Hi!

I'm Giulio Purgatorio, a Computer Science student from the "ICT Solutions Architect" curriculum.

These are my notes for the "Algorithm Engineering"'s course (2019-2020)

I took them while attending real-life lessons (the last semester before the Covid pandemic), so.. they're hand written! ¯_(ツ)_/¯

If these were somehow useful to you, consider giving it a star on Github!

https://github.com/GPurgatorio/Master-Computer-Science-ICT

Some pages are a bit off, but it's still somewhat readable and easily zoomable. Sorry about that!

Algorithm Engineering

16/09

Ricevimento: LUN 15:00 - 17:00

Model Computation: a model to count the cost of an algorithm.

Def Algorithm: an A. is a finite, definite, effective procedure with some output. (input :))

- finite → defined #steps && it must end
- definite → each step must be executed in a finite time (="scrivibile a penna") ⇓ related to programming languages.
- effective → steps must be very compact instructions ⇒ basic and efficient steps

⇐ while(TRUE); [..] is not an algorithm, it's a procedure

The issue with A **Resources** (minimize) — SPACE — TIME

Differently from ALG this time it will matter (GB + of data)

$T(n)$

today: energy compsumption, communication

time cost of alg. — input dim.

$T(n) = n$ [linear]

main idea: we don't evaluate real time

it's a function that models the time with #steps

ideally, if #steps increases ⇔ time increases

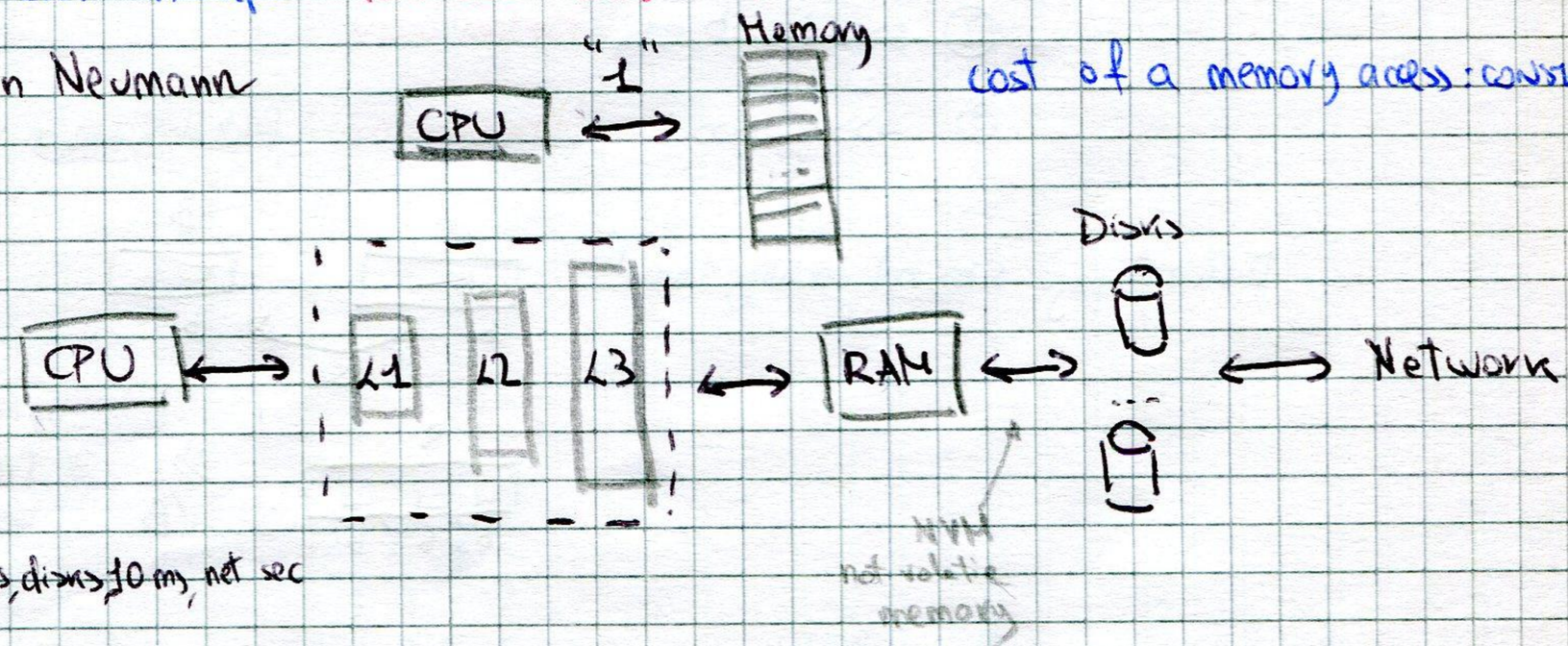
we want then from #steps ⟶ #some crucial steps (not considering constants) ⇒ $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$

⇒ Asymptotic behaviour $T(n)$, $n \to +\infty$

only the one matters ⟶ $n^3 + 1000n^2 + 10^9 n$ good model, not perfect (small n ⇒ RIP)

So, the assumptions:
- $T(n)$, $n \to +\infty$ GOOD
- $T(n) \simeq$ #steps NOT GOOD (ANYMORE)

Past computers ← Von Neumann



cost of a memory access: const

But Today

hierarchy of model levels

different access times!

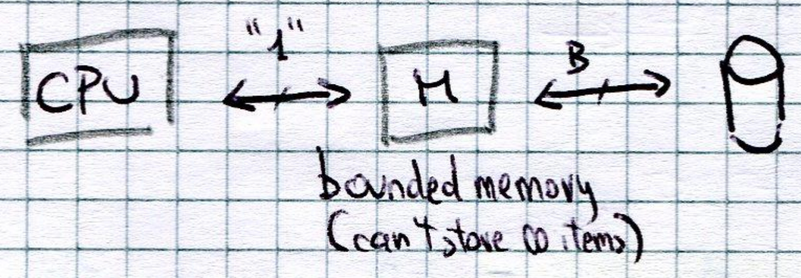
cache 10 nanos, RAM 100 nanos, disks 10 ms, net sec

if the data is not well organized in the memory, you've potentially losing 10 mlns!

2 alg. with same #steps but different layers ⇒ enormous differences!

We introduce a model to simplify calculus

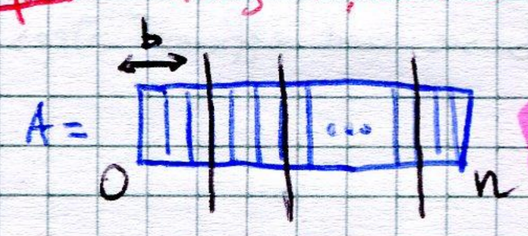
RAM → 2 level memory model (it can be applied in any point, depending on data size)



if the item isn't in Memory (cost 1) then we fetch an entire page (cost B) from disk.

read/write ≡ I/O ≅ 10^5-10^6 slower

<u>example</u> Array A, sum all items



Def Family of Algorithm: $A_{s,b}$

A is LOGICALLY divided in blocks of size b (e.g. 3)

⇒ # logical blocks = $\frac{n}{b}$ (assume for simplicity that $n\%b=0$)

s = STEP the jump, "algorithm works this way"

s=1 take 1st block, sum inside items, take 2nd block, ..

s=2 take 1st block, sum inside items, take 3rd block, ..

...

⇒ sum items in the block $s*i \% \frac{n}{b}$

→ not all s are good! (s=0 :())

we need to cover all blocks ⟺ s and $\frac{n}{b}$ must be COPRIME

⇓

to obtain a permutation $\{0,1,\dots,\frac{n}{b}-1\}$

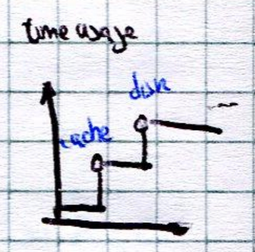
Lets assume n=15, b=3, $\frac{n}{b}=5$

$A_{2,3}$

still takes n sums

All the alg. of this type are EQUIVALENT (= literally the same)

but what if size of A → +∞? Time cost is totally different!

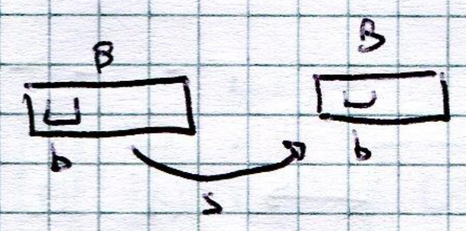


① s = b = 1 sequential scan 1 item at a time

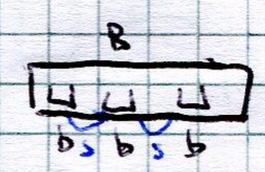
② s >> 1, b = 1 random access $2*i\%5 \Rightarrow 0,2,4,1,5$

③ s >> 1, b >> 1 block-wise access fetch block, sum, jump, ..

Different page sizes, not counting them!



b/B #I/Os



$3b/B$ # I/Os

We will assume that fetch page is identical on any page (not true, but who cares)

So our algorithm A — time complexity $\simeq$ #steps by asymptotic analysis
— I/O complexity $\simeq$ #disks I/Os — sequential
— random
— space

$n = (1+\varepsilon) M$

$$P(\varepsilon) = \frac{\varepsilon M}{n} = \frac{\varepsilon \cancel{M}}{(1+\varepsilon)\cancel{M}} = \frac{\varepsilon}{1+\varepsilon}$$

probability of I/O fault

time for computation

$\Rightarrow$ average cost of a step $= (1-a) * 1 + a *$ time for memory access

$$= (1-a) * 1 + a * [\underbrace{(1-P(\varepsilon)) * 1}_{\text{internal mem}} + \underbrace{P(\varepsilon) * C}_{\text{disk}}]$$

$C \geq 10^5$

$a \simeq 30\% \Rightarrow 0.7 * 1 + 0.3 * [\dots] \geq 3 * 10^4 * P(\varepsilon)$

$\Rightarrow$ if $P(\varepsilon) = 0.001$ $\quad 30\cancel{000} \cdot \frac{1}{\cancel{1000}} = 30$ times more than being in internal mem

very small percentages $\longrightarrow$ 0.01 $\quad 30\cancel{000} \cdot \frac{1}{\cancel{100}} = 300$ times

$P(\varepsilon)$ is called LOCALITY OF REFERENCE

spacial locality $\qquad$ temporal locality

17/09

it's dimension, not elements...

Example Array $A[1,n]$ SORTED already

Apply binsearch() → # I/Os ? (in 2 level MemMod)

M = memory size

B = page size

$$\log_2 n - \log_2 B = \log_2 \frac{n}{B}$$

	UPPER BOUND	LOWER BOUND
RAM	$\lg_2 n$	$\lg_2 n$
I/O	$\lg_2 \frac{n}{b}$	$\lg_{B+1} n+1$

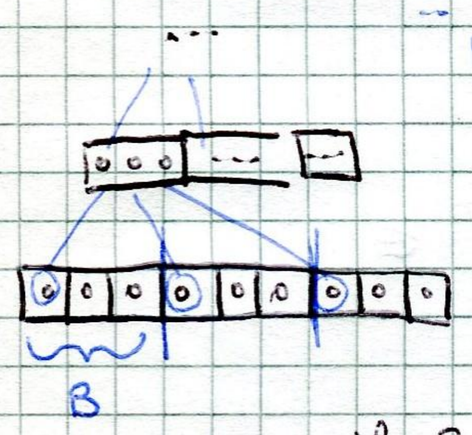
$$\lg_B n = \frac{\lg_2 n}{\lg_2 B}$$

$n = 2^{40}$ $B = 2^{12} = 4Kb$

CLASSIC $\lg_2 \frac{2^{40}}{2^{12}} = \lg_2 2^{28} = 28$ I/Os

OURS must be $\geq \frac{\lg_2 n}{\lg_2 B} = \frac{40}{12} = 3,\ldots \Rightarrow 4$ I/Os at least

what if we organize data? [B-TREE]



$\frac{n}{B^2}$ keys

+

$\frac{n}{B}$ keys

+

n keys

↑ continue until < B keys

if $B=2 \Rightarrow \Sigma \uparrow$ $n + \frac{n}{2} + \frac{n}{4} + \ldots \leq 2n$

this 2 is here because we are COPYING values

if we MOVE keys → n

⇒ REORGANIZE DATA!

Example

S = | 2 | 6 | 0 | 3 | 4 | 7 | 5 | 8 | 4 | 5 |

↑ buy ↑ sell

3 solutions $n^3 \rightarrow n^2 \rightarrow n$

S[i] = value at the end of the day

maximize profit buy = 2 sell = 0 ⇒ gain = -2 !!

⇓

(distance)

D = | ~~X~~ | 4 | -6 | 3 | 1 | 3 | -2 | 3 | -4 | 1 |

not valid

now just sum everything between Buy and sell to see gains

Example

$S = (\ell_1, \ell_2, \dots, \ell_n)$ and GIVEN $m > 0$

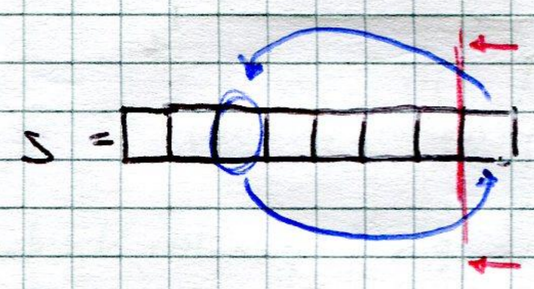
(RANDOM SAMPLING) sample uniformly at random m items out of S

1. S given in ADVANCE
2. S given in STREAMING → n known / → n unknown [Knuth]

in "stream" you don't kn the next ones

Assume:
- a procedure Rand() [actually Random]
- m positions are sorted of the sampled items

min $\{m, \frac{n}{B}\}$ I/Os depends on how much far



NOTE if dimension of data is fixed (int) you can jump

otherwise (string) use another array for positions

random position $[1,n]$

pick item, swap with last one

do not count last one next loop

```
s = 0, 1, ..., m-1
    p = Rand(1, n-s)
    swap(S[p], S[n-s])
```

swapping POINTERS, not ITEMS

for each sample I move data ⇒ m I/Os

PYING values

Dictionary

```
D = ∅
while |D| < m do
    p = Rand(1,n)
    if p ∉ D
        insert p in D
sort D
```

if I get a $p \in D$ then I "lose" a loop

NOT working on S

complexity? $P(\text{fault}_{p \in D}) = \frac{|D|}{n}$

assume $m \le \frac{n}{2}$ because if it gets higher just swap the problem!

