Analysis of Algorithms

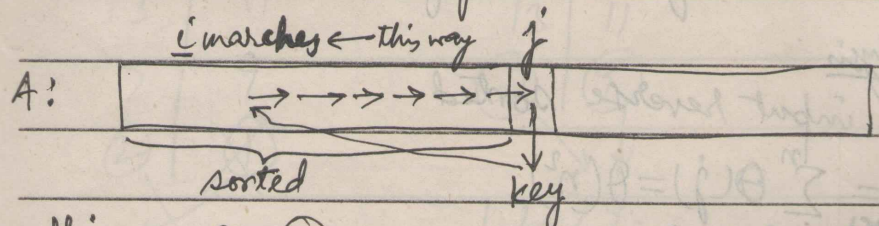
— Sorting Problem:

* input sequence: $\langle a_1 a_2 \cdots a_n \rangle$ of numbers
* output sequence: permutation $\langle a'_1 a'_2 \cdots a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

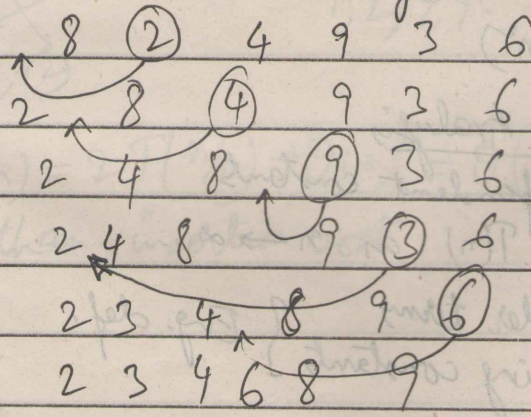
* Insertion sort (A, n)

```
for j ← 2 to n                          c1   n
do  key ← A[j]                          c2   n-1
    i ← j-1                             c4   n-1
    while i ≥ 0 and A[i] > key          c5   Σ_{j=2}^{n} t_j
    do  A[i+1] ← A[i]                   c6   Σ_{j=2}^{n} (t_j - 1)
        i ← i-1                         c7   Σ_{j=2}^{n} (t_j - 1)
    A[i+1] ← key                        c8   n-1
```

Pseudo code



example:



Running time:

- depends on input (e.g. already sorted)
- Depends on input size (e.g. number of element/input)
- parametrize input size
- want upper bound
- guarantee to user

Kinds of analysis

Worst-case analysis (usually)
$T(n)$ = max time on any input of size 'n'.
average case (sometimes)
$T(n)$ = expected time over all inputs of size 'n'
(used assumption of probability distribution)

Best-case (bogus)
(Cheat with slow alg. that works fast on some input)

Insertion sort analysis

Worst case: input reverse sorted

Worst-case → Time $= \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$

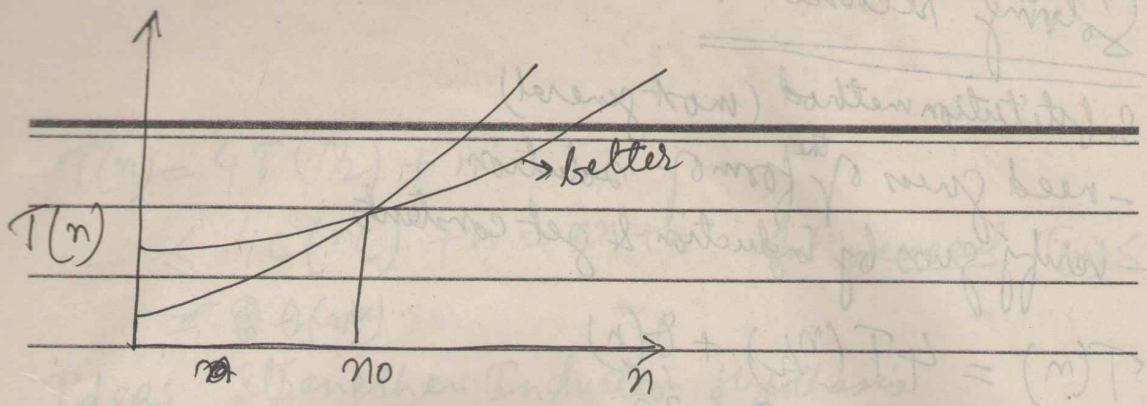
Ar. case → $T(n) = \sum_{j=2}^{n} \Theta(\frac{j}{2}) = \Theta(n^2)$

* Big Idea: Asymptotic Analysis

1. ignore machine-dependent constants
2. look at growth of $T(n)$ as $n \to \infty$

Θ-notation: drop low-order terms
Ignore leading constants } Eng. def.

