SORTING & PERMUTING : CHAP 5

SORTING: given a sequence of n atomic items $S[1,n]$ and a total ordering $\leq$ between 2 of them, want to sort S in increasing order

PERMUTATION: given $S[1,n]$ want to permute the sequence according to the permutation $\pi[1,n]$ of the integers $\{1,2,\dots,n\}$; want to permute S according to π to obtain $S[\pi[1]]; S[\pi[2]]; \dots; S[\pi[n]]$.

include ↓

- SORTING $\leq$ PERMUTING $\Rightarrow$ MERGE sort || heap sort
- IN RAM MODEL: sorting $\Theta(n \log_2 n)$ time; permuting $\Theta(n)$ (lower bound) time $\Rightarrow$ NO LOWER-BOUND GAP in I/O with conditions on input and model parameters n, M, B

$\Rightarrow$ CUZ moving items on the disk is the real bottleneck, rather than finding the SORTED PERMUTATION

MERGE-BASED SORTING ALGORITHM

Algorithm 5.1 The binary merge-sort: MERGESORT(S, i, j)

```
1: if (i < j) then
2:     m = (i + j)/2;
3:     MERGESORT(S, i, m − 1);
4:     MERGESORT(S, m, j);
5:     MERGE(S, i, m, j);
6: end if
```

line 3–4: $2T(\frac{n}{2})$; line 5: $\to O(n)$; $\to T(n) = 2T(n/2) + O(n) = \Theta(n \log_2 n)$

- |internal memory| = M.
- blocked-access to disk r/w B items at once.

1) if array has at least 2 items, otherwise 1 $\Rightarrow$ array already sorted;
2) split the array in 2 parts; $m = 0 + 4/2 = 2$
3-4) recurse on each part; // 1 element, "sorted by def."
5) as recursion ends, two halves are sorted, so MERGE (fuse them) in $S[i,j]$

	0	1	2	3	4
S	50	10	1	7	20
	i		m		j

mergeSort(S, i=0; m − 1 = 1);

mergeSort(S, m, j); Merge(S, i, m, j)

(m = 2, j = 4)

- MERGE SORT requires an extra space (is not an in-place algo like Quicksort || heapsort) $\Rightarrow O(n)$ extra-working space
- # recursive calls: $O(\log_2 n)$
- MERGE-procedure: takes $O(j - i + 1)$ time with 2 pointers x, y stating at the beginning of the 2 halves, $S[x]$ and $S[y]$ are compared
  $\Rightarrow$ smaller is written out and fused in the sequence
  $\Rightarrow$ its pointer advances

$\Rightarrow$ each comparison advances 1 pointer (the smallest) $\Rightarrow$ # steps upperbounded by # pointers advancements upperbounded by length $S[i,j] = n$ #steps $\leq$ # advancements $\leq n$

$$\Rightarrow T(n) = 2T(\tfrac{n}{2}) + O(n) = O(n \log n)$$

- ASSUMING: $n > M$ => S goes on Disk, running MERGE SORT

- 1 I/O: 5 ms, if roughly $5ms \cdot \Theta(n \log_2 n)$ where $n = 2^{30}$ few GB

$$\Rightarrow 5ms \cdot 2^{30} \cdot 30 = 10^8 ms \Rightarrow 1 \text{ day}$$

⇒ in reality few hours

⇒ CUZ ISN'T TAKEN IN ACCOUNT SIZE OF THE internal MEMORY M!

- SO:

- merging z items: $O\left(\frac{z}{B}\right)$ I/Os iff $\boxed{M \geq 3B}$ cuz MS keeps in internal memory:
  - 2 pages containing 2 pointers that scan $S[i,j]$, where $z = j - i + 1$
  - 1 page to write out the sorted output

- when 1 pointer advances to another disk page: an I/O FAULT occurs
  ⇒ so pages is fetched in internal memory & fusion continues (MERGE)

- Given S is stored contiguously on disk $S[i,j]$ occupies $O\left(\frac{z}{B}\right)$ => I/O bound for MERGING to subsequences of length z.

- WRITING OUT the merged sequence takes $O\left(\frac{z}{B}\right)$ => subsequences

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + O\left(\frac{n}{B}\right) = O\left(\frac{n}{B} \log_2 n\right)$$

MergeSort (rec. call) — $O\left(\frac{n}{B}\right)$ MERGE

⇒ NOT takes M in account yet!

⇒ MS splits recursively seq S in subsequences to be sorted

- when a subseq. of length z fits in internal memory $z = O(M)$
  ⇒ is entirely cached using $O\left(\frac{z}{B}\right)$ IOs, and subsequent sorting steps are done in memory (not on the disk)

⇒ SORTING a subsequence of $z = O(M)$ items no longer $\Theta\left(\frac{z}{B} \log_2 z\right)$ (PREV CORRECT REL)

BUT $O\left(\frac{z}{B}\right)$ I/Os ⇒ just for loading the subsequences in M

⇒ this save for all subseq. of size $\Theta(M)$ on which MS runs, that are $\Theta\left(\frac{n}{M}\right)$ in total

⇒ OVERALL saving $\Theta\left(\frac{n}{B} \log_2 M\right)$ ⇒ $\boxed{\Theta\left(\frac{n}{B} \log_2 \frac{n}{M}\right) \text{ I/Os}}$

⇒ accounts disk page and size of internal memory

OPTIMIZATIONS:

(1) STOPPING RECURSION

- set a threshold on subsequence size: $\boxed{j - i < cM}$ triggers stop of the recursion

⇒ fetches all sub-sequence in entirely in internal memory and applies the internal memory sorter to the subsequence

- c depends on the space-occupancy of the sorter ⇒ must guaranteed if works in internal M

- c is 1 for in-place sorters (InsertionSort || Heapsort)
- c close to 1 for Quicksort (cost of its recursion)
- c < 2,5 for MS (cost of extra array used by MERGE)

=> cost recursion stops at cM items => complexity becomes $\Theta\left(\frac{M}{B} \cdot \log_2 \frac{N}{cM}\right)$ I/Os

=> useless in asymptotic but cool in real cases

=> want have c close to 1, so reducing logaritmic factor in I/O complexity thus preferring in-place sorters (HPS || QS), InsertionSort cool if M is smaller => M few MegaBytes