$\Rightarrow \frac{m}{n} \le \frac{1}{2} = 50\%$

e gains

Birthday Paradox

```
D = ∅
while |D| < m do
    x = randomly draw m positions [1,n]
    sort x and eliminate duplicates
    D = x
```

⇒ on average, 2 rounds to fetch a ☺ ⇒ $2m$

Probs of duplicates? $P(\text{not dup}) = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-m-1)}{n^m}$ = # distinct config. / m positions in n ways $\forall$pos $n*n*n \ldots *n$

$= \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \ldots \cdot \frac{n-m+1}{n} = \left(1-\frac{1}{n}\right) \cdot \left(1-\frac{2}{n}\right) \cdot \ldots$

$\lim_{x \to 0} 1 - x = e^{-x}$

$\approx e^0 \cdot e^{-\frac{1}{n}} \cdot e^{-\frac{2}{n}} \cdot \ldots \cdot e^{-\frac{m-1}{n}} = e^{-\frac{1}{n}(1+2+\ldots+m-1)} = e^{-\frac{1}{n} \cdot \frac{m(m-1)}{2}}$

$e^{-\frac{m(m-1)}{2n}}$

18/03

We will now look at the **streaming model** items arrive one after the other

we can't buffer (≡ keep) items, gotta manage each one of them individually

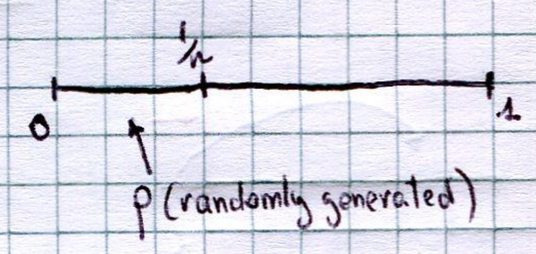
2 possible scenarios, n is either known or it's unknown

Conditions to guarantee: ① sample m items by the end

② uniform sampling every item must be sampled with prob. $\frac{m}{n}$

Assuming the Sequence $= i_1, i_2, i_3, \dots, i_j, \dots, i_n$ how to make the sampling process?

$m=1$, **n known** Prob. to sample item $i_1 = P(i_1) = \frac{1}{n}$



If I pick the item, I then have m items with uniform sampling.

If I fail to pick the item

if $p < \frac{1}{n}$ pick item, else do

$\Rightarrow P(i_2) = \frac{1}{n-1}$ uniform in the rest of the items

remember that to get this one, you HAVE failed $P(i_1)$

$P(i_3) = \frac{1}{n-2}$ etc.

$\Rightarrow P(i_j) = \frac{1}{n-j+1}$

at the end, I can't fail!

[for $m==1$, ① is guaranteed]

(FROM NOTES) **PSEUDOCODE**

```
s=0;
for (j=1; (j≤n) && (s<m); j++) do
        p= Rand(0,1);
        if( p < (m-s)/(n-j+1) ) then
            select S[j];
            s++;
        end if;
end for;
```

All of this implies that $P(i_r) = \frac{1}{n}$ $\forall r \le j$ $\overset{\text{INDUCTION}}{\Longrightarrow}$ prove that $P(i_j) = \frac{1}{n}$ (Condition ②)

$P(i_j) = P(\text{NOT picked } i_1, i_2, \dots, i_{j-1}) * P(i_j)$ to get to $P(i_j)$ I have failed all $j-1$

$$\underbrace{P(i_1) = \frac{1}{n} \quad P(i_2) = \frac{1}{n} \quad \dots \quad P(i_{j-1}) = \frac{1}{n}}_{P(\text{NOT pick all}) = \frac{j-1}{n}} \Rightarrow P(i_j) = \left(1 - \frac{j-1}{n}\right) * \frac{1}{\cancel{n-j+1}}$$

$$\overset{\shortparallel}{\frac{\cancel{n-j+1}}{n}} = \frac{1}{n}$$

OK ✓

NOTE! n "random" numbers to generate is costly!

pseudorandom generators cycle, so not good on big data!

$m > 1$

$$P(i_s) = \frac{\overbrace{m-s}^{\text{how many items I have to pick}}}{\underbrace{n-s+1}_{\#\text{items left}}}$$

s: # items picked (earlier, it was 0)

$$\overbrace{i_1, i_2, \dots, \underbrace{i_s, \dots, i_n}_{n-s+1}}^{\text{extract } m-s \text{ items from here}}$$

let's assume $m=2$

$i_1 = \frac{2}{n}$ ok for ①

- picked → $\frac{1}{n-1}$ prob. decreases
- not picked → $\frac{2}{n-1}$ prob. increases

NUM -- if item picked
DEN -- always.

⇒ if you don't get ANY item 'til m ⇒ 100% prob. to get each one

about ②

BASE $P(i_1) = \frac{m}{n}$ by def $[s=0, s=1]$ ✓

IND $P(i_s) = ($ s $(<m)$ are picked in $i_1, \dots, i_{s-1}$ AND i_s is picked$)$

if I took $s-1$ items in the first $s-1$ items

$$\overbrace{i_1, \dots, i_{s-1}}^{s-1 \text{ here}}, \underbrace{i_s, \dots, i_n}_{m-s-1}$$

"How many configs of $m-s$ items I can construct on $n-s+1$?"

$$P(i_s) = \frac{\binom{n-s}{m-s-1}}{\binom{n-s+1}{m-s}}$$

$$\binom{b}{a} = \frac{b!}{a!(b-a)!}$$

All possible configs, including (and not) i_s

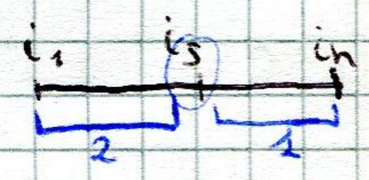
remove the ones that don't

$$\binom{n-s+\cancel{1}-\cancel{1}}{m-s-1}$$

ALGEBRA ⇒ $\frac{m}{n}$

assume $m=4$, $s=2$ ⇒ 2 more items and 10 items left

≡ 2 out of 10 && pick i_s



tot. configs that have 2 items $\binom{10}{2}$

$\binom{n-s}{m-s-1} = (i_s, _)$ 9 possible ways

NOTE Random numbers are considered as resources, gotta min-max!
To use less of these, "flip a coin" for HOW MANY to SKIP

⇒ # rand is way lower

Exercise

5 objects O_1, O_2, O_3, O_4, O_5 — Rand: O_1 3/4, O_2 1, O_3 3/4, O_4 1/4, O_5 1

Remember that it doesn't need to use ALL objects!

	O	Rand	$P = \frac{1}{n-s+1}$
×	O_1	3/4	< 1/5
×	O_2	1	< 1/4
×	O_3	3/4	< 1/3
✓	O_4	1/4	< 1/2
	O_5	1	DONE

what if n is unknown?

In any position it may stops, I have to guarantee ① and ②

use min-heap for $O(1)$ time Heap (H) of size m initialized with $i_1, \dots, i_m$

(obviously $n \geq m$)

every item has a priority $r_1, \dots, r_m \in [0,1]$

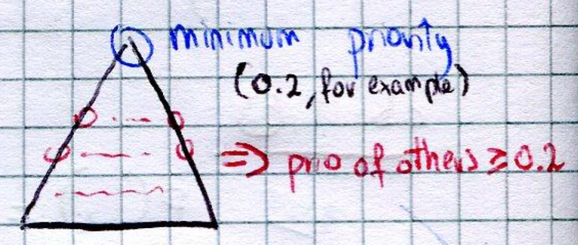
for each $s[j]$

pick $r_j = \text{rand}(0,1)$;

$y = \min$ prio of the heap

if $r_j > y \rightarrow$ delete obj with prio y && insert $(r_j, s[j])$

return H



can this be better? Time $= \log m$ Space $= m$

Reservoir sampling

$R[1,m] = s[1,m]$ initialize with the first m items

for each $s[j]$ do $j = m+1, m+2, \dots$

$h = \text{RandInt}(1, j)$

if $(h \leq m)$ $R[h] = s[j]$

return R

assume $m=2$

$R = \boxed{s_1 | s_2}$

s_3 arrives $\Rightarrow h = \text{RandInt}(1,3)$

IRREVOCABLE DECISION if $h = 1,2$ overwrite, else skip s_3

① ✓ obviously ② $P(i_j) = \frac{m}{n}$?

BASE if $m=n$ $\frac{m}{n} = 1$ ✓

IND if $n \geq m$, step $j > m$, $P(s[j]) = \frac{m}{j}$ concentrate on LAST item

assume $j=n$ $P(s[n]) = \frac{m}{n}$

so, $\underbrace{s_1, \dots, s_{n-1}}_{\text{and have ??}}, \underbrace{s_n}_{m/n}$

$\Rightarrow P(\text{picking } s[j], j<n) = P(s[j] \in R)$
- $s[j]$ in array and $s[n]$ NOT picked
- $s[j]$ in array, $s[n]$ picked in ANOTHER p

$P(s[j] \in R) = P(\underbrace{s[j] \in R_{n-1}}_{\text{"it was already in R"}}) \wedge (s[n] \text{ not sampled}) \vee P(s[j] \in R_{n-1}) \wedge (s[n] \text{ picked}) \wedge (s[j] \text{ ov}$

$\frac{m}{(n-1)} * \left(1 - \frac{m}{n}\right) + \frac{m}{(n-1)} * \frac{m}{n} * \frac{m-1}{m} \xRightarrow{\text{ALGEBRA}} \frac{m}{n}$ INDEPENDENT ON j

$\frac{n}{B}$ I/Os OPTIMAL

(assuming it can stay in memory)

Example O_1 O_2 O_3 O_4 O_5 $R = \boxed{O_1 | O_2}$

$h \leq m$ overwrite

$[1,3] \Rightarrow h=2$, O_3 $R = \boxed{O_1 | O_3}$

$[1,4] \Rightarrow h=3$ O_4 $R = \boxed{O_1 | O_3}$

$[1,5] \Rightarrow h=1$ O_5 $R = \boxed{O_5 | O_3}$

23/09

If you can get to an ordering problem :)

BUT following pointers we get $O(n)$ #I/Os ← huge difference!

Theoritally, with #items $= n$ and page size(disk) $= B$ → AT LEAST $\frac{n}{B}$

It can happen that we fetch an ENTIRE disk page and use ONLY ONE item!

⇒ List Ranking

Having a monodirectional list L, compute for each item $i \in L$ the DISTANCE (RANK) of i to the end of L

HOW WE SEE IT ex

$L \dashrightarrow (2) \rightarrow (5) \rightarrow (4) \rightarrow (6) \rightarrow (1) \rightarrow (3)$ ↺ t (TAIL)

HOW IT'S STORED

i	succ	rank
1	3	1
2	5	5
3	3	0
4	6	3
5	4	4
6	1	2

"(3) has distance = 0 from the end"

⇒ dist (head) $= n-1$

We introduce the P-RAM MODEL

It's the RAM ideal model with P processors

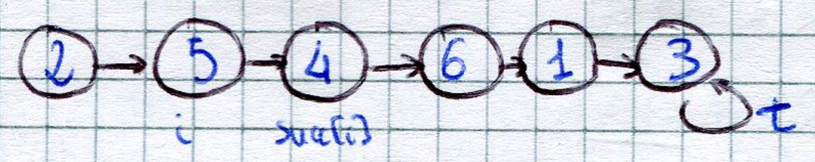
"if you can't solve it in the P-RAM you can't in the reality either"

Each processor can read ANY memory cell in parallel and they're synchronized

(MEMORY IS BOUNDED)

Algorithms that work on pointers on disk are HARD

Lets show how we use P-RAM $P = 6 = n$ every processor in charge of one i



i	succ	rank
1	3	1
2	5	1
3	3	0
4	6	1
5	4	1
6	1	1

3 → the Tail

POINTER JUMPING

every processor in parallel

① rank[t] = 0;
rank[i] = 1 $\forall i \notin t$

② • R[i] = R[i] + R[succ[i]]
• succ[i] = succ[succ[i]]

complexity (in P-RAM!)

① sum & assignments : takes succ[i] 1+1 = 2

② pointer jumping 5 → 4 → 6

keeping infos on how many items I've jumped (constan

In parallel all P_i ⇒

1	3	1+0 = 1
2	5	2
3	3	0
4	6	2
5	4	2
6	1	2

another jump? R[5] = 2 R[succ[5]] = 2 (6)

2+2 = 4

... 4+4 = 8 ... etc.

You double until you get to the end of the list

Every step takes constant time, but HOW MANY steps?

At step s I have 2^s length ⇒ $2^s \geq n$ ≡ $s = \log_2 n$

On P-RAM: Time = $O(\log n)$ Total #steps Work = $O(n \log n)$

[it gets to $O(n)$ turning off processor

What we've looking for is something in this form

A[a_i] op A[b_i] (both ① and ② have this form)

≡

* A[a_i] = A[a_i] op A[b_i]

example succ[succ[b_i]] (A, b_i)

How to implement this one?

SIMULATE ↖ $\forall i$

① SCAN away A, write <a_i, b_i, 0> storage container

"a_i (store) is destination address, b_i (fetch) the source address"

<1,3,0>
<2,5,0>
<3,3,0>
<4,6,0>
<5,4,0>
<6,1,0>

② SORT the Triples by 2nd component [≡ succ[i]]

<6, 1, ~~0~~ 1>
<1, 3, ~~0~~ 1>
<3, 3, 0>
<3, 4, ~~0~~ 1>
<2, 5, ~~0~~ 1>
<4, 6, ~~0~~ 1>

because I want to fetch the data in A[bi] WITHOUT jumping

Then we want to scan the Triples AND the table so that <6,1,0> stores 1

③ SCAN stores in 3rd component A[bi]

"repeat": A[_] → ... sumping "

⇒ SORT Triples R[i] = R[i] + R[succ[i]] (stored in a_i)

"In RAM sorting is bad, it's better to jump. In disks the opposite"

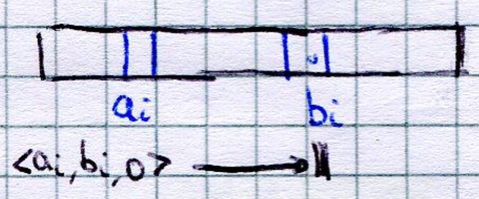
④ SORT Triples by first component [≡ i]

<1, 3, ~~0~~ 1> ma allora ce l'ha
<2, 5, ~~1~~ 2>
<3, 3, 0>
<4, 6, ~~1~~ 2>
<5, 4, ~~1~~ 2>
<6, 1, ~~1~~ 2>

⑤ SCAN Triples (in parallel with array) and execute

$A[a_i] = A[a_i] + \underbrace{A[b_i]}_{\text{3rd component}}$

sum to position:
(1) +0
(2) +1
(3) +0
(4,5,6) +1

So... A = 

for every Triple
create <ai, bi, 0> Triples, sort by bi (like putting the Triple "under" bi)
and take • value to store it in <ai, bi, A[bi]> (and put it "back")

COST?

①,③,⑤ scan Triples in array $O(\frac{n}{B})$ I/Os OPTIMAL

②,④ sort (n items, B disk page size, M memory) $= O(\frac{n}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B}) \Rightarrow \tilde{O}(\frac{n}{B})$

⇒ cost (2, stRanking)

$2 * \tilde{O}(\frac{n}{B}) + 3 * O(\frac{n}{B}) \Rightarrow$ L.R takes $\tilde{O}(\frac{n}{B} \cdot \log n)$ — don't remove it, TB → n=40 gets high

vs (worst case) $\Theta(n)$ I/Os

⇒ advantageous when $\tilde{O}(\frac{n}{B} \cdot \log n) < \Theta(n)$ ... so:

$\frac{n}{B} \cdot (\log_{\frac{M}{B}} \frac{n}{B}) \cdot \log_2 n < n$ $\quad (\log_{\frac{M}{B}} \frac{n}{B}) \cdot \log_2 n < B$

$n = 2^{40}$ $\quad \approx 2-3$ $\quad \approx 40$ $\quad 2^{12} - 2^{15}$ (4K – 32K)

HUGE GAP!