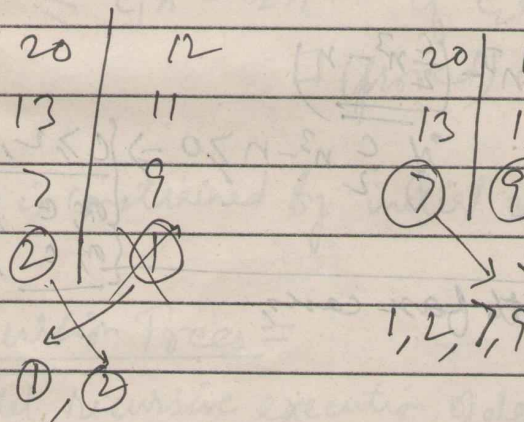
ex: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$



* MERGE-SORT $A[1..n]$ | $T(n)$

1. If $n=1$, done | $\Theta(1)$
2. Recursively sort $A[1 \cdots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \cdots n]$ | $2T(\frac{n}{2})$
3. Merge 2 sorted lists | $\Theta(n)$

Ex:



$T(n) = \Theta(n)$ on n total elements

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$

* better than insertion sort

Solving recurrences

1. Substitution method (most general)
   - need guess of the form of solution
   - verify guess by induction & get constants

$T(n) = 4T(n/2) + \Theta(n)$

Guess: $T(n) = O(n^3)$

Assume by induction $T(k) \leq ck^3$ for $k < n$

Prove $T(n) \leq cn^3$

$T(n) = 4T(n/2) + \Theta(n)$

$\leq 4c(n/2)^3 + \Theta(n)$ ($n/2 < n$)

$= \frac{c}{2}n^3 + \Theta(n) = cn^3 - \left(\frac{c}{2}n^3 - n\right)$

$\leq cn^3$ if $\frac{c}{2}n^3 - n \geq 0 \Rightarrow$ { $c \geq 2, n \geq 1$; or c ; n ; or c , n

Need to ground induction with base cases,

Initial conditions:

$T(n) = \Theta(1)$ for $n \leq n_0$ for const. n_0

- For $1 \leq n \leq n_0$, $T(n) = \Theta(1) \leq cn^3$ if we choose c large enough □

This bound found is not tight.

Guess: $T(n) = O(n^2)$

Prof. Assume $T(k) \leq ck^2$ for $k < n$

$T(n) = 4T(n/2) + n$

$\leq 4c(n/2)^2 + n = cn^2 + n$ but we need $T(n) \leq cn^2$

$\neq O(n^2)$

Idea: Strengthen Induction hypothesis

$T(k) \leq c_1 k^2 - c_2 k$ subtract lower order term

$\therefore T(n) \leq 4(c_1(n/2)^2 - c_2(n/2)) + n$

$= c_1 n^2 - 2c_2 n + n$

$= c_1 n^2 - c_2 n - (c_2 n - n)$

$\leq c_1 n^2 - c_2 n$ if $c_2 n - n \geq 0$ (e.g. $c_2 \geq 1$)

□ (proved)

c_1 is constrained by initial condition

2. Recursion Trees =

- model recursive execution of algorithm
- useful for generating a guess for substitution method

ex: $T(n) = T(n/4) + T(n/2) + n^2$

$T(n) = T(n/4) + T(n/2) + n^2$

$= (n/4)^2 + (n/2)^2 + n^2$

$= \frac{5}{16}n^2$

$T(n/16)\ T(n/8)\ T(n/8)\ T(n/4)$ $= \frac{25}{256}n^2$

$\vdots$

$T(1)$

Total: $T(n) = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \cdots\right) \leq O(n^2)$

3. Master method:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

($a \geq 1$, $b > 1$, f asymptotic positive)