(2) SNOW PLOW

- virtually increase M by a factor of 2 (ON AVG)
- scans input sequence S[1,n]
- generates SORTED RUNS whose length has VARIABLE SIZE LONGER THEN M (2M ON AVG)
  - => • First creates SORTED RUNS of variable length
  - • then applies over the SORTED runs the MERGE procedure -> in optimal # I/Os for the merge of sorted runs

=> $O\left(\frac{N}{B}\right)$ I/Os will suffice to halve the # of runs and thus total of $O\left(\frac{N}{B} \log_2 \frac{N}{2M}\right)$ I/Os will be used

ON AVG to produce the SORTED SEQUENCE => saving 1 pass over the data (cool if S is very big)

- Scanning the input items => # I/Os is linear
- Algorithm proceeds in PHASES => each phase produces a sorted run
  - A phase starts with internal memory filled of M unsorted items stored in an heap: H
    - Since the array-based implementation of heap doesn't require additional space

    => in addition to indexes items, can fit in the H as many items as we have memory cells available
  - PHASE: scans the input S (unsorted), at each step writes out the minimum item in H, called min and loads in in memory the next item from S, say next
  - TO GENERATE SORTED OUTPUT: can't store NEXT if NEXT < MIN in H

    => will brake the SORTED OUTPUT, will be the new min-heap and thus written out! PROPERTY

    => NEXT must stored in in an auxiliary array U (unsorted)
  - of course |H| + |U| = M over the whole execution of a PHASE
  - PHASE stops when H is empty and U has M unsorted items

    => thus next PHASE can start (storing new items in H and emptying U) correct ✓

- OBSERVATIONS:

1) During a phase execution, the minimum of H is non-decreasing => so it's non-decreasing the OUTPUT RUN
2) Items in H at the beginning of the phase will be eventually written out which thus is longer than M.

=> no less efficient than ReplaceSort

Algorithm 5.2 A phase of the Snow-Plow technique

Require: $\mathcal{U}$ is an unsorted array of M items
1: $\mathcal{H}$ = build a min-heap over $\mathcal{U}$'s items;
2: Set $\mathcal{U} = \emptyset$;
3: **while** ($\mathcal{H} \neq \emptyset$) **do**
4: `min` = Extract minimum from $\mathcal{H}$;
5: Write `min` to the output run;
6: `next` = Read the next item from the input sequence;
7: **if** (`next` < `min`) **then**
8: write `next` in $\mathcal{U}$;
9: **else**
10: insert `next` in $\mathcal{H}$;
11: **end if**
12: **end while**

Require: M is unsorted array of M items
1) H: build min-heap over M's items
2) set U = ∅;
3) while (H ≠ ∅) do:
4) min = extract min from H
5) write min to output RUN
6) next = read next item from Sequence
7) if (next < min)
8) put next in U
9) else
10) put next in H
11) end if
12) end while

- ACTUALLY MORE EFFICIENT ON AVG than $O\left(\frac{N}{B}\log_2\frac{N}{2M}\right)$ I/Os
- suppose a phase reads K-items in total from S
- by while (H ≠ ∅) a phase will ends when H is empty and |U| = M
- Read items will go part in H and part in U.
- but since the items are added to U and never removed during a phase, M of the K-items end up in U
  => (K − M) items are inserted in H and written out to the output (SORTED) RUN

=> length of the sorted run at the end of the phase is $M + (K - M) = K$

where M: items in U -> H at the beginning of a phase

(K − M): items read from S and inserted in H during a phase

- Key issue is to compute AVG of K

=> assuming a RANDOM DISTRIBUTION of the input items

=> we have $P(next < min) = \frac{1}{2}$

=> thus have equal Probability that a read item will go in H or in U

K: runs the keys in one run

=> OVERALL $\frac{K}{2}$ items go to H & $\frac{K}{2}$ items go to U

=> but we know that items inserted in the U are M, so we can set $M = \frac{K}{2}$

and thus we get $K = 2M$ cvz $2 \cdot M = \frac{K}{2} \cdot 2 \Rightarrow K = 2M$

NB • if NO RAND DIST: like in H goes K items with prob $\frac{1}{3}$

=> ∞ in $P(U) = \frac{2}{3}$

$M = U \Rightarrow \frac{3}{2} \cdot M = \frac{2}{3} K \cdot \frac{3}{2}$

$\Rightarrow K = \frac{3}{2} M$

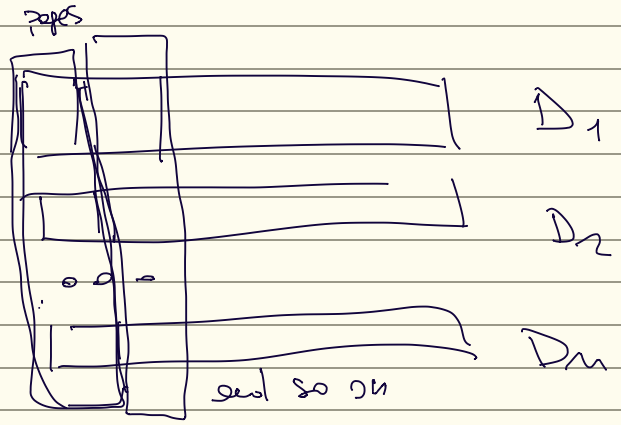
MULTI-WAY MERGE SORT

- prev. optimizations use M to stop recursion
- but merging was binary (fuses 2 input runs at time)
  => binary-merge impact the base of $\log_2$ ($\frac{N}{B} \log_2 \frac{N}{M}$)

- DISK STRIPING

  D - DISK if I fetch 1 page per disk D every time I finish to sort a page need to fetch a new one, not said that next page I fetch is not the next in sequence
  => can't guarantee parallelism
  => disk stripy