$\frac{\log_2 \frac{n}{B}}{\log_2 \frac{M}{B}} = \frac{\log_2 \frac{2^{40}}{2^{12}}}{\log_2 \frac{2^{34}}{2^{12}}} = \frac{28}{22} \simeq 1.2$ ⇒ sorting 1 TB will use 2 scans only (worst scenario, 4K = B)

we're not limited by algorithms but by computing! (DBMS !!)

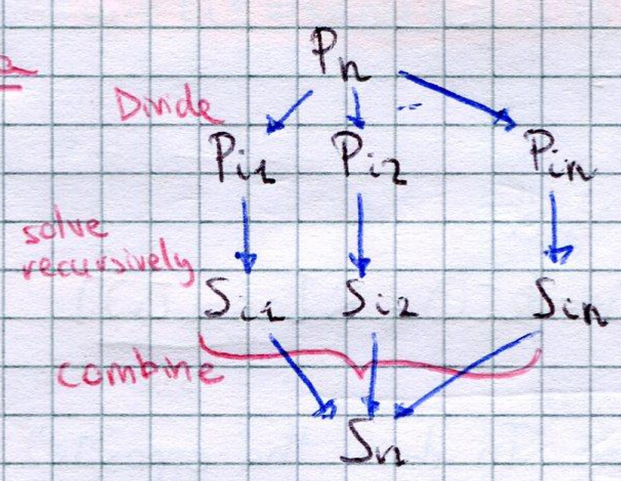
Theo A P-RAM algorithm taking T parallel steps over n processors with operations that have the form $A[a_i]$ op $A[b_i]$

can be simulated on disk taking $\tilde{O}(\frac{n}{B} \cdot T)$ I/Os

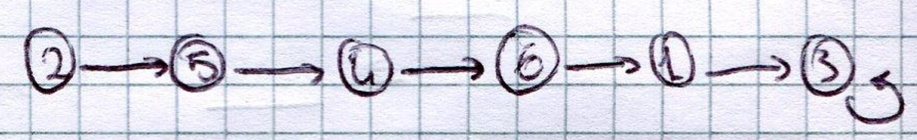
if $T < B$ this approach is advantageous

24/09

Dividi et Impera



D&I In the list ranking example...



Divide step: identify a subset of items that form an independence set I

INDEPENDENCE SET: if $\odot \in I \wedge \square \to \odot \to \square \Rightarrow \square \notin I$

In RAM model it's easy, take one, skip, take one, skip, ...

but in the disk ¯\(ツ)/¯

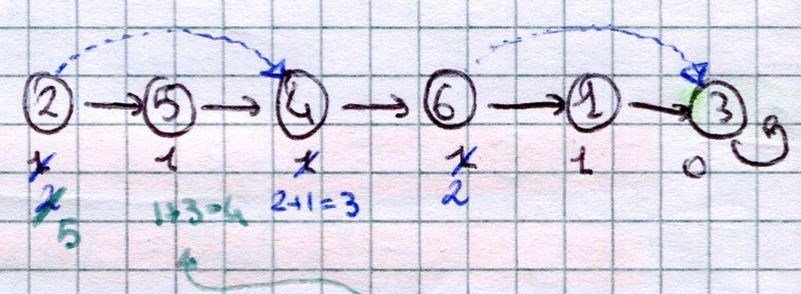
Lets assume nodes 5 and 1 as I

$\frac{n}{2} \geq |I| \geq \frac{n}{c}$ c constant ≥ 2

Drop I from L $\Rightarrow L^*$ $|L^*| \geq n - \frac{n}{c} = (1 - \frac{1}{c})n$

$\forall x.\ succ[x] \in I$ so 2 and 6

$\tilde{O}(\frac{n}{B})$ $\left[\begin{array}{l} Rank[x] += Rank[succ[x]] \\ Succ[x] = Succ[Succ[x]] \end{array}\right.$

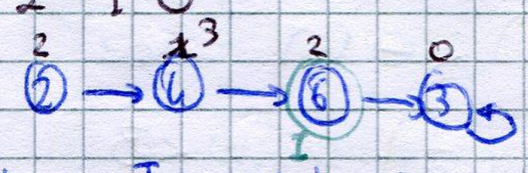


Recursively solve the ListRanking problem on L^*

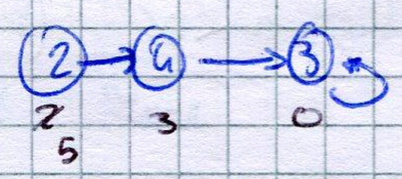
Reinsert I in L $Rank[x] = Rank[x] + Rank[succ[x]]$

REDO: Ist step 2 1 1 2 1 0

Recursive call on

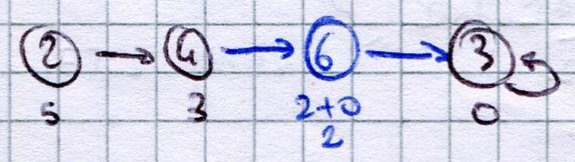


Reapply alg, get an I and drop it



Assume we own on a machine that can solve if length = 3

Go back, reinsert I in L^* (its rank = rank + rank(succ))



and so back

Complexity $T(n) = \underbrace{T\left(\left(1-\frac{1}{c}\right)n\right)}_{\text{cost of recursion}} + \underbrace{\tilde{O}\left(\frac{n}{B}\right)}_{\text{cost of splitting/reinserting}} + \text{cost}(\text{find } I)$ I/Os

$T(1) = O(1)$ "1 I/O"

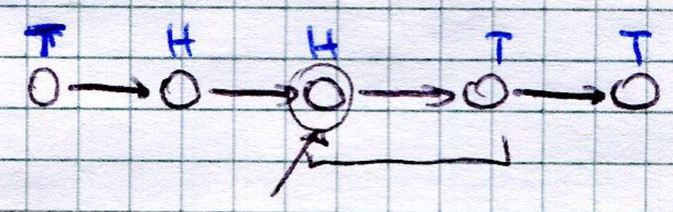
What's "REINSERTING"?

The opposite of the 2 operations [Rank+= succ =] $\Rightarrow$ it's $\tilde{O}\left(\frac{n}{B}\right)$ too

How to select an Ind. Set (I)?

Randomized Approach

$\forall$ item i I draw a coin (flip:) $\Downarrow$ $\{T,H\}$



$i \in I$ if $\text{coin}(i) = H \wedge \text{coin}(\text{succ}[i]) = T$ TH would also be ok

$\Rightarrow P(i \in I) = \underbrace{P(i = H)}_{1/2} \cdot \underbrace{P((i+1) = T)}_{1/2} = \frac{1}{4}$ $[c=4]$

on avg $T(n) = T\left(\left(1 - \frac{1}{4}\right)n\right) + \tilde{O}\left(\frac{n}{B}\right) = \tilde{O}\left(\frac{n}{B}\right)$ ($1 - \frac{1}{4} = \frac{3}{4}$)

$\equiv$ List Ranking equivalent to sorting

if you don't get $1/4$ of items?

Just repeat! :)

Deterministic approach

Assign to each item an integer $[0, n-1]$ $\text{coin}[i] = \{0, 1, \dots, n-1\}$

b is size of $\text{coin}[i]$ in bits $= \lceil \lg_2 n \rceil$

and $\text{bit}_b(i)$ = binary representation of $\text{coin}(i)$ in b bits

es get 6 coin values $\{0, \dots, 5\}$

$\forall$ item i compute position $\pi(i)$ where $\text{bit}_b(i) \wedge \text{bit}_b(\text{succ}[i])$ differ

and call $z(i)$ that bit value in $\text{bit}_b(i)$

$n = 8$

$\text{coin}(i) = 6$ 110 ($z(i)$)

$\text{coin}(\text{succ}[i]) = 4$ 100 ($\pi(i)$)

$\text{coin}(i) = \overbrace{2\pi(i) + z(i)}^{2(b-1)+1 = 2b-1}$ $\lceil \lg_2 (2b) \rceil$

$\Downarrow$

set the new $b = \lceil \lg b \rceil + 1$

PROB 2 values! same value

assume they do $\Leftrightarrow \text{coin}(i) = \text{coin}(\text{succ}(i))$

$\underbrace{2\pi(i)}_{0,2,4,\dots} + \underbrace{z(i)}_{0,1} = \underbrace{2\pi(\text{succ}[i])}_{0,2,4,\dots} + \underbrace{z(\text{succ}[i])}_{0,1}$ $\Rightarrow 2\pi(i) = 2\pi(\text{succ}[i]) \wedge$

$z(i) = z(\text{succ}[i])$ IMPOSSIBLE!

So the alg goes $n \to 2b \to \text{smaller} \to \dots$ ($\lg n$, $\lg\lg n$, $\lg\lg\lg n$ ..) reduce by exponential function

Def $\lg^* n$ i: $\lg^{(i)} n \to$ constant (1) see that $2^{65536} \to 5$ here so..

definitely no more than 5 iterations of the alg.

Get just 3 coin values $\{0,1,2\}$ from $\{0,1,\dots,5\}$

$\forall$ element i, if $coin(i) \in \{3,4,5\}$

$coin(i)$ picks smallest value in $\{0,1,2\} - \{coin(prec[i]), coin(succ[i])\}$

$$\underset{prec}{0} \to \overset{i}{3} \to \underset{succ}{0}$$

if $prec \wedge succ \in \{0,1,2\} \Rightarrow$ one is left free

else $\nearrow$

select the i that are local minimals

1 2 1 0 1

local minimal, pick it

so to have the smallest Ind. Set $\Rightarrow$ find largest distance between 2 minimals

$\underbrace{0\ 1\ 2\ 1}_{|I.S.| \geq n/4}\ 0$ ← not 2 or 1 or "prec" would be a minimal

cost? 1 sort + scan + 1 SS??

$\tilde{O}(\frac{n}{B}) * \cancel{\log^* n}$ constant

example

$\pi(i)$	1	2	1	0	2	1	2	0
$z(i)$	0	1	1	0	0	0	1	1
$2\pi(i)+z(i)$	2	5	3	0	4	2	5	1
	000	011	010	001	100	101	111	110

done with 1st item

#possible config = 8

we get 6 configs ?

if they differ in more positions, first pos they ($\longleftrightarrow$)

when you can assign 0 or 1, assign minimum

$$2 \to \underset{}{0} \to \underset{1}{\cancel{2}} \to 0 \to \underset{1}{\cancel{2}} \to 2 \to \underset{0}{\cancel{2}} \to 1$$

Ind. Set

SORTING ATOMIC ITEMS

	RAM	2LMM
Sorting	$\Theta(n \log n)$	$SORT(n,M,B)$
Permuting	$\Theta(n)$	$\min\{\underset{I/Os}{O(n)}, SORT(n,M,B)\}$

Permuting Problem: Given an array/sequence $S[1,n]$ and a permutation π over $\{1,2,\dots,n\}$

→ Permute S according to π

$S[1], S[2], \dots \to S[\pi(1)], S[\pi(2)], \dots$

$S = [A, C, B, E, D]$ $\pi = [3,1,5,4,2]$

$\to S = [B, A, D, E, C]$

stupid in RAM algorithm because it takes $\Theta(n)$ [copy for collisions: $S'[i] = S[\pi$

$\Rightarrow$ DISK

NOTE SORTING $\overset{\text{includes}}{\supseteq}$ PERMUTING

simulating the RAM alg. in disk $\Rightarrow O(n)$ I/Os !!

Remind of that previous algorithm, how do you permute?

$O(\frac{n}{B})$ **SCAN** π, create $<i, \pi(i), ->$ in the sequence

$<1, 3, ->$ store $s[3]$
$<2, 1, ->$ store $s[1]$
$<3, 5, ->$

SORT triples (II)

$<2, 1, A>$
$<5, 2, C>$
...

SCAN S ∧ triples, fill III ($s[\pi[i]]$)

$<i, \pi(i), s[\pi[i]]>$

SORT triples (I)

$<1, 3, B>$
$<2, 1, A>$
...

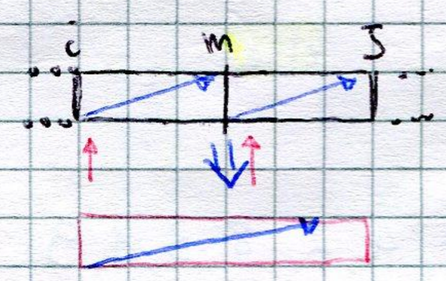
SCAN S ∧ triples, write in S the III

create records and pickup the records

2 sorts + 3 scans
LINEAR

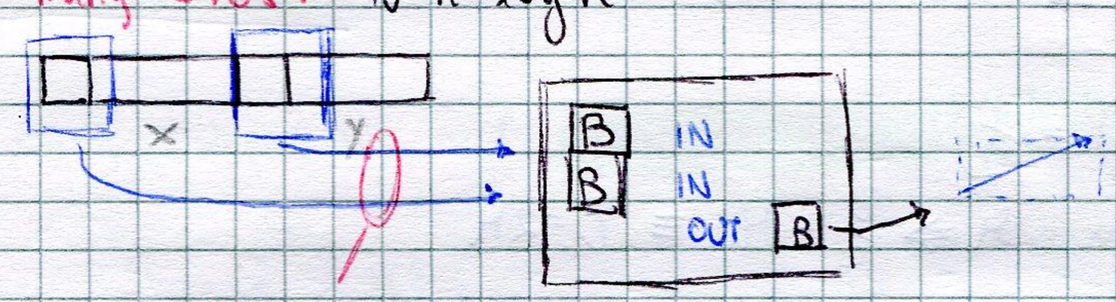
Merge Sort (s, i, j)

```
if (i < j)
    m = (i+j)/2          // divide
    MS (s, i, m-1)
    MS (s, m, j)
    Merge (s, i, m, j)   // recombine
```



2 pointers + additional array

How many I/Os? $\approx n \log n$



Buffering memory

- when I fill the block, write OUT, free block ↺ $\frac{n}{B}$
- when input block is exhausted, I fetch another one $\frac{n}{2}B \wedge \frac{n}{2}B$

$\frac{n}{B} \log_2 n$ we're not exploiting M! [table]

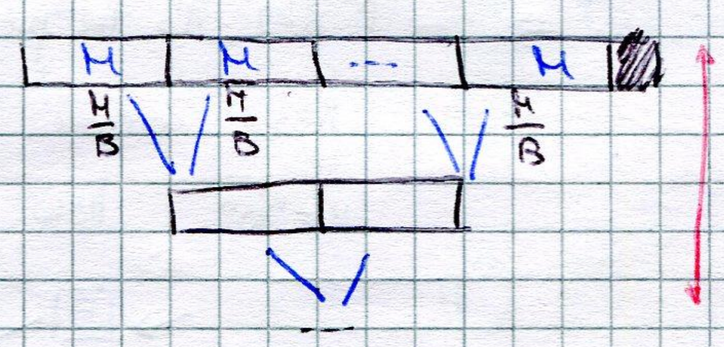
Make bigger buffers? NO, fetch $k*B$ pages ⇒ total cost is the same! [k fetches]

Fill the memory? pick the sequence and fetch as much as I can (M items)

$\frac{M}{B}$ to fetch, $\frac{M}{B}$ to write

I/Os = ?