$f(n) > 0$ for suff. large n

3 common cases: log

compare $f(n)$ vs. $n^{\log_b a}$

① $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for const. $\varepsilon > 0$

"$f(n)$ is polynomially smaller than $n^{\log_b a}$"

Solution: ① $T(n) = \Theta\left(n^{\log_b a}\right)$

② "$f(n)$ and $n^{\log_b a}$ grow at similar speeds"

$$f(n) = \Theta\left(n^{\log_b a} \log^k n\right) \longrightarrow (\log_2 n)^k$$

for const. $k \geq 0$

Solution: $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$

③ "$f(n)$ grows polynomially faster than $n^{\log_b a}$"

$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$, $\varepsilon > 0$

and regularity condition

$a f\left(\frac{n}{b}\right) \leq c f(n)$ for const $c < 1$

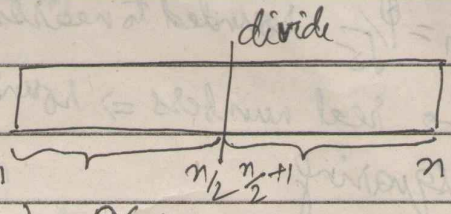
Solution: $T(n) = \Theta(f(n))$

— Divide and conquer paradigm :—

1. divide problem (instance) into subproblems
2. Conquer subproblems by solving recursively
3. Confine subproblems solutions

ex.

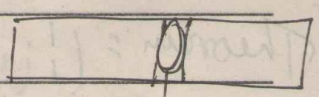
— Merge sort :



$T(n) = 2T(n/2) + \Theta(n)$

master method for solution = $(n \log n)$

= Binary search = (on sorted array

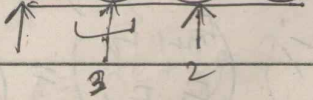


① divide : Compare with middle element

② Conquer : search in 1 subarray

③ Combine : trivial

ex. ③ ⑤ ⑦ ⑧ ⑨ ⑫ ⑮ Find ⑨



* Powering a number ⇒ $a^n = \underbrace{a.a.\cdots.a}_{n} \Rightarrow T(n) = \Theta(n)$

$$a^n = \begin{cases} (a^{n/2})(a^{n/2}) & \text{if even } n \\ a^{\lfloor n/2 \rfloor}\, a^{\lfloor n/2 \rfloor}\, a & \text{if odd } n \end{cases}$$

divide & conquer recursive squaring

$T(n) = 1T(n/2) + \Theta(1)$

$= \Theta(\lg n)$

Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ F_{n-2} + F_{n-1} & \text{if } n \geq 2 \end{cases}$$

0 1 1 2 3 - - -
$F_0 F_1 F_2 F_3 F_4$ - - -

Algorithms:

- naive recursive: exponential time
  $\Theta(\phi^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$
  know: $F_n = \phi^n/\sqrt{5}$ rounded to nearest integer
  → real numbers ⇒ roundoff (error)
- recursive squaring

✓ bottom-up algorithm (re-use #s) (efficient)
$T(n) = \Theta(n)$

✓✓ Theorem: $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$ (more efficient)

recursive squaring on $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ $T(n) = \Theta(\lg n)$

Proof: induction on n

for base $n=1$: $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$

step: $n \geq 2$; $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$

Matrix multiplication :-

input : $A=[a_{ij}]$, $B=[b_{ij}]$ $\left.\begin{matrix}\\\\\end{matrix}\right\}$ $i,j = 1,2,\dots,n$

output : $C = A \cdot B = [c_{ij}]$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

standard alg: $\Theta(n^3)$

for $i \leftarrow 1$ to n
  do for $j \leftarrow 1$ to n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ to n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} b_{kj}$

divide and conquer alg.:

idea: carve $n \times n$ matrices into 2×2 matrix of $\frac{n}{2} \times \frac{n}{2}$ submatrices

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{matrix} 8 \text{ mults} \\ 4 \text{ adds} \end{matrix} \quad \text{of } \frac{n}{2} \times \frac{n}{2} \text{ submat.}$$