- D disks
- Block the pages in the following



$B' = D \cdot B$

applying MULTI-WAY MS

$$O\left(\frac{N}{B'} \cdot \log_{B'} \frac{N}{M}\right) \text{ I/O} = O\left(\frac{N}{D \cdot B} \log_{D \cdot B} \frac{N}{M}\right) \text{ I/O}$$

not optimal but see what's going on

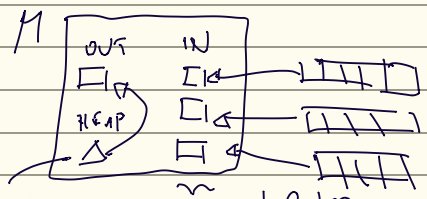
D-DISK I/Os

OPTIMAL I/O with sh-disk $\Omega\left(\frac{N}{D \cdot B} \cdot \log_B \frac{N}{M}\right)$

$$\Rightarrow \frac{\frac{N}{DB} \cdot \log_{DB} \frac{N}{M}}{\frac{N}{DB} \cdot \log_{\frac{M}{B}} \frac{N}{M}} \approx 1$$

$$\frac{\log_2 \frac{N}{M}}{\log_2 \frac{N}{DB}} : \frac{\log_2 \frac{N}{M}}{\log_2 \frac{N}{B}} = \frac{\log_2 \frac{N}{M}}{\log_2 \frac{N}{DB}} \cdot \frac{\log_2 \frac{N}{B}}{\log_2 \frac{N}{M}} =$$

$$= \frac{\log_2 \frac{N}{B}}{\log_2 \frac{N}{DB}} = \frac{\log \frac{N}{B}}{\log_2 \frac{N}{B} - \log}$$



$\log_2 K$

- K - in blocks
- 1 out block
- M has $\frac{M}{B}$ pages
- in + out = $\frac{M}{B}$
- $K = \frac{M}{B} - 1$ # in blocks

- MULTI - WAY MERGESORT (K)
  - want merge more then 2 runs at time
  - => merge K - runs at time

(NORMAL)

- Binary merge uses 3 Blocks: 2 for reading in, 1 for output on disk
  - but memory M has more than 3 blocks (exploit this fact!)

$$\Rightarrow \frac{M}{B} >> 3$$

=> FUSE K-runs at time, $K >> 2$

- want again hinges onto min-heap DS:
  - contains K pairs:

$\langle R_i[1], i \rangle$ where R_i : i-th input RUN

(item, origin run)

- at each step:
  - EXTRACT THE PAIR CONTAINING THE SMALLEST item in H ($R_i[1]$),
  - WRITE $R_i[1]$ to the OUTPUT
  - INSERT IN THE HEAP the NEXT ITEM IN ITS ORIGIN RUN

example!

t[redacted]. As an example, if the minimum pair is $\langle R_m[x], m \rangle$ then we write in output $R_m[x]$ and insert in $\mathcal{H}$ the new pair $\langle R_m[x+1], m \rangle$, provided that the mth run is not exhausted, in which case no pair replaces the extracted one. In the case that the disk page containing $R_m[x+1]$ is not cached in internal memory, an I/O fault occurs and that

- Merging requires $O(\log_2 K)$ time per item & $O(\frac{z}{B})$ I/O to merge K runs of length z

- as a result, merging scheme recalls a K-way TREE with $O(\frac{N}{M})$ leaves (RUNS)
(like with snow-plow)

=> total # of merging levels is $O(\log_{\frac{M}{B}} \frac{N}{M})$ for IOs equal to $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{M})$

wt $\log_{\frac{M}{B}} M = \log_{\frac{M}{B}} \left(B \cdot \frac{M}{B}\right) =$

$= \log_{\frac{M}{B}} B + \log_{\frac{M}{B}} \frac{M}{B} = \left(\log_{\frac{M}{B}} B\right) + 1 = \Theta\left(\log_{\frac{M}{B}} B\right)$

so asymptotically $\log_{\frac{M}{B}} \frac{N}{M} = \Theta\left(\log_{\frac{M}{B}} \frac{N}{B}\right)$

$= O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$

THEOREM 5.1 *Multi-way Mergesort takes $O(\frac{n}{B} \log_{M/B} \frac{n}{M})$ I/Os and $O(n \log n)$ comparisons/-time to sort n atomic items in a two-level memory model in which the internal memory has size M and the disk page has size B. The use of Snow-Plow or integer compressors would virtually increase the value of M with a twofold advantage in the final I/O-complexity, because M occurs twice in the I/O-bound.*

LOWER BOUND PERMUTING VS SORTING

place of n.

	Time complexity (RAM model)	I/O complexity (two-level memory model)
Permuting	$O(n)$	$O(\min\{n, \frac{n}{B} \log_{M/B} \frac{n}{M}\})$
Sorting	$O(n \log_2 n)$	$O(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{M})$

MSD $= O(d \cdot \sigma)$

MSD can hash $(d \log_2 \sigma)$

LSD $= \Theta\left(\frac{L}{\log_2 M}\right)$

$\Omega = \left(d + n \log_2 \frac{n}{2}\right)$

SNOW PLOW

$O\left(\frac{N}{B} \log_{\frac{M}{2M}}\right)$ I/Os

- Require: M unsorted array of N items

1) H = bid min-heap over U items
2) set U = ∅
3) while (H ≠ 0) do
4) min = extract the minimum from H
5) write min to the output
6) next = read next item from S
7) if (next < min)
8) add next in U
9) else add next in H
10) end if
11) end while

- N items ends up in U
- length of sorted run is $N + (K - N) = K$
  - items read from S that go in H
- assuming random distribution

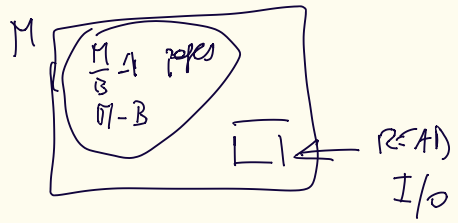
$$P(\text{items go in } U) = P(K \text{ items go in } H) = \frac{K}{2}$$

- But items inserted in U are N

⇒ so $N = \frac{K}{2}$ $\quad K = 2N$

PROVIDING LOWER BOUND FOR SORTING

1) $\binom{M}{B}$ = # ways for distributing B items among M-B existing in memory



total # of comparisons done for 1 I/O

2) 1) NEW I/O : $\binom{M}{B} \cdot B! = x \iff \frac{M}{B}$

2) OLD I/O : $\binom{M}{B} = y \iff t - \frac{M}{B}$

3) $n!$ leaves

=> I've to find t for which

$$x^{\frac{M}{B}} \cdot y^{t-\frac{M}{B}} \geq n!$$

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

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

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

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

$$\frac{M}{B} \log B! + t \log\binom{M}{B} \geq n \log n$$

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

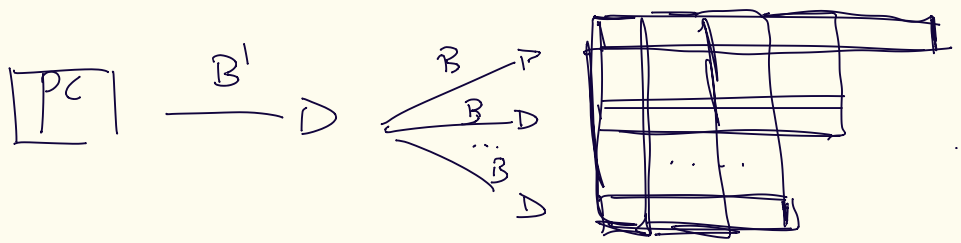
$$t B \log \frac{M}{B} \geq n \log n - n \log B$$

$$t \frac{B \log \frac{M}{B}}{B \log \frac{M}{B}} \geq n \left(\log \frac{n}{B} \right) \Big/ B \log \frac{M}{B}$$

$$t \geq \frac{n}{B} \frac{\log \frac{n}{B}}{\log \frac{M}{B}}$$

$$t \geq \frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}$$ asymptotically is $\frac{n}{M}$ ✓