$\frac{n}{M} \cdot \frac{M}{B} = \frac{n}{B}$



$\log_2 \frac{n}{M}$

$\left(\frac{n}{M} \text{ runs LEAVES}\right)$

to merge two = TOTAL LENGTH (x+y)

Every level cost $O(\frac{n}{B})$ this # times

with M items ⇒ not paying I/Os

$\Rightarrow O\left(\frac{n}{B} \log_2 \frac{n}{M}\right)$ I/Os
ALL PARAMETERS

more memory ≡ lower cost

STILL NOT OPTIMAL ($\log_2$ due to binary search)

Of course we can improve by buying more memory or by compressing data

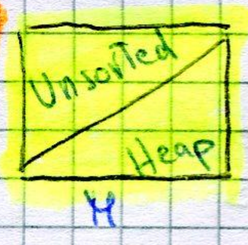
OR SNOWPLOW

on average $M \to 2M$ which would imply $O\left(\frac{n}{B} \log_2 \frac{n}{2M}\right)$

$$\overset{|||}{\frac{n}{B} \cdot \left[\log_2 \frac{n}{M} - 1\right]}$$

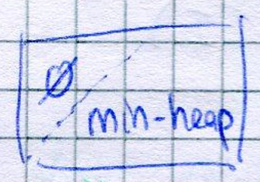
"In 1 TB of data, That -1 might mean ho

Idea



① fetch items → unsorted

② create (MIN) heap



③ fetch another item in the disk IF $H \neq \emptyset$

write min(H)

next = read next item on stream

if next < minimum insert next → U

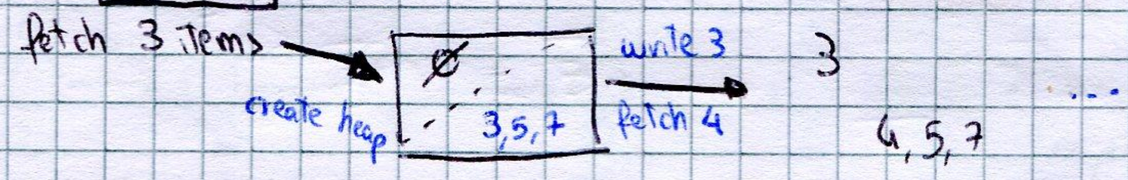
else insert next → H

Invariant $|H| + |U| = M$

start	M	∅
end	∅	M

intuition

$S = \underbrace{5, 3, 7}_{\text{fetch 3 items}}, 4, 2, 8, 3, 1, 9$



fetching τ items?

Lets assume that $\mathbb{P}(\text{next} \leq \text{minimum}) = \frac{1}{2}$

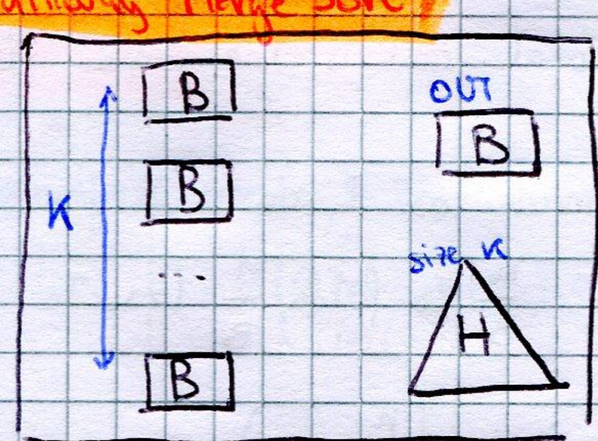
→ U $\frac{\tau}{2}$ items → max is $M \Rightarrow \frac{\tau}{2} = M \equiv \tau = 2M$

→ H $\frac{\tau}{2}$ items → written out

In total I've processed 3M items and 2M are in output (on avg.)

FINAL ALGORITHM

Multiway Merge Sort



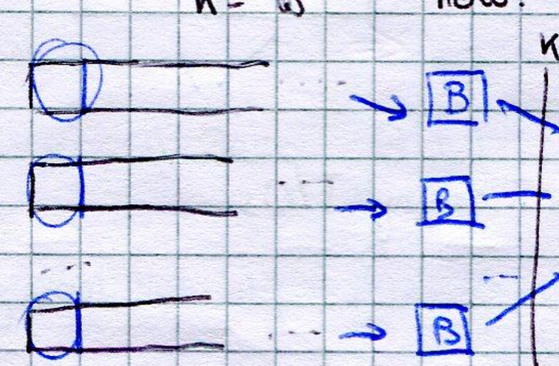
$(K+1)B = M \Rightarrow K = \frac{M}{B} \cancel{-1}$

instead of merging two / time, we merge K

if K=2 it's the previous algorithm

K=3 compare 3 items

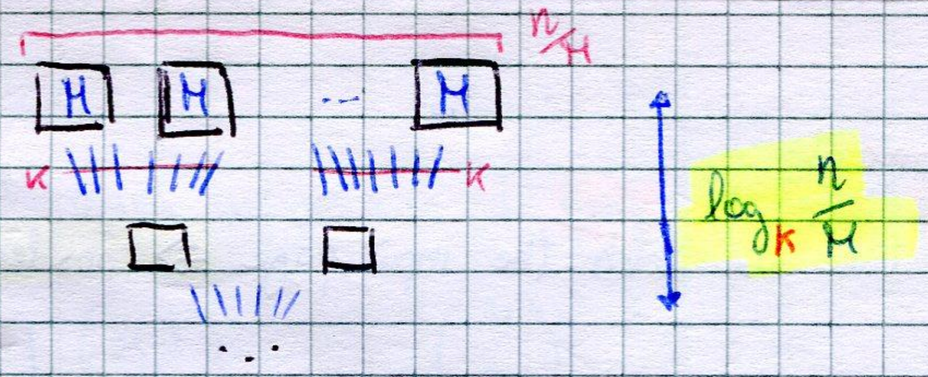
K = "..." how?



note H = <key, run>

because if I get a minimum I must know from where it came so I can get another item

Depth of the Tree?



every step costs $\frac{n}{B}$

Cost to merge?

$$\text{Sort}(n,M,B) = O\left(\overset{\text{scan}}{\frac{n}{B}} * \log_{\frac{M}{B}} \frac{n}{M}\right)$$

$$\frac{M}{B} = \frac{2^{34}}{2^{12}} = 2^{22} \text{ runs/merge}$$

30/09

So we said that the upper bounds are

	Perm	Sort
RAM	$\Theta(n)$	$\Theta(n \log n)$
DISK	$\min\begin{cases} n \\ \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B} \end{cases}$	$\frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}$

but **Lower Bounds?**

① **Sorting** in the RAM model we introduced decision trees ≡ comparisons between 2 items

$P_0 = 1$ At the beginning, the permutation is 1 (the input)

$P_1 = 2 * P_0$ After 1 comparison → 2 total perms

...

$P_t = 2^t \cdot P_0 = 2^t$

a:b

≤ / \ >

⇒ ordering the sequence in ↙ a:b or ↘ b:a

⇒ all the leaves are like this

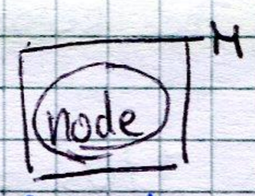
⇒ # leaves = # permutations = $n!$

How many steps? We must guarantee that $2^n > n!$ (otherwise I'm not generating some perms)

$2^t > n!$

$t \geq n \log_2 n$ comparisons

#I/Os? reading a disk page of B items



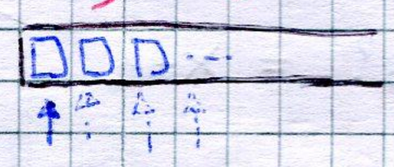
when we read the B items, these are fetched in the memories

⇒ **BEST:** M filled with items

≡ (M − B) ← items present; B ← items entering

in how many ways can I arrange these B items?

(2 items)



in the tree $\binom{2}{1}$ "In how many ways can I take 1 item out of 2"

here $\binom{M}{B}$

Now we introduce a new concept:

UNTOUCHED page: first time we fetch the input page ⇒ consider permutations of the item

$\binom{M}{B} * B!$ $b_1 b_2$ v $b_2 b_1$

TOUCHED page: perms already accounted

$\binom{M}{B}$

$U = P_{t-1}\left[\binom{M}{B} B!\right]$ $T = P_{t-1}\binom{M}{B}$

$\#U_s = \frac{n}{B}$ $[t = \#I/O_s]$

$\#T_s = t - \frac{n}{B}$ during the course of the algorithm

what the alg. is able to create

$$P_t = \binom{M}{B}^{t-\frac{n}{B}} \cdot \left[\binom{M}{B} \cdot B!\right]^{\frac{n}{B}}$$

$$= \binom{M}{B}^{t} \cdot (B!)^{\frac{n}{B}} \geq n!$$

[NOTE: $\log \binom{a}{b} \approx b \log_2 \frac{a}{b}$]

$\left[\frac{\log \frac{n}{B}}{\log \frac{M}{B}} = \log_{\frac{M}{B}} \frac{n}{B}\right]$

calcs $t > \frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}$

note we're NOT moving the items, just seeing what can creat

Can't improve the Multiway, unless

- items are divisible
- ∃ item that appears > 1 times

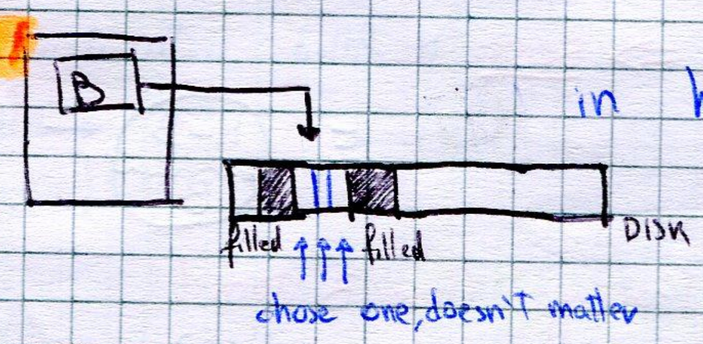
② perms

NOT only the READS, but also the WRITES

$P_0 = 1$ no I/Os ⇒ can't change the input ≡ input is the only perm.

$P_t = ?$

- write I/O



in how many ways can I write the page on

with X occupied pages ⇒ # stores B? = X+1

how many pages can my algorithm occupy?

if 5 steps ⇒ max 5 pages $n \Rightarrow n+1$?

I_w # write I/Os $\leq n+1$

$X+1$

t_r

- **Untouched** read I/O

In how many ways can I **bring** a new **page in memory**?

$$(B!) * \binom{M}{B} * \frac{n}{B}$$

all possible permutations — they can be shuffled in memory — take the page

- **Touched** read I/O

$$\binom{M}{B} * n$$

shufflings — max pages in n steps

$t = t_r + t_w$ so

$$P_t = \underset{(+1)}{n^{t_w}} \cdot \left[\binom{M}{B}(B!)\frac{n}{B}\right]^{n/B} \cdot \left[\binom{M}{B}\cdot n\right]^{t_r - \frac{n}{B}}$$

[$n^{t_w} \rightarrow$ every writing step increases #perms by n]

$$= n^{t_w} \cdot \left[\binom{M}{B}\frac{n}{B}\right]^{n/B} \cdot \left[\binom{M}{B}\cdot n\right]^{t_r - \frac{n}{B}} \cdot (B!)^{\frac{n}{B}}$$

$$\leq n^{t_w} \cdot \left[\binom{M}{B}\cdot n\right]^{t_r} (B!)^{n/B}$$

$$\leq \left[\binom{M}{B}\cdot n\right]^{\cancel{t_r+t_w}\ t} \cdot (B!)^{n/B}$$ has to be $\geq n!$

$$t \log_2 \left[\binom{M}{B}\cdot n\right] + \frac{n}{B}\log_2 (B!) \geq n \log n$$

$$t \log_2 \binom{M}{B} + t \log_2 n + \frac{n}{\cancel{B}}(\cancel{B}\log_2 B) \geq n \log n$$

$$t \cdot \left[B \log_2 \frac{M}{B} + \log_2 n\right] \geq n \log_2 n - n \log_2 B$$

$$t \geq \frac{n \log_2 \frac{n}{B}}{B \log_2 \frac{M}{B} + \log_2 n}$$

$$\equiv \min \begin{cases} n & ① \\ \frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B} & ② \end{cases}$$

dim?

DIM

① $B \log \frac{M}{B} > \log n$

$$\Rightarrow t > \frac{n \log \frac{n}{B}}{B \log \frac{M}{B}} = \text{SORT}$$

min

② $B \log \frac{M}{B} < \log n$

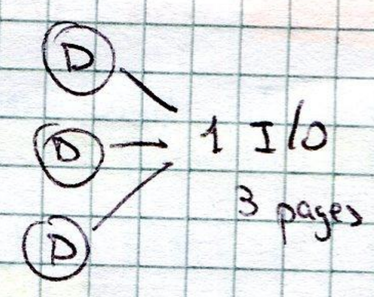
$$\Rightarrow t > \frac{n \cancel{\log \frac{n}{B}}}{\cancel{\log n}} = n$$

(asymptotically)

In practice: $\underset{4K}{B} > \frac{\log n}{\log \frac{M}{B}} = \log_{\frac{M}{B}} n$

What happens with D disks? (independents)

LowerBound sorting $\Omega\left(\frac{n}{DB} * \log_{\frac{M}{B}} \frac{n}{DB}\right)$



more disks ⇒ less I/Os

Notes only (not in exam)

greedSort, disk striping, lose independentness, RWHS+DS not effective (D → 1)

1/10

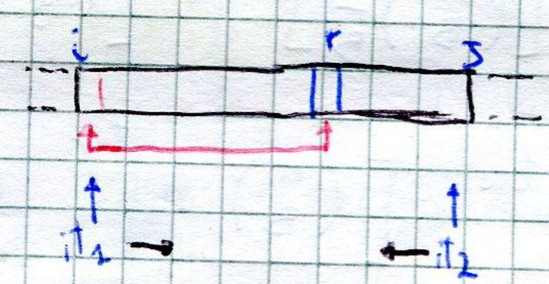
Distributed based Sorting (QuickSort)

MS	QS	D&C
	✓ partition	Divide
✓	✓	Recurse
✓ merge		Recombine

Partitioning → 2 way (II)
Partitioning → 3 way (III)

(II) QuickSort(S, i, j)