$T(n) = 8 \cdot T(n/2) + \Theta(n^2)$

(8 = # sub. mults; $n/2$ = size of submat.) $\Rightarrow T(n) = \Theta(n^3)$

Strassen's algorithm:

Idea: Multiply 2×2 matrices with 7 multis

$P_1 = a(f-h)$

$P_2 = (a+b)\cdot h$

$P_3 = (c+d)\cdot e$

$P_4 = d(g-e)$

$P_5 = (a+d)(e+h)$

$P_6 = (b-d)(g+h)$

$P_7 = (a-c)(e+f)$

$r = P_5 + P_4 - P_2 + P_6$

$s = P_1 + P_2$

$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7$

$r = ae + bg$

$s = af + bh$

$t = ce + dg$

$u = cf + dh$

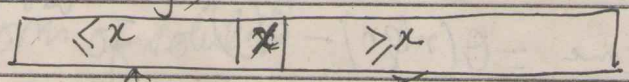
1. divide – partition A, B into $n/2 \times n/2$ submatrices
   – using + and –, form terms to be multiplied
2. conquer → multis rec.
3. combine: from C using + and –

$$T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\lg 7})$$

Quicksort :-

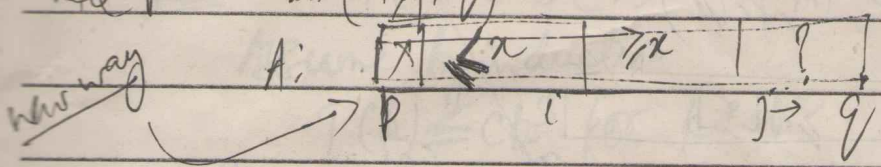
- divide and conquer
- sorts "in place"
- very practical (with tuning)

1. divide :

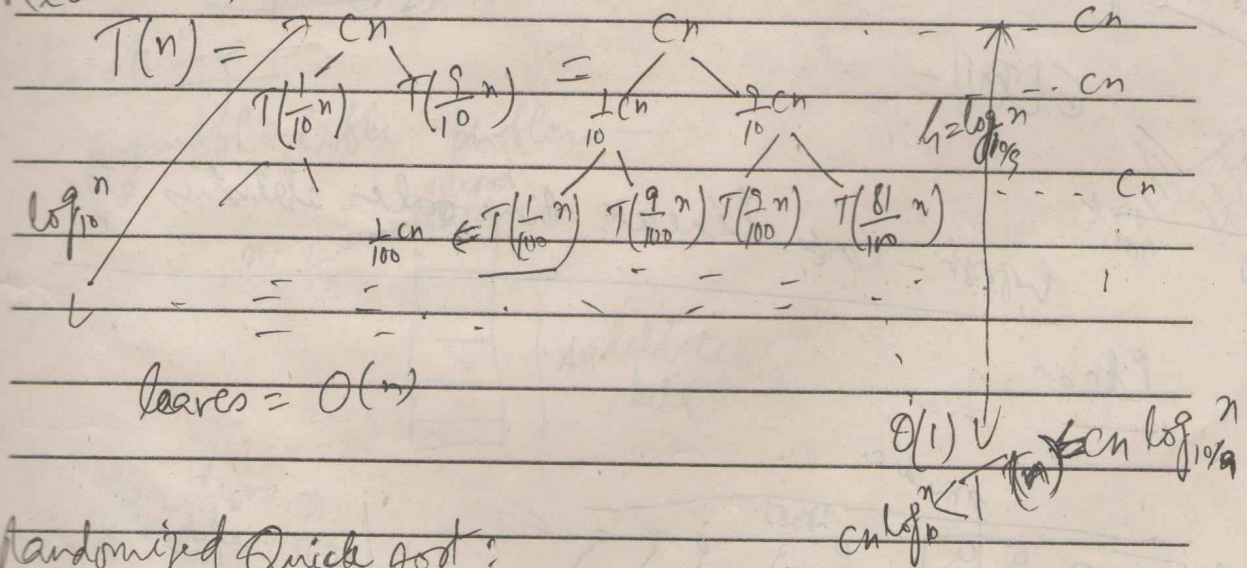


2. conquer : recur. sort.
3. combination :

Partition (A, p, q)



Recursion tree :-



Randomized Quicksort :

pivot around random element

$$E[X_k] = P_r\{X_k = 1\} = \frac{1}{n}$$

$$E[T(n)]$$

- order statistics :

Select ith smallest number in n numbers

$i = 1 \Rightarrow$ min

$i = n$ max.