D - DISK => DISK STRIPING



$B' = D \cdot B$

performance of disk striping / optimal

on sorting: $O\left(\frac{M}{DB} \cdot \log_{\frac{M}{DB}} \frac{M}{M}\right)$

$$\frac{\frac{M}{DB} \log_{\frac{M}{DB}} \frac{M}{M}}{\frac{M}{DB} \log_{\frac{M}{B}} \frac{M}{M}} \overset{(\log_2)}{=} \frac{\frac{\log_2 \frac{M}{M}}{\log_2 \frac{M}{DB}}}{\frac{\log_2 \frac{M}{M}}{\log_2 \frac{M}{B}}} = \frac{\log_2 \frac{M}{M}}{\log_2 \frac{M}{DB}} \cdot \frac{\log_2 \frac{M}{B}}{\log_2 \frac{M}{M}} =$$

$$= \frac{\log_2 \frac{M}{B}}{\log_2 \frac{M}{DB}} = \frac{\log_2 \frac{M}{B}}{\log_2 \frac{M}{B} - \log_2 D} = \frac{\log_2 \frac{M}{B}}{\log_2 \frac{M}{B}\left(1 - \frac{\log_2 D}{\log_2 \frac{M}{B}}\right)} =$$

$$= \frac{1}{1 - \log_{\frac{M}{B}} D} \geq 1$$

=> WITH 1 DISK is optimal

$\frac{1}{1}$

GOING FOR QS : 2 SORTING PARADIGMS

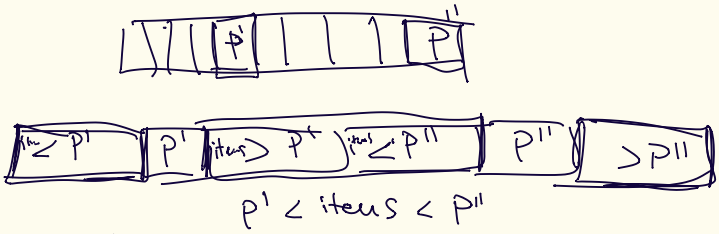
1) MERGE - BASED (MS)
   - partition O(1) (easy) (middle element)
   - recursion : recurse
   - recombination (merge) $O(n)$ / $O(\frac{n}{B})$ (linear)

2) DISTRIBUTION - BASED (QS) - pick array $O(n)$ $(O(\frac{n}{B}))$
   - PARTITION (costly): - take pivot
     - items distributed according to the pivot [<P][=P][>P]
   - RECURSION : recurse
   - RECOMBINATION : items are already relatively ordered
     : cuz divide into < smaller and > larger
     => when < and > are sorted => everything is sorted

=> we want MULTI-WAY QS => need MORE THEN 1 PIVOT

- 1 PIVOT => binary
- 2 PIVOTS => 3 partitions
- k PIVOTS => k+1 partitions

- PARTITIONS MUST BE BALANCE



- 1 PIVOT => binary => each partition $\frac{n}{2}$ items
- 2 PIVOTS => PARTS $n/3$ items
- k PIVOTS => each partition $\frac{n}{k}$ items

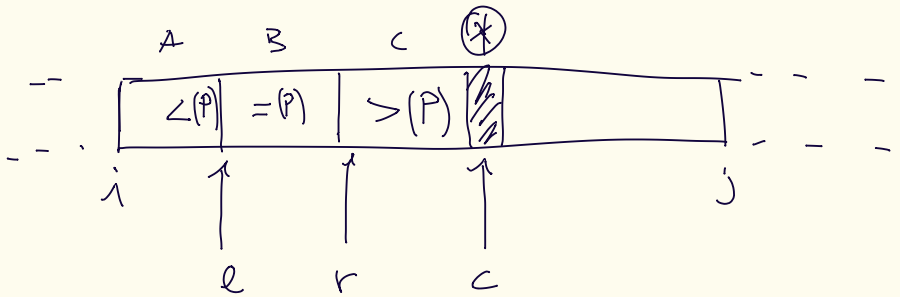
STD QS

```
if (i < j)
    m = PARTITION (S, i, j)
    qs(S, i, m)
    qs(S, m+1, j)
```

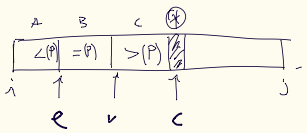
3-WAY PARTITIONING

PARTITION procedure

- pointers: c, left, right : points to the first item of each part
  (left = l, right = r)



Invariant: consider 3 parts instead of 2



part of ARRAY processed inductively

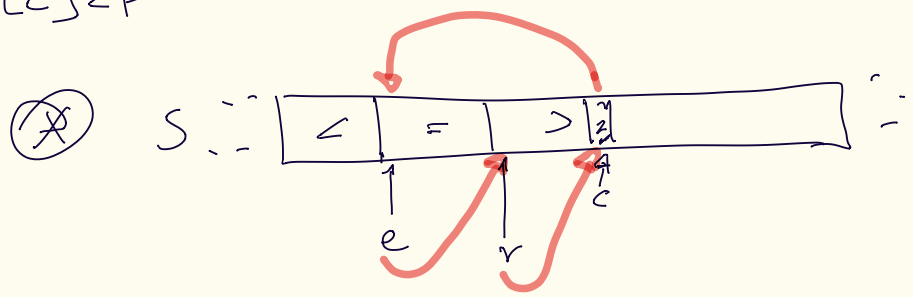
- RECURSIVELY: S[c]

- c points to first item next to $\geq$. Can happen that element pointed by c:

  1) $S[c] < P \Rightarrow$ (*)
  2) $S[c] = P \Rightarrow$ SWAP S[c] with S[r]; r++; c++
  3) $S[c] > P \Rightarrow$ S[c] stays here, increment c (c++)

and I put S[c] in A, B or C
$\leq$, $=$ or $\geq$

S[c] < P



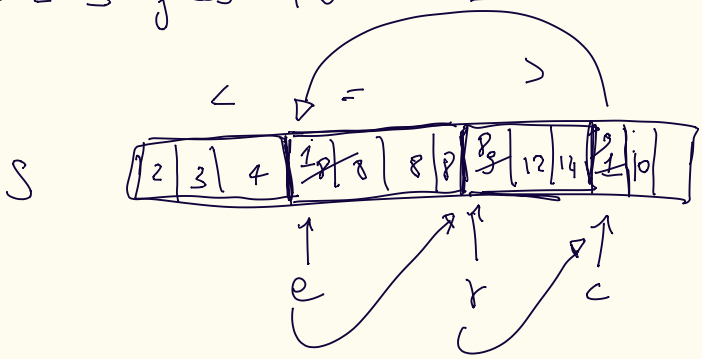
S[c] goes to S[l]

S[l] goes to (and to the equal) S[r]

S[r] goes to S[c]

- l++
- r++
- c++

P = 8



b) ADVANTAGE OF 3-way partitioning

→ with a LOT of equal items => recursion goes just for |L| and |S|

=> |E| has NOT to BE SORTED

- **3 PIVOT ≠ 3-WAY PARTITIONING** (PARTITION)
  - Take **S-PIVOTS**: $P_1, P_2, \dots, P_s$ taken at **RANDOM**
  - choose the **BEST** (one) according to **BALANCED PARTITIONING**
    - **SORT** the pivots
      - take the **MEDIAN**

es 3 pivot: $P_1 = 5$, $P_2 = 4$, $P_3 = 4$

=> sort $(P_1, P_2, P_3) = 1, 4, 5$

=> take (4) = P_2

- MORE PIVOT I HAVE, THE BETTER IS THE ESTIMATION FOR **BALANCED PARTITIONING**

- BUT, to sort the pivots I pay $O(s \log s)$

=> the problem you want to solve

=> so choose **3 PIVOTS**

• Proof time complexity of Quicksort is $n \log n$ (on avg)

Cuz is $o(n^2)$ wc

• given a RANDOM VARIABLE; to count # of comparisons

$$X_{u,v} = \begin{cases} 1 & \text{iff item } S[u] \text{ is compared with item } S[v] \\ 0 & \text{otherwise} \end{cases}$$

AVG # comparisons: $\mathbb{E}\left[\sum_{u=1}^{n} \sum_{v>u}^{n} X_{u,v}\right]$ (1)

$\mathbb{E}$ (avg): VALUE · PROB. of value and sum all possible values

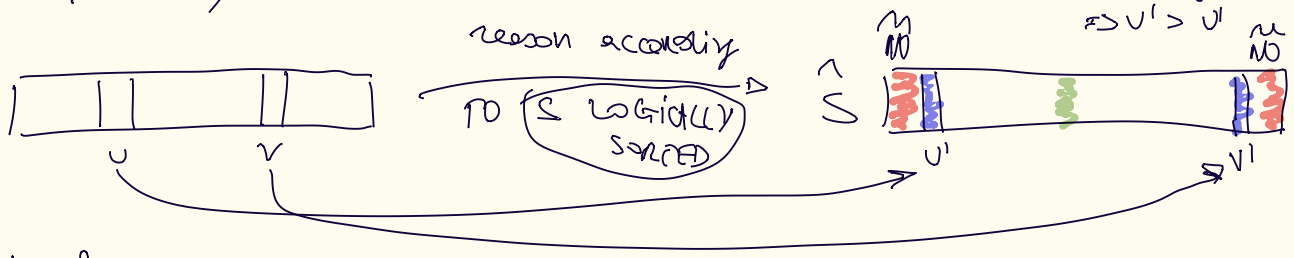
Prob to get 1 = prob 2 items are compared

$$(1) = \sum_{u=1}^{n} \sum_{v>u}^{n} \mathbb{E}[X_{u,v}] = \sum_{u=1}^{n} \sum_{v>u}^{n} 1 \cdot \mathbb{P}(S[u] \text{ compared } S[v]) + 0 \cdot \mathbb{P}(\text{otherwise}) =$$

u / v Binary values = 2 values

$$= \sum_{u=1}^{n} \sum_{v>u}^{n} \mathbb{P}(S[u] \text{ compared } S[v])$$

$\mathbb{P}(S[u] \text{ comp } S[v])$: $\mathbb{P}$ ($S[u]$ pivot or $S[v]$ pivot in 1 call involving both)



• 3 SITUATIONS

A) PIVOT is [illegible] or [illegible] => left of u' or right of v' => u', v' NOT COMPARED (INTERVAL cuz one on the same PARTITIONING)

B) PIVOT between u' and v' => u' and v' NOT COMPARED EACH OTHER AND NEVER cuz go in different partitions

C) PIVOT is u' or v' [illegible]

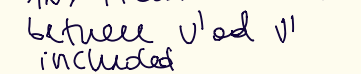
<=> HAVE THE COMPARISON $X_{u,v} = 1$

=> estimate Probability (C)

$$\mathbb{P}(C) = \frac{2}{\underbrace{B+C}_{\text{TOTAL events}}} = \frac{2}{v' - u' + 1} \Rightarrow O(n \log_2 n)$$

ANY item chosen between u' and v' included

NOT u cuz drop case (1)



$$\sum_{u'=1}^{n} \sum_{v'>u'} \frac{2}{v'-u'+1} = O(n \log n)$$

$$\sum_{u=1}^{n}\sum_{v=u+1}^{n} p_{u,v} = \sum_{u'=1}^{n}\sum_{v'>u'} \frac{2}{v'-u'+1} = 2\sum_{u'=1}^{n}\sum_{k=2}^{n-u'+1} \frac{1}{k} \le 2\sum_{u'=1}^{n}\sum_{k=2}^{n} \frac{1}{k} \le 2n\ln n,$$