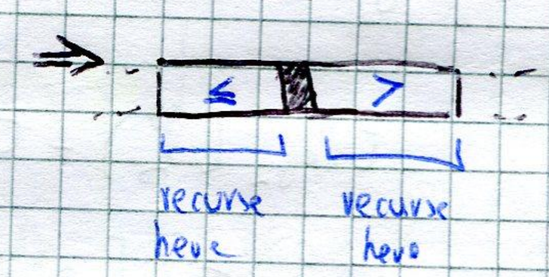
```
if (i<j) then
    r = pick "good" pivot
    swap S[r] with S[i]
    p = Partition(S, i, j)
    QS(S, i, p-1)
    QS(S, p+1, j)
```



at some point, the iterators are not satisfied anymore

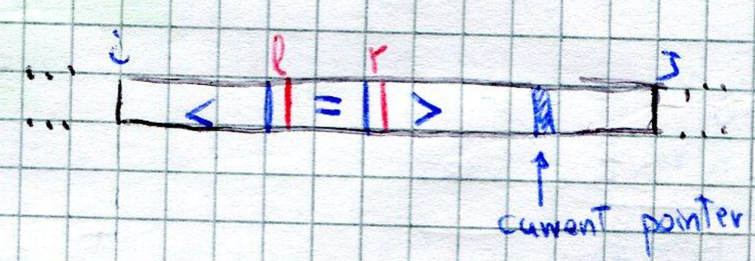
⇒ swap them and restart

untill they point to the same place



(III) Partition (S, i, j)

```
P = S[i]; l = i; r = i+1;
for (c = r; c ≤ j; c++) do
    if (S[c] == P) then
        swap S[c] with S[r]
        r++;
    else if (S[c] < P) then
        swap S[c] with S[l]
        swap S[c] with S[r]
        r++; l++;
return <l, r-1>
```



if pivot ∈ ≤ ⇒
- item > pivot → nop
- item = pivot → swap with r
- item < pivot → swap c, l; swap l, r; swap r, c; l++, r++

Theo (Con avg) The random selection of the pivot drives QS to compare at most $2n\log n$ items

DM

S array to be sorted Pick $S[u]$ and $S[v]$

Define $X_{u,v} = \begin{cases} 1 & \text{if } S[u] \text{ and } S[v] \text{ are compared} \\ 0 & \text{o/w} \end{cases}$

count (avg) #comparisons

$$= \mathbb{E}\left[\sum_{u<v} X_{u,v}\right] \quad \text{linearity of expectation}$$

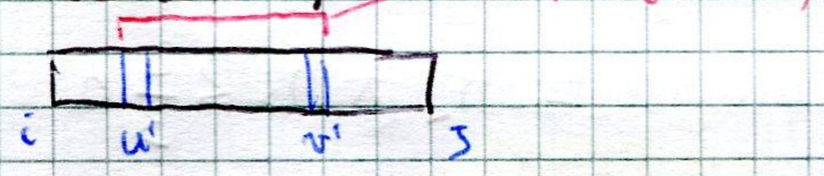
$$= \sum_{u<v} \mathbb{E}[X_{u,v}]$$

$$= \sum_{u<v} 1 \cdot \mathbb{P}(X_{u,v}=1) + 0 \cdot \mathbb{P}(X_{u,v}=0)$$

$$= \sum_{u<v} \mathbb{P}(X_{u,v}=1)$$

$$= \sum_{u<v} \mathbb{P}(u \text{ or } v \text{ is the pivot})$$

Imagine S' (S sorted)



now if pivot is $<$ (or $>$) than both

$\Rightarrow S[u]$ & $S[v]$ will be in the same partition

else (pivot $\in v'-u'+1$) $\rightarrow$ ① pivot is u or v ok

$\rightarrow$ ② pivot is in the middle

$S[u]$ to the left, $S[v]$ to the right, <u>NEVER</u> compared

Given the 2 situations, $\mathbb{P}(①) = ?$

$$= \sum_{u'<v'} \frac{2}{\underbrace{v'-u'+1}_{=k}} \quad ② \text{ u or v}$$

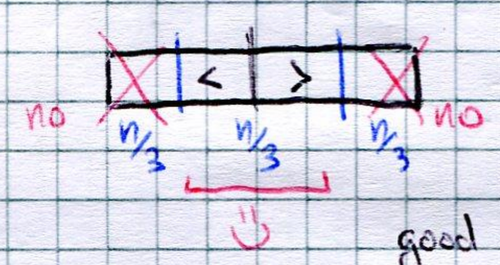
$$\sum_{u'=1}^{n} \sum_{v'=u'+1}^{n} \frac{2}{k}$$

$$2 \cdot \sum_{u'=1}^{n} \cdot \sum_{\substack{v'=u'+1 \\ k=2}}^{n-k+1} \frac{1}{k} \leq 2 \cdot \underbrace{\sum_{u'=1}^{n}}_{n \text{ (doesn't appear)}} \cdot \overbrace{\sum_{k=1}^{n} \frac{1}{k}}^{\text{armonic sum}} \leq 2n \cdot (\log n + 1)$$

Random Select

```
RS (S, k)
    v = random pos (1, n)
    S<, S=, S>                    S< = items smaller than S[v]
    n< = |S<|, n= = |S=|
    if (k ≤ n<)                   the item I'm searching is in [<]
        return RS (S<, k)
    if (k ≤ n< + n=)              //  [=]
        return S[v]
    return RS (S>, k - n< - n=)
```

$\hat{T}(n) \le O(n) + ?$

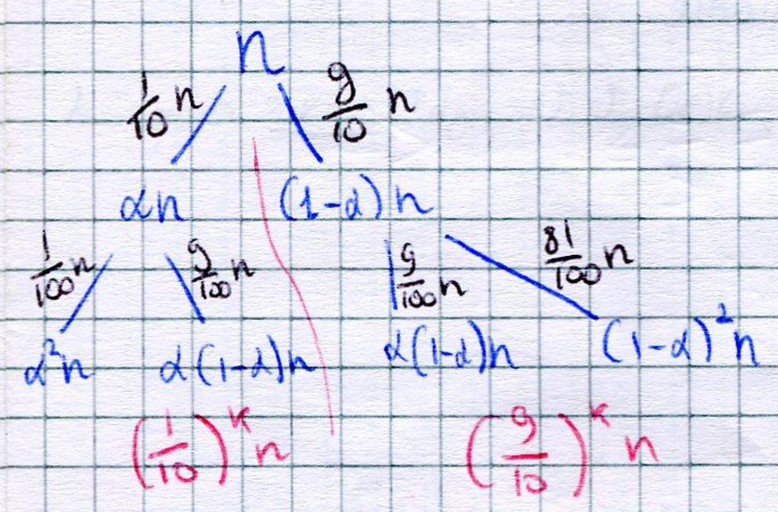


good partition $\equiv n_< \wedge n_> \le \frac{2}{3} n$

$\Rightarrow \dots + \frac{1}{3}\hat{T}\left(\frac{2}{3}n\right) + \frac{2}{3}T(n-1)$

$\frac{1}{3}\hat{T}(n) \le O(n) + \frac{1}{3}\hat{T}\left(\frac{2}{3}n\right) = O(n)$

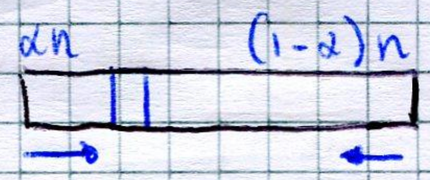
master theorem



small $\alpha \Rightarrow$ right branch is HUGE !!

$(1-\alpha)^k \le \frac{1}{n}$

$k \le \frac{\log_2 \frac{1}{n}}{\log_2 (1-\alpha)} \ge \frac{\log_2 n}{\log_2 \frac{1}{1-\alpha}}$ (negative) (negative)



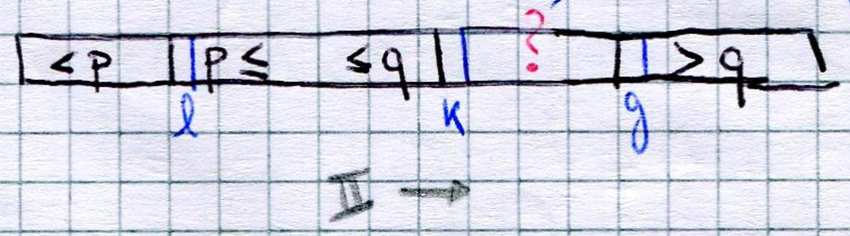
- αn swaps
- n comparisons
- $\frac{\log_2 n}{\log_2 \frac{1}{1-\alpha}}$ recursive steps

small α → less swaps / more rec steps

What should we do?

DUAL PIVOT

$p < q$




```
if s[k] < p
else if s[k] > q
        s[k] ↔ s[g-1]
else
        enlarge II
```

Balance
- misbranch prediction
- size parts
- #swaps

} on avg

$1.9\, n \log n$ comparisons $< 2 n \log n$ [qsort]

0.6 swaps (more than qsort) but swaps less costly we use th

2/10

In quicksort every time we make a recursive call, we allocate extra space.
In insertionsort this doesn't happen. Removal of tail recursion.

Bounded QS (S, i, j)

while $j - i > n_0$ do — n_0 depends on the architecture

r = pick a "good" pivot

swap s[r] with s[i]

p = Partition (S, i, j)

RECURSE ONLY IN THE "SMALL" PART! ⇒ at most $\frac{n}{2}$

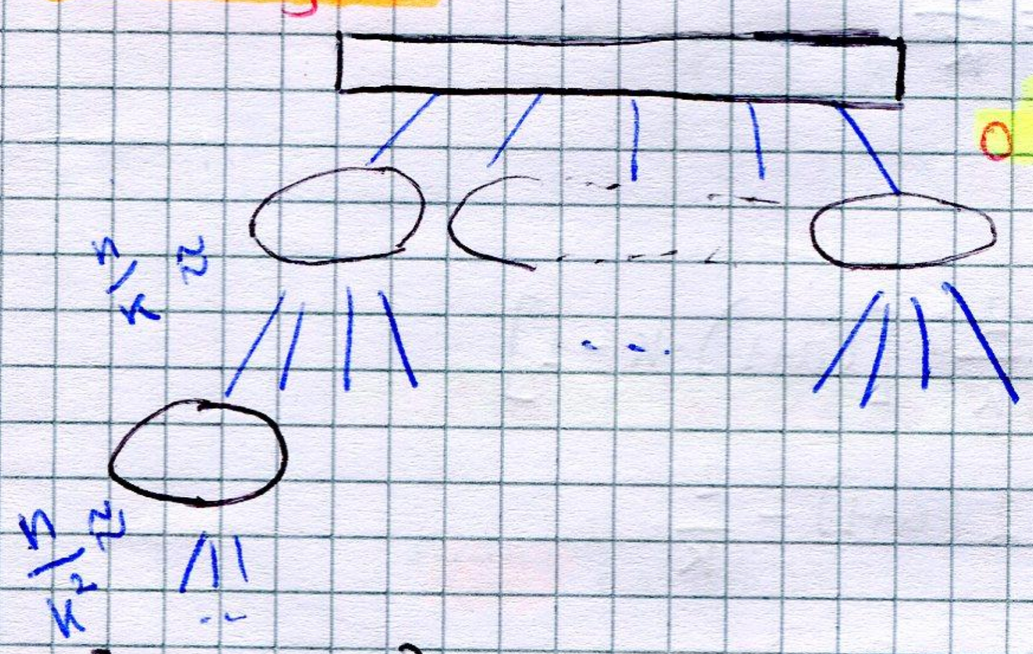
if $p \le \frac{i+j}{2}$ → left part is small — BoundedQS (S, i, p-1); i = p+1

else — BoundedQS (S, p+1, j); j = p-1

InsertionSort (S, i, j)

⇒ extraspace: $O(\log n)$ — not okay

MultiWay QS



$O\left(\frac{M}{B}\right)$ parts, $\frac{M}{B} = k$ ⇒ k-1 pivots — not 2 anymore :)

If partition is balanced (strong assumption!) the depth I is $\log_k n$

⇒ At level i: $\frac{n}{k^i} < 1 \iff k^i \ge n \iff i \log_2 k > \log_2 n \iff$
$\iff i > \frac{\log_2 n}{\log_2 k} \iff i > \log_k n$

But how to have k-1 good pivots? so that we get that sweet $\approx \frac{n}{k}$ with high prob [w. h. p.]

1. Take $(a+1) \cdot k - 1$ random pivots from S
2. Sort them and create a sequence A (ordered)
3. for $i = 1, 2, \dots, k-1$ pick pivot $s_i = A[(a+1) \cdot i]$
4. return the pivots s_i

⇒ we pick 1 every $n(a+1)$ items in the ordered sequence
≅ pick a number of items that represent a distribution of the original items
"In some sense, I'm balancing the parts: a elems before, a after"

a hyperparameter: $a = 0 \Rightarrow k-1$ items

need some balance! $a > 0 \Rightarrow (a+1) \cdot k - 1 \longrightarrow n$, all array

Lemma

Let $k \geq 2$ and $a+1 = 12 \ln k$.

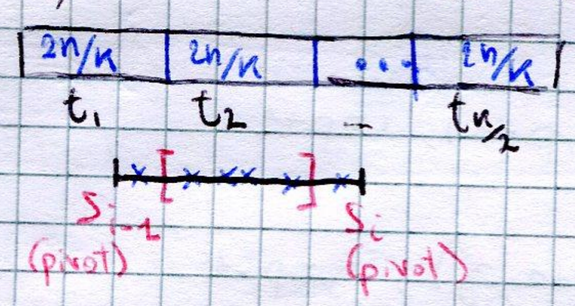
A sample of size $(a+1)^* k - 1$ GUARANTEES that every bucket B_i has size $< \frac{4n}{k}$ items, with probability $> \frac{1}{2}$

$\Rightarrow$ w.h.p. Just repeat Π logarithmic times to get h.p.