$i = \lfloor n/2 \rfloor$ or $\lceil n/2 \rceil$: median

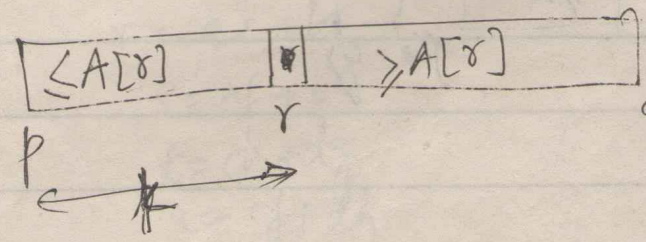
Naive alg: sort and return ith element

Time $= \Theta(n \lg n) - \Theta(1)$ - worst case

$= \Theta(n \lg n)$

Randomized divide-and-conquer alg.

Rand-Select (A, p, q, r)



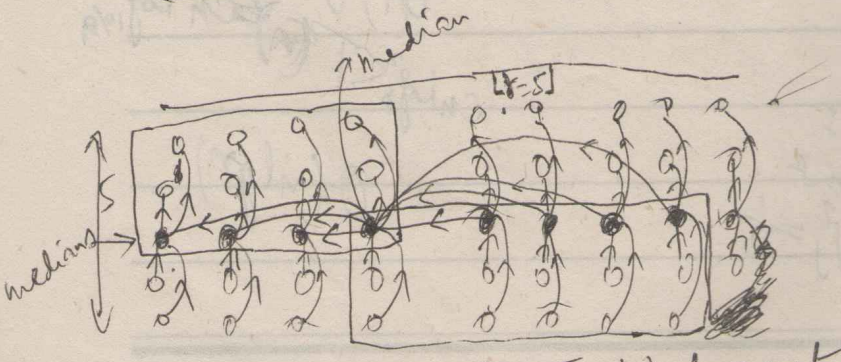
Ex.

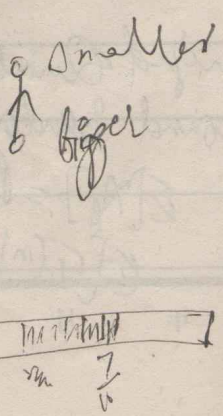
R.R. Lucky

$E[T(n)] =$ ..

worst-case linear time order statistics algo

Idea:

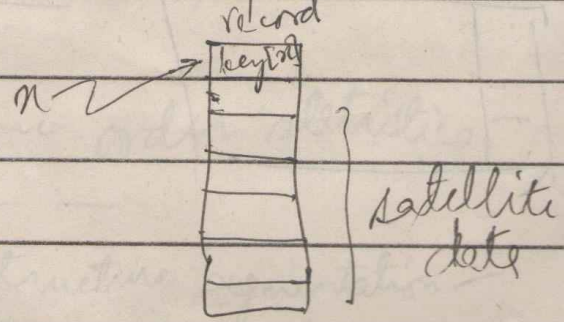




Analysis

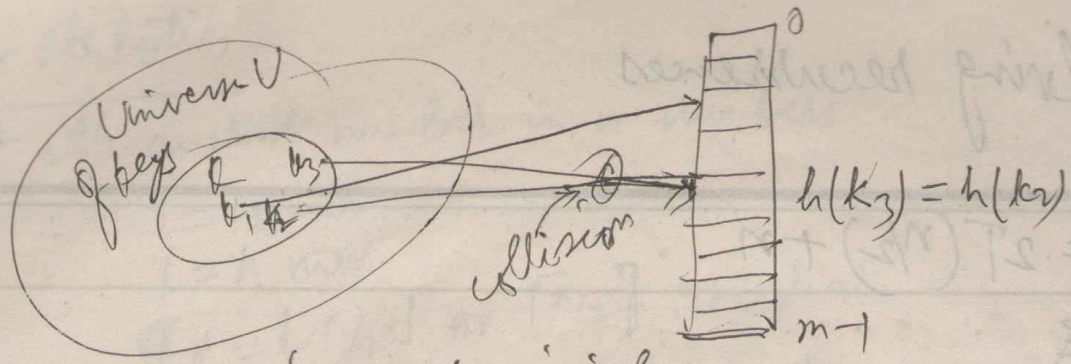
= Hashing =

symbol-table problem —



Operations =

- insert (T, x)
- delete (T, x)
- search (T, k)