- CHOOSE K^{TH} - RANKED ITEM IN AN (UNSORTED) ARRAY S
  => O(n) on AVG

---

1) take S
2) choose an item that if S would be sorted, is the K^{TH} one

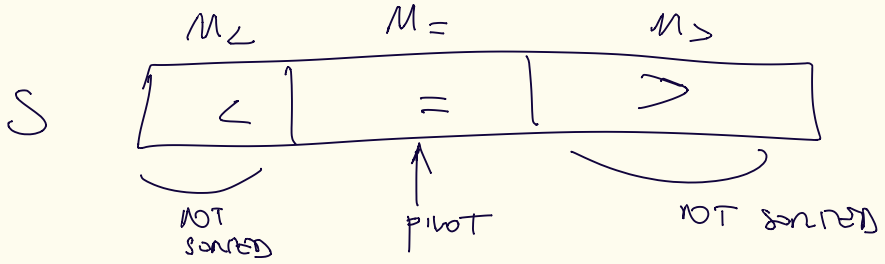
(obvious: sort, take item in position k)
↳ $n \log(n \log n)$ on AVG

=> SOLVE IT WITH PARTITIONING STRATEGY with 1 recursive call (not 2 like es)

IDEA
- take S
- take RANDOM PIVOT
- APPLY 3-WAY PARTITIONING
  O(n) time

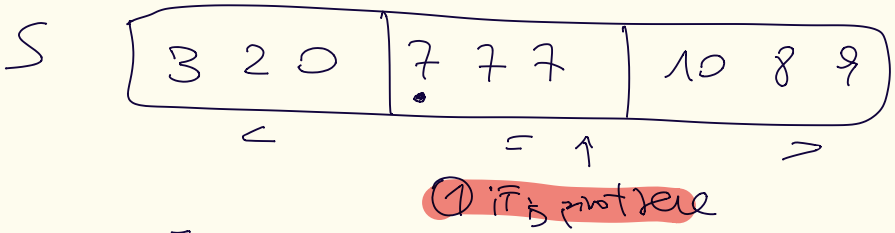
search K^{TH} item (smallest)

n = # of items in each partition



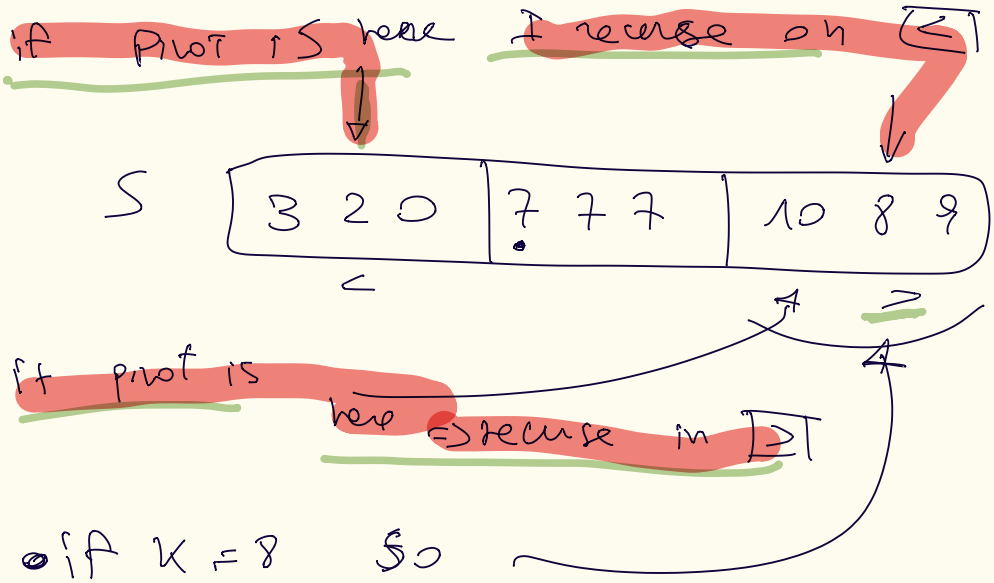
- search K^{TH} - item (or the smallest K^{th} item

  - if k falls in |=| I know what is the item



k=4

- if S[k] => S[4] is the repeat
  => I know => 7
- if k falls in |<| or |>| I can discard some item



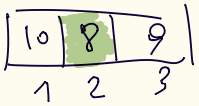
1 RECALL
=> discard
< and =

- if $k = 8$ so I have to re-define k

$k = k - n_= - n_<$

=> drop ↑

k was 8
NEW $k = 8 - 3 - 3 = 2$
in fact



=> so 2ND item in > not 0TH item
ON THE LEFT, NOTHING TO DROP

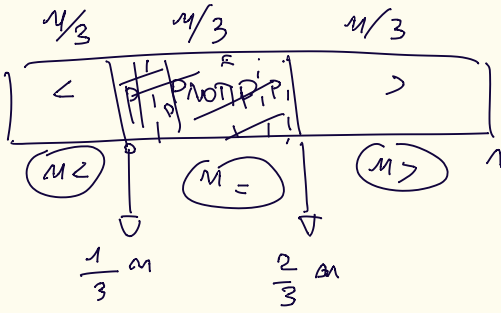
COMPLEXITY of k^{TH}-ranked item in unsorted array

$$T(n) = \underbrace{O(n)}_{\text{PARTITIONING (3-WAY}} + \underbrace{\left[T(n_<) \text{ OR } T(n_>) \right]}_{\text{GOING TO THE LEFT OR TO THE RIGHT}}$$

=> $n_<$ complexity · P(I'm there in $n_<$ OR (sine)

- Probability I'm in one of these ones

considering S sorted



- if pivot falls between $\frac{1}{3}n$ and $\frac{2}{3}n$ =>

#items in $n_<$ $\leq \frac{2}{3}n$
#items in $n_>$ $\leq \frac{2}{3}n$
} SAME

PIVOT COULD BE AT THE END

=> so probability that Pivot goes in $[=]$ is $\frac{1}{3}$

cuz $\frac{n}{3} : n = \frac{1}{3}$

So GOOD CASE is $\frac{1}{3}$

BADE CASE is $1 - \frac{1}{3} = \frac{2}{3}$

- in WORST CASE I've n-1 items (just drop the pivot)

$$\Rightarrow T(n) \leq \underbrace{O(n)}_{\text{PARTITION}} + \frac{1}{3} T\left(\frac{2}{3}n\right)_{\text{GOOD}} + \frac{2}{3} T(n-1)_{\text{BAD}}$$