DIM

$$\mathbb{P}\left(\exists i : |B_i| \geq \frac{4n}{k}\right) \leq \frac{1}{2}$$

For simplicity, consider S' ($\equiv S$ sorted) divided in blocks.



if the $|\text{bucket}| \geq \frac{4n}{k} \Rightarrow$ it will take at least one block $\left(\frac{2n}{k}\right)$ because it has to be twice of it

$$\Rightarrow \mathbb{P}\left(\exists i : |B_i| \geq \frac{4n}{k}\right) \leq \mathbb{P}(\exists t_j : t_j \text{ contains} < a+1 \text{ oversampled pivots})$$

$$\leq \frac{k}{2} \cdot \mathbb{P}(\text{specific block } t_j \text{ ''})$$

$$\mathbb{P}(\text{random pivot is picked from } t_j) = \frac{\frac{2n}{k}}{n} = \frac{2}{k}$$

$[X = \#\text{pivots picked from } t_j]$

$$\frac{k}{2} \cdot \mathbb{P}(X < a+1) \xrightarrow[\text{avg}]{} \mathbb{E}[X] = \frac{2}{k} \cdot [\overbrace{(a+1)\cdot k}^{\#\text{Items}} - 1]$$

$$= 2(a+1) - \frac{2}{k} \quad \text{decreases!}$$

$$\geq 2(a+1) - 1$$

$$\Rightarrow a+1 \leq \frac{2}{3}\mathbb{E}[X] = \left(1 - \frac{1}{3}\right)\mathbb{E}[X]$$

$$\overset{k \geq 2}{\geq} \frac{3}{2}(a+1) \quad \text{true } \forall a \geq 1$$

Chernoff Bound

$$\mathbb{P}(X) \leq (1-\delta)\mathbb{E}[X] \leq e^{-\frac{\delta^2}{2}\mathbb{E}[X]}$$

"The prob. that X is far from average of $(1-\delta)$ is something exponentially slow"

$$\frac{k}{2}\mathbb{P}(X < a+1) \leq \frac{k}{2} \cdot \mathbb{P}\left(X < \underbrace{\left(1 - \frac{1}{3}\right)}_{\delta}\mathbb{E}[X]\right)$$

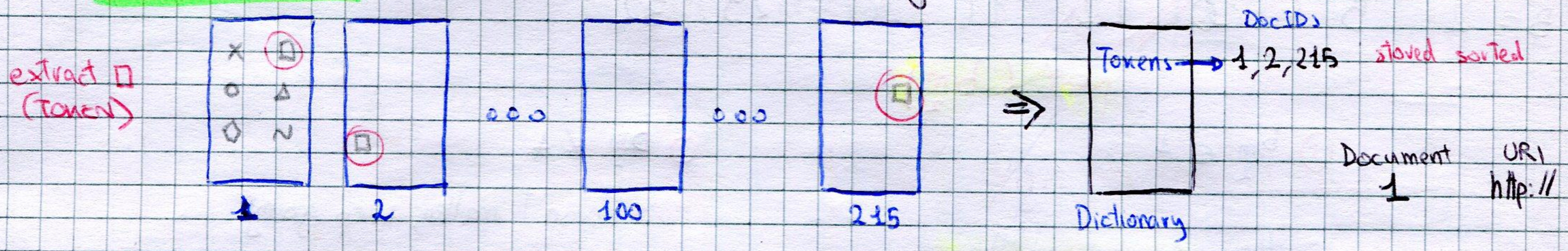
$$\leq \frac{k}{2} \cdot e^{-\frac{1}{18} \cdot \mathbb{E}[X]} \quad \leftarrow \frac{1}{9 \cdot 2}$$

$$\leq \frac{k}{2} e^{-\frac{1}{18}\left[\frac{3}{2}(a+1)\right]} = \frac{k}{2} e^{-\frac{a+1}{12}} = \frac{k}{2} \cdot e^{-\frac{\cancel{12} \ln k}{\cancel{12}}} = \frac{\cancel{k}}{2} \cdot \frac{1}{\cancel{k}} = \frac{1}{2}$$

7/10

Set Intersection

Database & Search Engines



Query. $Q = w_1 w_2$ 2 words

to find all docs where $w_1 \wedge w_2$ appear

A $w_1 \rightarrow \ldots 100 \ldots$

B $w_2 \rightarrow \ldots 100 \ldots$

contained in both $\Rightarrow A \cap B$

Sorted Set Intersection

Given 2 sorted sequence of integers $A = a_1, a_2, \ldots a_n$ $B = b_1, b_2, \ldots, b_m$

we will assume always that $m \leq n$

sol. ① Merge-based Solution

$T: O(n+m)$ $S: O\left(\frac{n+m}{B}\right)$ I/Os

[like Pearson Correlation in the apprenticeship]

1 3 4 7 || 2 4 5 6

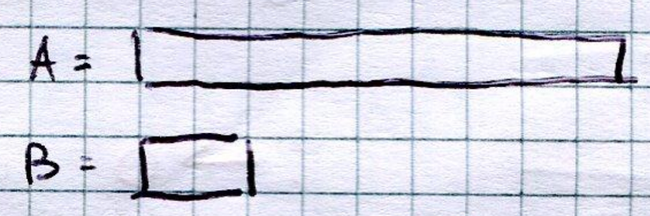
1 2 3 4 4 5 6 7

4

OPTIMAL if $m = \Theta(n)$

what if $|m| << >> |n|$? 'cause ① sucks in that case

② we are in a situation like



Let's assume that $B \geq A \Rightarrow A \cap B = B$

LowerBound $= \Omega(\log_2 \#\text{solutions})$

all the sols are all the ways: m items in n items

$= \Omega\left(\log_2 \binom{n}{m}\right) = \Omega\left(m \log_2 \frac{n}{m}\right)$ num. of comparisons (≡ time in RAM model)

NOTE if $m \rightarrow n$ $\Omega(m) = \Omega(n) \Rightarrow$ merge

First proposed algorithm: **Mutual Intersection Algorithm**

There is an inefficacy in the bin search approach, when we search item b_i in A we possibly recheck over and over the same elements in A.

$\Rightarrow$ useless comparisons

A = 1 3 7 12 16 19 20 25 27 40 50 100

B = 2 5 6 10 12 16 20 29 32

pivot ($b_{\frac{m}{2}}$)

① $b_{\frac{m}{2}} \in A$

then recurse on the smaller parts

② $b_{\frac{m}{2}} \notin A$

doesn't matter, keep going

$$A \cap B \Rightarrow \left(A[1,s] \cap B[1,\tfrac{m}{2}]\right) \left(A[s+1,n] \cap B[\tfrac{m}{2}+1,m]\right)$$

note that A is smaller now ⇒ take ③

when even #elems, $\lfloor \frac{n}{2} \rfloor$

"In the exam if length is the same, state which one you get"

1 3 7

2 5 6 7

16 19 20 25 27 40 50 100

16 20 25 32

what if $B(\frac{m}{2}) < A[0]$? less than every item in A

DROP $\frac{m}{2}$ set of B

⇒ complexity $O\left(\underbrace{(\log n)}_{\text{cost 1 call}} \underbrace{(\log m)}_{\text{\#calls}}\right)$

$$T(n,m) = O(\log n) + T(?, \tfrac{m}{2}) + T(?, \tfrac{m}{2})$$

depends on pivot

$? \to 0$ we happy :)

worst case : $? \to \frac{n}{2}$

OPPOSITE SITUATION OF QUICKSORT!

$$= O(\log n) + 2\left(T(\tfrac{n}{2}, \tfrac{m}{2})\right)$$

$$= O\left(m \log \tfrac{n}{m}\right)$$

optimal in terms of com

SCHEME	TIME	
MERGE	$n+m$	
BINSEARCH	$m \log_2 n$	smaller of $(n+m)$ if $m(\log n - 1) < n \Leftrightarrow m < \frac{n}{\log n - 1} \Leftrightarrow m < O(\frac{n}{\log n}$
MUTUAL	$m \log \frac{n}{m}$	
DOUBLING	$m \log \frac{n}{m}$	next (no recursion)

DOUBLING b_1 compared with: $a_1 \to a_2 \to a_4 \to \dots \to a_{2^k}$

until $a_{2^i} > b_1$

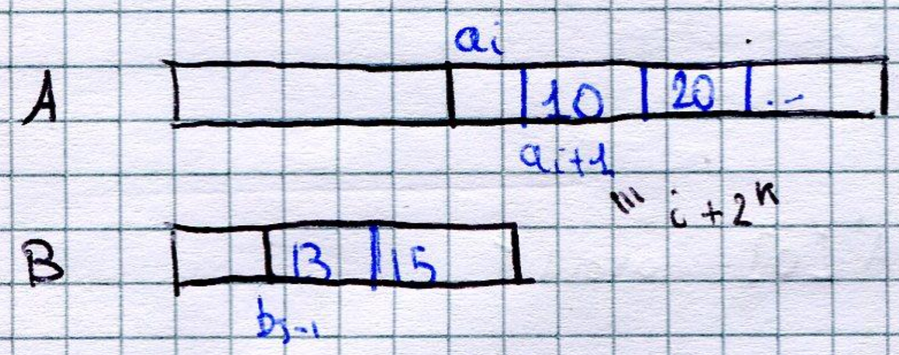
31 → 5 6 ~~10~~ 21 … 93

must be here (if present)

$\Rightarrow$ 4 items $\Rightarrow$ discard first 4 items

assume you find 31 there 21 31 [× × 93

restart from here



$a_i \le b_{j-1} < a_{i+1}$

keep going until $A[i+2^j] < b_j$ $\quad j{+}{+}$

at some point $b_j < A[i+2^k]$

$\Rightarrow A[i+2^{k-1}] < b_j$

$$2^{k-1} \le \underbrace{i' - i}_{i_j - i_{j-1}} < 2^k$$

Δ_j = from a_i to 2^k ("0") it's "stupid" but it's just one more compar

$$2^{k-1} \le i' - i < 2^k = \Delta_j$$

$\Rightarrow$ $k+1$ doubling steps $O(1)$
$\log_2 \Delta_j$ bin Search $O(k)$ } $O(k)$

For each step $\to O(\log_2 \Delta_j)$ $\forall$ key $\quad$ cost $\sum_{j=1}^{m} O(\log_2 \Delta_j)$

estimate $\sum_{j=1}^{m} \Delta_j$

$$2^{k-1} = \frac{\Delta_j}{2} \le i_j - i_{j-1}$$

$$\Delta_j \le 2(i_j - i_{j-1}) \quad \text{so} \quad \sum \Delta \le 2 \sum_{j=1}^{m} (i_j - i_{j-1})$$

TELESCOPIC SUM

$i_1 - i_0$
$i_2 - i_1$
…
$i_m - i_{m-1}$

$\Rightarrow 2(i_m - \cancel{i_0})$

$\le 2 i_m$

= last item of B

$\le 2n$

$$\sum_{j=1}^{m} O(\log_2 \Delta_j) = O\left(\sum_{j=1}^{m} \log_2 \Delta_j\right)$$

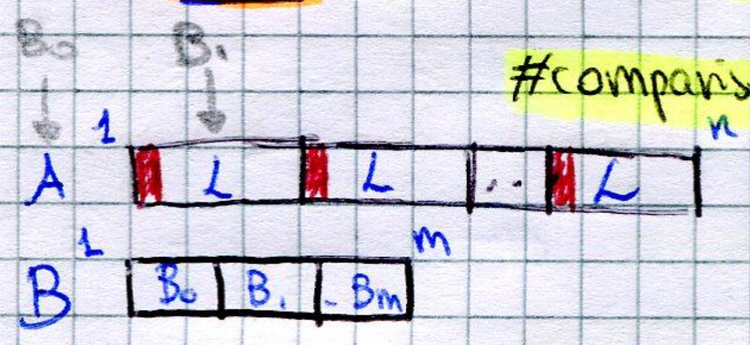
$$\underset{\text{Jensen inequality}}{\le} O\left(m \cdot \log_2 \frac{\sum \Delta_j}{m}\right) = O\left(m \log_2 \frac{\overset{\text{constant}}{\cancel{2n}}}{m}\right)$$

8/10

Today we'll see a not optimal algorithm but actually faster because it's exploiting the structure of m

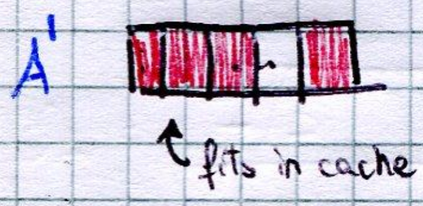
Idea: use the items in the cache to SELECT items in Memory

#comparison will be larger, but the cache is faster!



fix $L > 0$ and partition A in blocks of size L

$\#\text{blocks} = \frac{n}{L}$



take 1st elem $\forall$ block and copy it in A'

$|A'| = \frac{n}{L}$

① MERGE A' and B

$\Rightarrow$ sort $A' \cup B$ if $B_i < A'[0]$ avoid it

$\Rightarrow \frac{n}{L} + m$ LINEAR

② Assume B_1 goes in first block (A_1), MERGE these blocks

$(B_1 \cap A_{i_1})$

$(B_2 \cap A_{i_2})$

...

$(B_m \cap A_{i_m})$

$$\Rightarrow \underbrace{\sum_h |B_h|}_{m} + \overbrace{|A_{i_h}|}^{L} \leq m + m \cdot L$$

$\Rightarrow$ Total complexity: $O\left(\frac{n}{L} + \cancel{m}^{\text{asympt.}} + m \cdot L\right)$

Increase $L \Rightarrow$ cost of scanning reduces but cost of merge incre

TRADEOFF

#I/Os ① $\frac{\left(\frac{n}{L} + m\right)}{B}$

+

② $\frac{m}{B} + m\left[1 + \frac{L}{B}\right]$

Delta (Δ) - coding

Store the first number explicitely, next ones just the differences

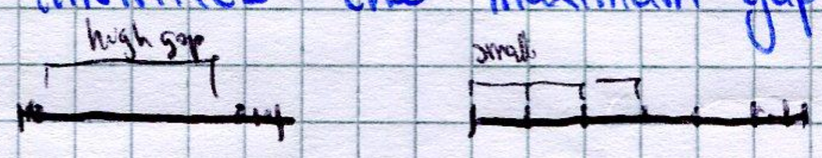
$1, 3, 6, 7, 9, \ldots \rightarrow 1, 2, 3, 1, 2$

$\#\text{bits to represent} = l = \max \lceil \log_2 a_1 \ldots a_{i-1} \rceil$

Assume UNIVERSE U and n integers

what is the distribution that MINIMIZES the maximum gap?

equally distributed!



but items might be internally distributed

$\Rightarrow$ Permutation $\pi: U \rightarrow U$

$A \cap B \longrightarrow \pi^{-1}(\pi(A) \cap \pi(B))$ permuting changes the ordering but doesn't change intersecti

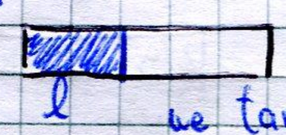
$(ax+b \bmod p) \bmod U$ p prime $\Rightarrow$ invertible, $p \geq U$

$\Rightarrow$ this way they've randomly permuted :) (forcing it)

Property given a random permutation π, maximum gap is $\leq \underbrace{\frac{U}{n}}_{avg} \log n$ w.h.p.