Analysis of chaining =

choosing a hash function

division method:

multiplication method:

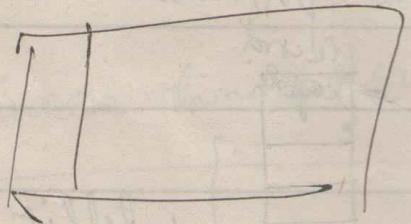
modular wheel

Open addressing

E [# probes]

= weakness of hashing =

universal hashing



perfect hashing:

- Binary Search Tree sort =

observation: BST sort ≡ Quicksort

- same comparisions different order

height of randomly built BST

Node depth

- Balanced-search trees =

- Binary-search-tree data structure supporting dynamic-set operations

ex AVL trees, B-tree/2-3 trees, red-black tree

= Red-black tree =

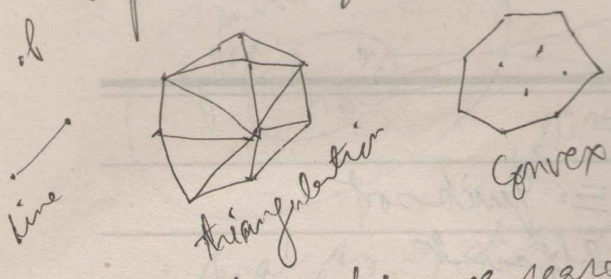
Rotation

= Dynamic order statistics -

- data structure augmentation -

Interval trees

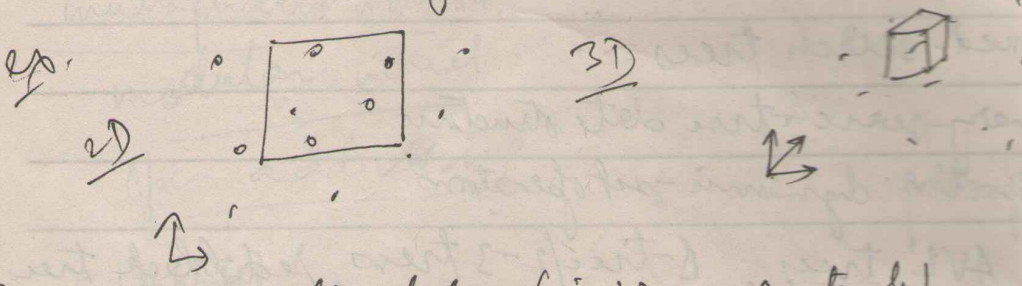
Computational geometry —



— orthogonal range searching —

n point in d dimension

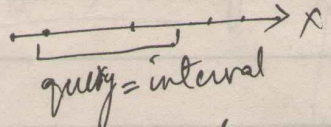
eg. representing database with n ~~dimension~~ records each with d fields



Query: axis-aligned box (in 2D a rectangle)

Goal: preprocess the points into a static data structure to support fast queries

1-D



① one approach: use interval trees with zero-length intervals

$x \rightarrow [x, x]$

→ dynamic DS

another: sort the points & binary search

→ static DS

idea: balance-search-tree

-binary-

sweep-algorithm

= Van Emde Boas =

- fixed-universe successor problem
maintain dynamic set S of size n of the universe $\mathcal{U} = \{0, 1, \dots, u-1\}$ of size u
subject to - insert ($x \in \mathcal{U} \setminus S$)
- delete ($x \in S$)
- successor ($x \in \mathcal{U}$): find next element $> x$
- predecessor ($x \in \mathcal{U}$)

solution: - link list
- balanced search tree

= Amortized analysis =
- accounting method
- potential method
table doubling -

= Dynamic programming =
brute-force algo.

= graphs (review) =
directed graph (digraph) $G = (V, E)$
un- "
"friends" algo.