$\leq T(n)$

$$T(n) - \frac{2}{3}T(n) \leq O(n) + \frac{1}{3} T\left(\frac{2}{3}n\right)$$

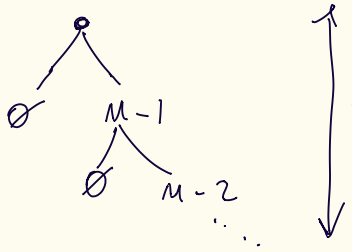
$$3 \cdot \frac{1}{3} T(n) \leq O(n) \cdot 3 + \frac{1}{3} T\left(\frac{2}{3}\right) n \cdot 3$$

$$T(n) \leq O(n) + T\left(\frac{2}{3}\right) n$$

Apply MASTER THEOREM $O(n)$

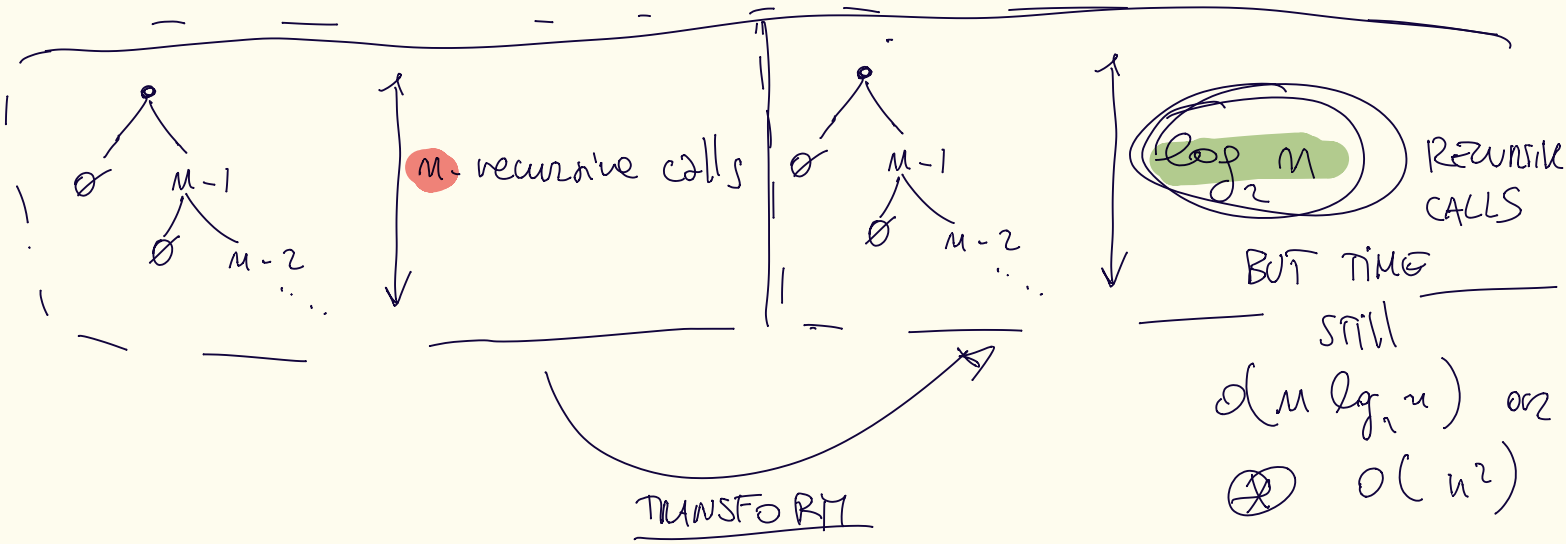
BOUNDED QS

QS:
- take away
- split array (partitioning)
- recurse left and right
- WC: totally unbalanced partition: 1 empty, other is full $O(n^2)$ ☹



=> space (n) in the heap for recursion
=> costly!

=> SOLUTION TO BOUND # recursive calls (not complexity but space)
time still AVG $O(n \log n)$ WC $O(n^2)$



☹ I'm increasing complexity of iterative part

- REMOVING TAIL RECURSION

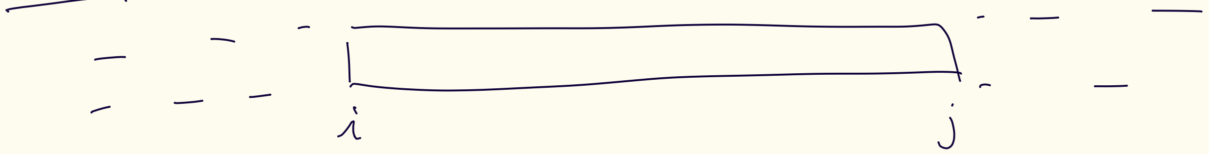
FROM STACK-OVERFLOW

QS ws $O(n^2)$ time => takes $O(n)$ SPACE

- CUT TAIL RECURSION: recurse on the smallest half and loop to sort larger half

ARRAY: generic call, I'm between position i and j

Q SORT

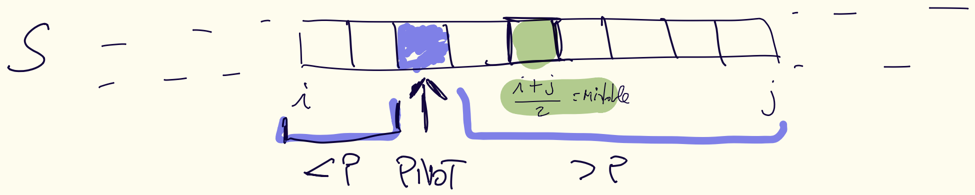


if $(j-i) >$ some value $M_\emptyset$ => insertion sort

// if this piece of array is small => useless shoot with recursion => just sort in cache
• insertion sort is quadratic but $M_0 \approx 10, 20$

else
- pick pivot
- do partitioning
- 2 SITUATIONS: (UNBALANCED PARTITIONS)

1) left smaller right large ←ANALYZE→ 2) left big or right small



1) _ left partition smaller
- right is longer

• middle element evidenzia che la parte di sx ha meno elementi della metà

$\Rightarrow \left(i \leq p < \frac{i+j}{2}\right)$

=> RECURSE ON THE LEFT (smaller half)

=> # of REC. CALL is logaritmic (cuz we RECURSE on PARTITION THAT'S SMALLER THEN HALF!!)