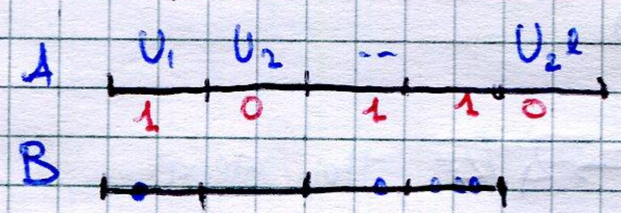
- We permute the elements of every list/set $\pi(s)$ $\forall s$
- for every set, we partition s into 2^{ℓ} buckets, $\ell = \lceil \log_2 \frac{\hat{n}}{L} \rceil$, $\hat{n}$ = maximum set size

  $x \in S \longrightarrow \pi(x)$ is a binary (integer) representation of $\log_2 U$ bits



we take the most significant ℓ buckets

- To intersect $A \cap B$



mask

I can skip empty buckets

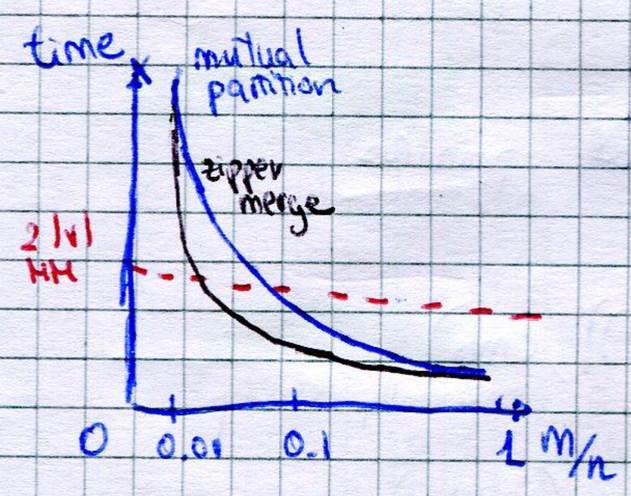
$\min\{\frac{n}{L}, ?\}$ $\leftarrow O(m)$ scan smaller!

cost intersection

$$\underbrace{\sum_{i:|B_i|\neq 0} \overbrace{|A_i| + |B_i|}^{\frac{|A|}{n/L}}}_{m} \leq \sum_{i:|B_i|\neq 0} \underbrace{\frac{|A|\cdot L}{\hat{n}}}_{\leq L} + m \leq \underbrace{mL}_{\leq n} + m = \min\{n, mL\}$$

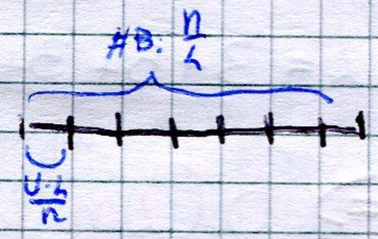
complexity $O(m + \min\{n, m \cdot L\})$

there's no "$+\frac{n}{L}$" anymore

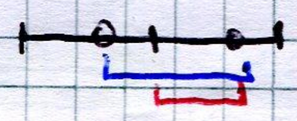


$2|v| >$ others at the start because $\frac{n}{L}$ is $> m \log n$ if m is small

Bit Space Complexity $\log_2 \frac{U}{n} \log n$



Store Not



$\Rightarrow$ max gap $\leq \min\{\frac{U}{n}\log n, \frac{UL}{n}\}$

$$\min\{\log_2(\frac{U}{n}\log n), \log_2 \frac{UL}{n}\}$$

$$\log_2 \frac{U}{n} + \log\log n \qquad \log_2 \frac{U}{n} + \log_2 L$$

$$\Downarrow$$

$$\log_2 \frac{U}{n} + \min\{\log\log n, \log_2 L\}$$

optimal smallest gap

4 min items $\Rightarrow$ 5

ex

$A = 1, 2, 3, 4, 7, 8, 11, 12$

$B = 2, 4, 6, 12$

$\pi(x) = 1 + (4x \bmod 13)$

$U = [1, \dots, 13]$

$\pi(A) = 5, 9, 13, 4, 3, 7, 6, 10$

$\pi(B) = 9, 4, 12, 10$

$\left.\begin{array}{l} L = 2 \\ \hat{n} = 8 \end{array}\right\}$ #bits $\quad \ell = \lceil \log_2 \frac{\hat{n}}{L} \rceil = 2$

= pick the 2 most significant bits to partition buckets

	00	01	10	11
U_A =	7/3	4/4 1/5 11/6 8/7	2/9 12/10	3/13
	1	1	1	1
U_B =		4/4	2/9 12/10	6/12

4 buckets

mask 1

Take 5 → 0101

we store the val too for simplicity, in reality it doesn't

$mask_B = 0111$ (check where == 1)

intersection between $\boxed{4/4 \dots}$ $\boxed{4/4}$ → 4/4, then intersect ...

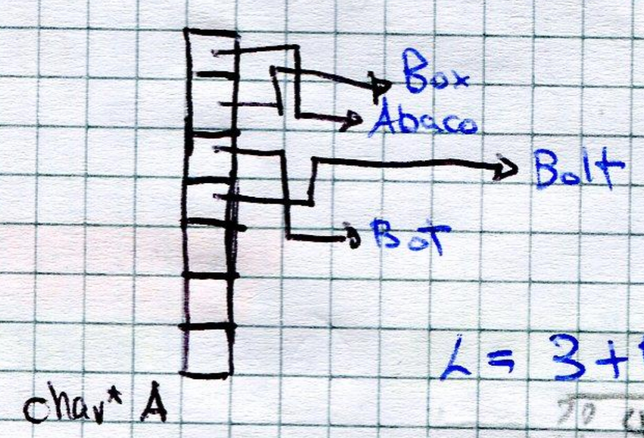
$A_{01} \cap B_{01} = 4/4 \quad A_{10} \cap B_{10} = 2/9, 12/10 \quad A_{11} \cap B_{11} = \times$

$\Rightarrow 4, 9, 10$ apply π^{-1} $4, 2, 12$

Note the fact the |bucket| is fixed permits "blindly" the intersection $A_i \cap B_i$

Sorting Streams

If you have to sort strings, we use an array A of pointers



we do qsort((A, n, sizeof(char*)), cmp)

usually strcmp

$O(n \log n)$ comparisons (on avg)

comparison might be for elem in ANY position

⇒ potentially 1 I/O every time

$L = \frac{3+5+3+4}{\text{avg length}}$

$\Rightarrow O(\underbrace{2 \cdot n}_{N} \cdot \log n)$

N: total string length

$\Sigma \equiv$ alphabet (256 ← byte, UNICODE → Big)

$\sigma = |\Sigma|$

Lower Bound: ① To sort the strings

$\Rightarrow$ at least first character sort $\quad \Omega(n \log n)$

② Distinguishing prefix

Abaco Box Bolt Bot

1 3 3 3

$\Omega(d) = \Omega(\sum_s d_s)$

3+1+3+3

qsort bad !!

$S_i = \underbrace{0 \ldots 0}_{l \text{ bits}}$ $\quad$ binary representation of i (log n bits)

useless to compare these

0 ... 0

00
01
10
11

$d_{s_i} = l + \log_2 n \Rightarrow d = \sum_s l + \log n = n(l + \log n)$

Low Bound $= \Omega(\underbrace{n(l+\log n)}_{n \cdot \text{length } \forall \text{string} = N} + n \log n) = \Omega(N + n \log n)$

qsort takes $(n \log n)$ comparisons.

$O(\underbrace{n(\log n)(l + \log_2 n)}_{N}) = O(N \cdot \log n)$

$\equiv$ the more the strings share characters, the worse !!

9/10

Today we will try to drop the log factor in the previouses complexities

Radix Sort
- MSD: MOST significant digit
- LSD: LESS significant digit

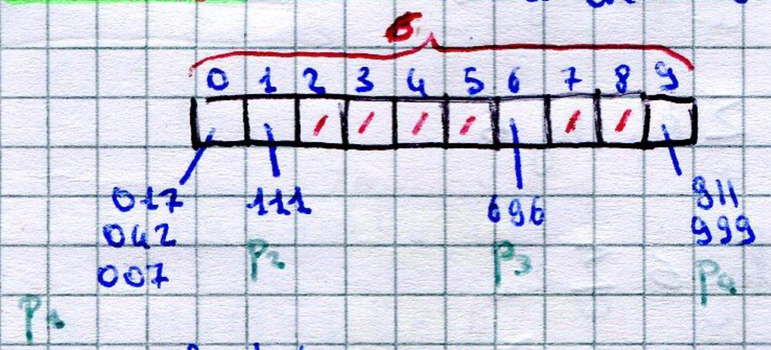
$\Omega(d + n \log m)$ — d: distinguishing prefix, n: #strings

both based on specific structures

In the MSD

Assume we have a set of strings $S = \{017, 042, 007, 111, 676, 911, 999\}$

$\Rightarrow$ **TRIE** Build a tree where every node is indexed with a digit

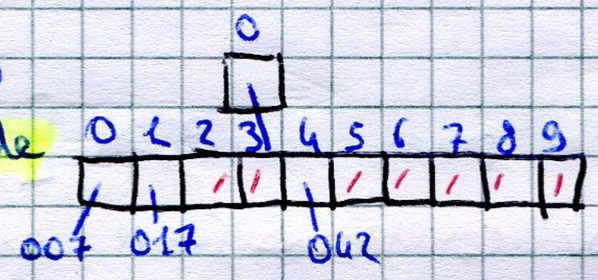


#entries = #possible chars $\quad$ bytes $\rightarrow$ 256, unicode $\rightarrow$ a LOT

where / it's a null pointer

if 1 elem $\Rightarrow$ 1 pointer to string

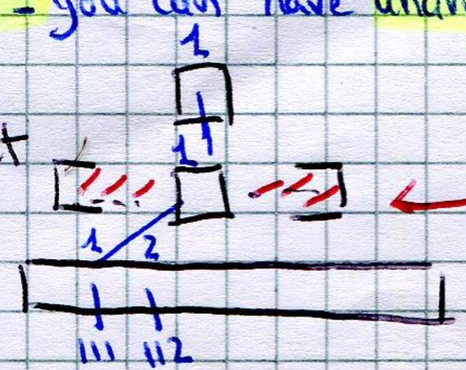
if > 1 elem $\Rightarrow$ create another node



$\Rightarrow$ every bucket contains one string

$\Rightarrow$ **UNCOMPACTED** (1 node per digit) $\equiv$ you can have unary paths

including 112 to S you get



$\leftarrow$ unary path (only 1 pointer used, inefficient)

With a PRE-ORDER visit you get the strings sorted

The tree consists of $O(d)$ nodes

$\Rightarrow$ space $= O(d\sigma)$ $\quad$ $\forall$ node $\rightarrow$ array of σ

Time $= O(d\sigma)$

sorting takes $\rightarrow$ construction (Trie) + visit (Trie) $\Rightarrow O(d\sigma)$

Creation(T) $O(d\sigma)$ $\quad$ Insertion(s) $O(\sum_s d_s) = O(d)$ $\quad$ $O(d\sigma)$

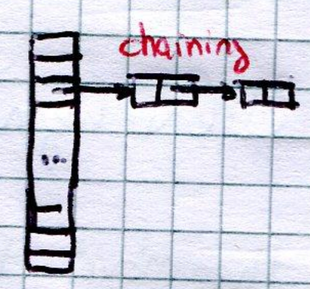
efficient if strings are different

inefficient if the distinguishing character is far from

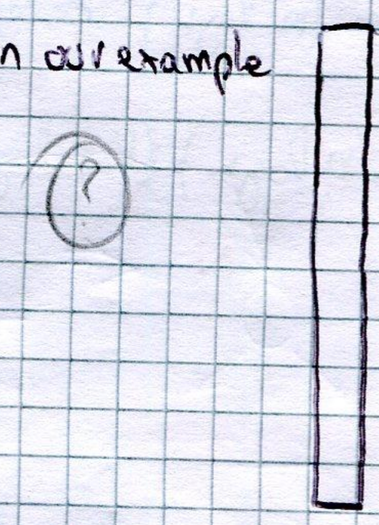
Improve this bound

Don't allocate σ when allocating the nodes

Instead of array $\rightarrow$ HASHTABLE



in our example



space $\Theta(\#keys) = \Theta(\#edges)$

$\Rightarrow$ creation $= O(d\cancel{\sigma})$

but visiting of the tree? Hashtable isn't sorted

$\Rightarrow$ sort edges according to the keys

$\Rightarrow$ construction (Trie) + SORT (edges $\forall$node) + visit (Trie)

Creat(T) $O(d)$ $\quad$ Ins $O(d)$ $\quad$ $O(d)$

$$\sum_u e_u \cdot \log e_u$$

$$\leq \sum_u e_u \cdot \underbrace{\log \sigma}_{\text{constant}}$$

$$= \log \sigma \underbrace{\sum_u e_u}_{d}$$

$$= O(d \log \sigma)$$

$\Rightarrow$ SPACE $O(d)$

TIME $O(d \log \sigma)$

NOTE Might sound better than the proved LowerBound $[\Omega(d + n \log n)]$

but that one is for COMPARISONS, we're exploiting the digits!

LSD

017	111	007	007
042	911	111	017
007	042	911	042
111	676	017	111
676	017	042	676
911	007	676	911
999	999	999	999

in-place sort ≡ counting sort

length

V step for L times

$O((n+6)\cdot L)$

space $O(N)$

cool idea: instead of one char by one, do two

$\Rightarrow O((n+6^2)\cdot \frac{L}{2})$

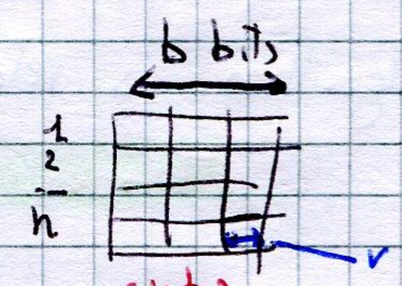
do 3 ...

$(n+6^3)\cdot \frac{L}{3}$

...

good until $6^i < n$

TRADEOFF



buckets =

$N \log_2 6$ bits all strings have same length $n\cdot b$ bits

$O\left(\frac{b}{r}(n+2^r)\right)$

Note ① $1 \le r \le \log_2 n \leftrightarrow 2^r \le n \Rightarrow O\left(\frac{nb}{r}\right)$ DECREASES WITH R

② $\log n \le r \le b \Rightarrow O\left(\frac{b2^r}{r}\right)$ INCREASES WITH R

$\Rightarrow$ **best** $r = \log_2 n \Rightarrow O\left(\frac{nb}{\log n}\right)$

Theo LSD-Radix Sort that sorts group of $\log_2 n$ bits each takes $O\left(\frac{nb}{\lg n}\right)$ time

and $O(nb)$ space (total length) $\equiv O(N\log 6)$

Less than linear !!

