word appears)

When we enter a word to search:

- The search engine locates the word by searching the inverted file → string searching.
- If the word is found in the inverted file then — depending on the retrieval techniques — the document / URL will be presented or not (this depends not solely on whether the word is present or not).