$\equiv O\left(\frac{N\log 6}{\log_2 n}\right)$

;1)

~~Kay~~ ~~Key~~ Quick ~~Hattiwag~~ ~~Quick~~ Sort (R, i) set to order

if $|R| \le 1$ return R

$p \in R$ is a PIVOT STRING

$R_< = \{ s \in R : s[i] < p[i] \}$ the i-th character is < than the i-th of p

$R_= = \{ \quad " \quad " = p[i] \}$

$R_> = \{ \quad " \quad " > p[i] \}$

gets called with parameters $(S, 1)$

$\Rightarrow$ Invariant all $s \in R$ share a prefix of $(i-1)$ characters

$\leftrightarrow$ 1st call, 0 chars in common

$A = \text{HxQS}(R_<, i)$

$B = \text{HxQS}(R_=, i+1)$

$C = \text{HxQS}(R_>, i)$

return $A \cdot B \cdot C$ (concatenation)

ex

R, 2 ≡ share 1 char (A)

Alabama
Abaco
Asturias
Avenue
Allure
Albanian
Amref
Attic
Average
Aviating
Amazing

take pivot at random → Amref & create $R_<, R_=, R_>$ DON'T SORT, JUST SCAN

$R_<$	$R_=$	$R_>$
Alabama	Amref	Asturias
Abaco	Amazing	Avenue
Allure		Attic
Albanian		Average
		Aviating
R, 2	R, 3	R, 2

Analysis Assume perfect pivot choice $|R_<| \wedge |R_>| \leq \frac{|R|}{2}$

$\Rightarrow |R_=|$ it's ok even if $= R$

Given (R, i), S → $s \in R_=$ → $i+1$ ①

→ $s \in R_< \vee R_>$ → i ②

#times ①? d_s

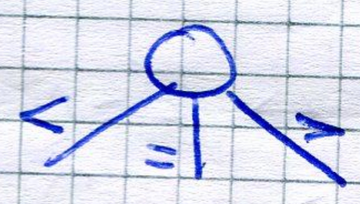
#times ②? $\log n$ times (every time we half the string set)

so total cost $\sum_s d_s + \log n = d + n \log n$

OPTIMAL in comparison based model

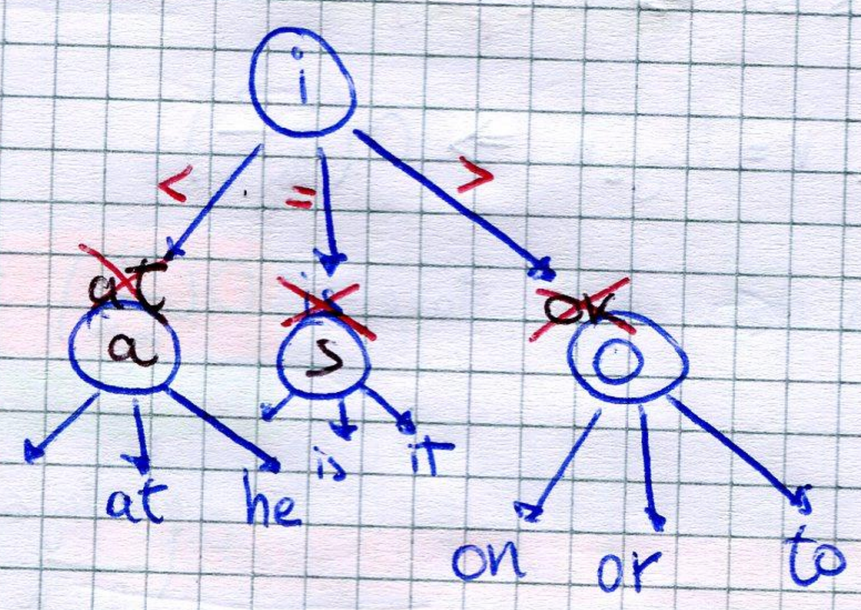
This algorithm is related to the data structure

Ternary Search Tree/Trie (3 childrens only)

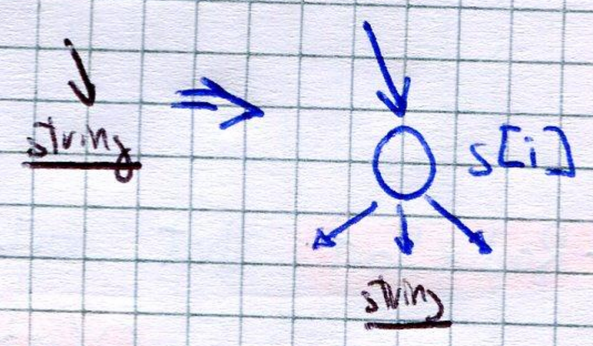


$\boxed{\text{fastest}}$

as
2 at
3 he
1 is random
6 on
5 or
7 to
4 it



In general



$\forall s$ $depth(s) = (d_s + \log n)$

$\equiv$ distinguish prefix $+ \log n$ as before

14/10

Argument #1 **Interpolation Search** [Similar to bucketSort]

NOTE start from 1 value!

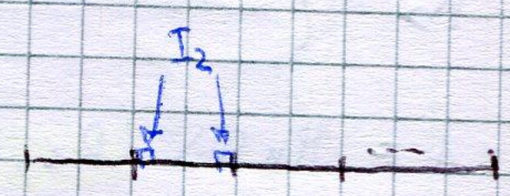
Array of m integers $x[1,m] = x_1, \dots, x_m \in \mathbb{N}$ SORTED

Preprocessing:

① PARTITION the range into m buckets #obj = #buckets $B_1, B_2, \dots, B_m$

$\forall i, |B_i| = \frac{x_m - x_1 + 1}{m}$

② BUILD an array $I[1,m]$



$\forall i$, $I[i]$ points to the first AND last keys in B_i

$\Rightarrow$ partition depends on values, so it's not necessarily true that $|b_i| = |b_j|, i \neq j$

might be empty as well!

ex

$B = \underbrace{1\ 2\ 3}_{B_1}\ \underbrace{8\ 9}_{B_3}\ \underbrace{17}_{B_6}\ \underbrace{19\ 20}_{B_7}\ \underbrace{28\ 30}_{B_{10}}\ \underbrace{32}_{B_{11}}\ \underbrace{36}_{B_{12}}$

$\underbrace{4,5,6}_{B_2}$

$|b_i| = \frac{36 - \cancel{1} + \cancel{1}}{12} = 3$

$I[1]$, $I[2]$ is null, ... $I[6]$ both points to the same, ...

How do you search in this?

Query (y) key $\longrightarrow$

- $J = \left\lfloor \frac{y - x_1}{b} \right\rfloor + 1$ this jump is constant time
- BinSearch (y, B_J) $\Rightarrow 1 + \text{cost}(\text{BinSearch})$

ex Query(27)

$\left\lfloor \frac{27-1}{3} \right\rfloor + 1 \Rightarrow I[9]$

Teo ① Searching takes $O(\log b)$ time

② Searching takes $O(\log \Delta)$ time

b = size of the basket

$\Delta = \frac{\max_{i=2,\dots,m} x_i - x_{i-1}}{\min_{i=2,\dots,m} x_i - x_{i-1}}$ $\left(\frac{\text{max gap}}{\text{min gap}}\right)$

DIM

Firstly, Lower Bound the max gap.

1st idea: with a sequence of items $\Rightarrow$ avg(sequence) <= max(sequence)

$$\max x_i - x_{i-1} \geq \sum_{i=2}^{m} \frac{x_i - x_{i-1}}{m-1} \overset{\text{Telescopic Sum}}{=} \frac{x_m - x_1}{m-1} \geq \frac{x_m - x_1 + 1}{m - \cancel{1} + \cancel{1}} = b$$

$\frac{a}{b} \geq \frac{a+1}{b+1}$ if $a \geq b$

$\Rightarrow$ max gap is AT LEAST the bucket size

#keys in the bucket AT MOST $\leq \frac{b}{s}$

bucket size $= b$, min gap $= 2 \Rightarrow$ AT MOST $\frac{b}{2}$ keys in b

$$\#\text{keys} \leq \frac{b}{s} \leq \frac{\max_{i=2,\dots,m} x_i - x_{i-1}}{\min_{i=2,\dots,m} x_i - x_{i-1}} = \Delta$$

$\Rightarrow$ binSearch in a bucket, log in #items $\Rightarrow$ proved $O(\log \Delta)$

Intuition: with an uniform distribution

max $\simeq$ min $\Rightarrow \Delta$ is a constant, so is the #keys, so everything's very fast! :)

m keys, m buckets, avg 1 key / bucket $\Rightarrow$ constant

We want to prove that $\Delta = O(\log^2 m)$ on uniform distributions w.h.p.

polylogarithmic in #keys

$\log(\log^2 m) \Rightarrow \log\log m$

we FORCE uniform distribution

$h(S) \longrightarrow S'$ the keys of S will be mapped in S' with a GOOD hash function

$\hookrightarrow S'$ uniformely distributed

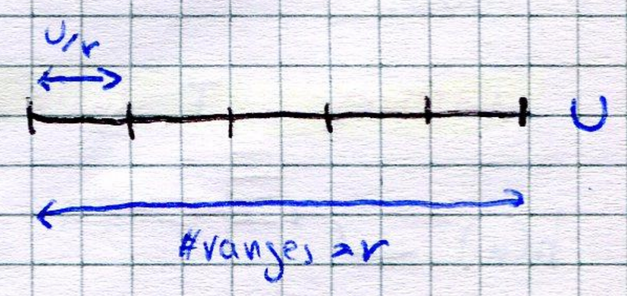
$\Rightarrow$ ask $h(y) \in S'$?

[NOTE the position of the key $\equiv$ Range Searches (give me all the keys between $y \leq \dots \leq y'$)

NOPE! Hashing shuffles stuff That's why we still use slower data structures

HASHING IS LIMITED IN ITS QUERIES!]

Partition the universe in $r = \frac{m}{2\log m}$ ranges of size $\frac{U}{r}$



$\mathbb{P}$(every range gets a key) $= 1 - \mathbb{P}(\exists$ range with no keys$)$ (#keys = m, #ranges = r)

$\Rightarrow$ max gap $\leq \frac{2U}{r}$ (2 blocks)

1st key goes outside, 2nd goes outside, ...

$\mathbb{P}$(key goes outside) $= \frac{\#\text{ranges} - 1}{\#\text{ranges}}$

$\mathbb{P}(\exists$ something$)$ can be bounded by UNION BOUND

$$\mathbb{P}(\exists \text{ range}) \leq \mathbb{P}(\text{range 1 empty}) + \mathbb{P}(\text{range 2 empty}) + \dots \qquad (\text{there's a } -)$$

$$1 - \mathbb{P}(\exists \text{ range with no keys}) \geq 1 - \sum_{i=1}^{r} \mathbb{P}(\text{range } i \text{ is empty})$$

$$= 1 - \underbrace{r}_{\frac{m}{2\log m}} \mathbb{P}(\text{range } \hat{i} \text{ is empty}) \quad \leftarrow \text{specific}$$

$$\geq 1 - m \cdot \mathbb{P}(\hat{i} \text{ is empty})$$

prob goes outside $= \frac{r-1}{r}$, must occur for $\left(\frac{r-1}{r}\right)^m$

$$\mathbb{P}(\text{empty}) = \left(1 - \frac{1}{r}\right)^m = \left[\left(1 - \frac{2\log m}{m}\right)^{\frac{m}{2\log m}}\right]^{2\log m} \simeq e^{-2\log m} = \frac{1}{e^{2\log m}} = \frac{1}{m^2} \qquad (e^{\log m} = m)$$

$1 - x \approx e^{-1}$

$$\Rightarrow 1 - r\,\mathbb{P}(\hat{i} \text{ empty}) \geq 1 - m \cdot \frac{1}{m^2} = 1 - \frac{1}{m}$$

$\mathbb{P}$ every range gets a key

$\Rightarrow$ max gap $\leq \frac{2U}{r}$ w.h.p. (m large)

$$\max = O\left(\frac{U}{m}\log m\right)$$
$$\min = \Omega\left(\frac{U}{m\log m}\right)$$

$$\Rightarrow \Delta \leq \frac{\frac{U}{m}\log m}{\frac{U}{m\log m}} = \frac{\cancel{U}\log m}{\cancel{m}} \cdot \frac{\cancel{m}\log m}{\cancel{U}} = \lg^2 m$$

Argument #2 — Dictionary Problem

D = dictionary of keys $\in U$

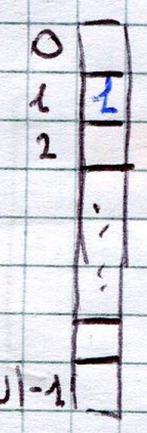
$n = |D|$ (the keys)

op ≠ ① SEARCH (key): $k \in D$?

② INSERT (key): $D = D \cup \{k\}$

③ DELETE (key): $D = D \setminus \{k\}$

Direct Address Tables

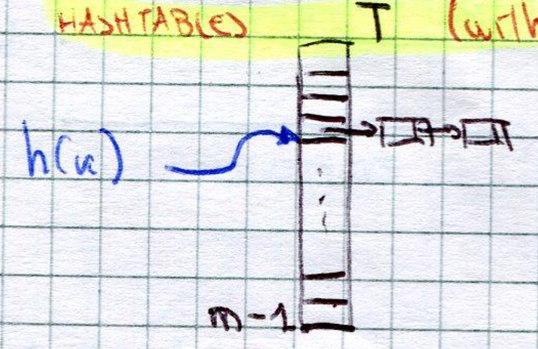


keys are integers or are easily mapped as ones

you receive ①, you put it in 1 [•] $\Rightarrow$ ① constant, ② return whole list, ③ put null

good only if $|U|$ is "small", otherwise ... ↙

HASHTABLES (with chaining)



Insert → in head ≡ constant

Delete = Searching

① the hash function $h: U \to \{0, 1, \dots, m-1\}$ is SIMPLE UNIFORM (distributes uniformely among the slots)

② **avg list length** $\mathbb{P}(\text{key goes in a specific slot}) = \frac{1}{m}$ (uniform)

$$\mathbb{E}[\text{length } s] = \sum_{i=1}^{n} \mathbb{P}(\text{key } x_i \text{ goes to slot } s) \cdot 1 = \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$ "α", load factor

on avg every list is $\frac{n}{m}$ long

③ **cost search**

cost of hashing + $O(1+\alpha)$ avg time

we want $\alpha = \Theta(1)$ so that the avg time is constant $\Leftrightarrow n \propto m \equiv m = \Theta(n)$

How many bits?

$m \cdot \log n$ bits for T + $n \cdot \log U$ bits for n keys + $n \log n$ bits for pointers "III→"

what about dinamicity? (α might increase over time)

⇒ **DOUBLING/HALVING** ≡ reconstruction r keys and table of $\Theta(r)$ ⇒ $\Theta(1)$ constant time per key ⇒ $\Theta(r)$

Start with n_0 keys, build T of size $m = 2n_0$

$$\alpha_0 = \frac{n_0}{m} = \frac{n_0}{2n_0} = \frac{1}{2}$$

At some point I insert n_1 keys so that #keys $= 2n_0$

$$\alpha_1 = \frac{n_1}{m} = \frac{2n_0}{2n_0} = 1$$ α is changing but cost is still constant

in α_1 I do reconstruction with $m > 2n_1$

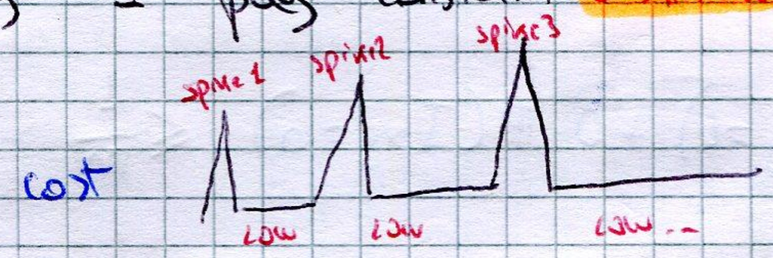
≈ "I return to the original solution"

10 → wait for 20 keys → 40 → wait for 80 keys → ...

$$\Rightarrow \underbrace{\Theta(2n_0)}_{m}^{\text{create T}} + \overset{\text{no insertions}}{n_0 \cdot \Theta(1+\cancel{\alpha}_{1})} + \underbrace{\Theta(2n_1)}_{\Theta(r)} = \Theta(4n_0)$$

⇒ $\Theta(n_0)$ to insert n_0 keys

∀key I pay constant ≡ **AMMORTIZED**



Ins/Del is $O(1)$ AMMORTIZED on avg

↑ depends on h