Algorithm 5.6 Binary QuickSort with bounded recursion: BOUNDEDQS(S,i,j)

```
while j − i > n0 do
    r = select the position of a "good pivot";   (RANDOM)
    swap S[r] with S[i];
    p = PARTITION(S, i, j);
    if p ≤ (i+j)/2 then
        BOUNDEDQS(S, i, p − 1);   left recursion
        i = p + 1;
    else
        BOUNDEDQS(S, p + 1, j);   right recursion
        j = p − 1;
    end if
end while
INSERTIONSORT(S, i, j);
```

• AFTER recursive calls on the left
• DON'T RECURSE ON THE RIGTH
=> JUST MOVE (i)

=> $i = p+1$

AND do a while

while
if $(j-i) > M_\emptyset$ => insertion sort

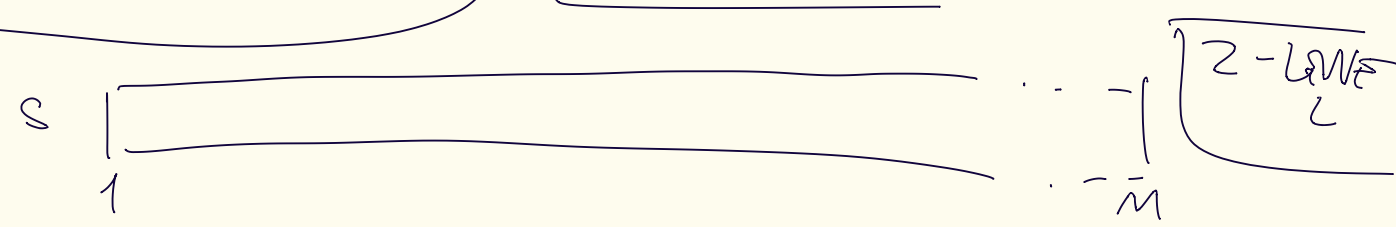
$\left(i \leq p \leq \frac{i+j}{2}\right)$ => • RECURSE on THE LEFT PARTITION
• $i = p+1$ => so move the iterator and continue the while

else => recurs on the right partition
- $j = p-1$

- on right core => rec. on right
now $j = p-1$

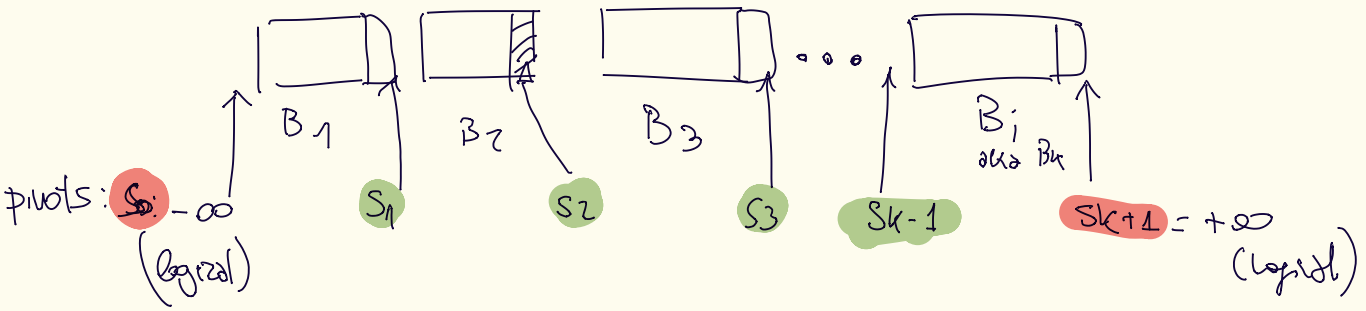
MULTI-WAY QS : SCALE ON DISK

- S not fit all in M



- PARTITION S among K buckets (parts)

- Ogni bucket ha associato un PIVOT => HO $(K-1)$ pivots



S: splitting element

idea: Bucket $B_i = \{ S[j] : \text{pivot } S_{i-1} < S[j] \leq S_i \}$

(what is in the picture)

es Bucket B_2:
- there are all items $> S_1$ e $\leq S_2$

$S_1 < B_2 \leq S_2$

GOAL: AVERE i BUCKET BILANCIATI

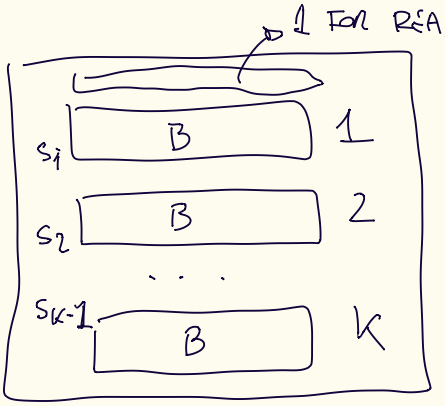
=> Balanced B_i, $\forall i$:

$$|B_i| \simeq \frac{M}{K}$$

=> recursion tree balanced

$\log_{(K)} \dots$

assume: in (M) we have K-BUKETS of size B



• Guarantee $B \cdot K \leq M$ (like MERGE WAY MS)

1) • extract => sample at random $(K-1)$ PIVOTS items from S

by SCAN => $O(\frac{N}{B})$ IOs (random sampling)

2) • SORT the PIVOTS : $s_1 < s_2 < \dots < s_{K-1}$

(QS, MS ...)

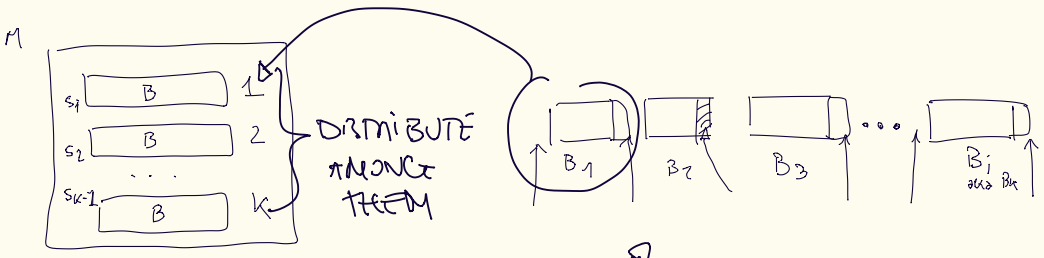
ϕ IOs (done in internal memory)

car $K \leq \frac{M}{B}$

by $K \cdot B \leq M$

3) • DISTRIBUTE : elements of S among the PIVOTS

(BUT $S >> M$)

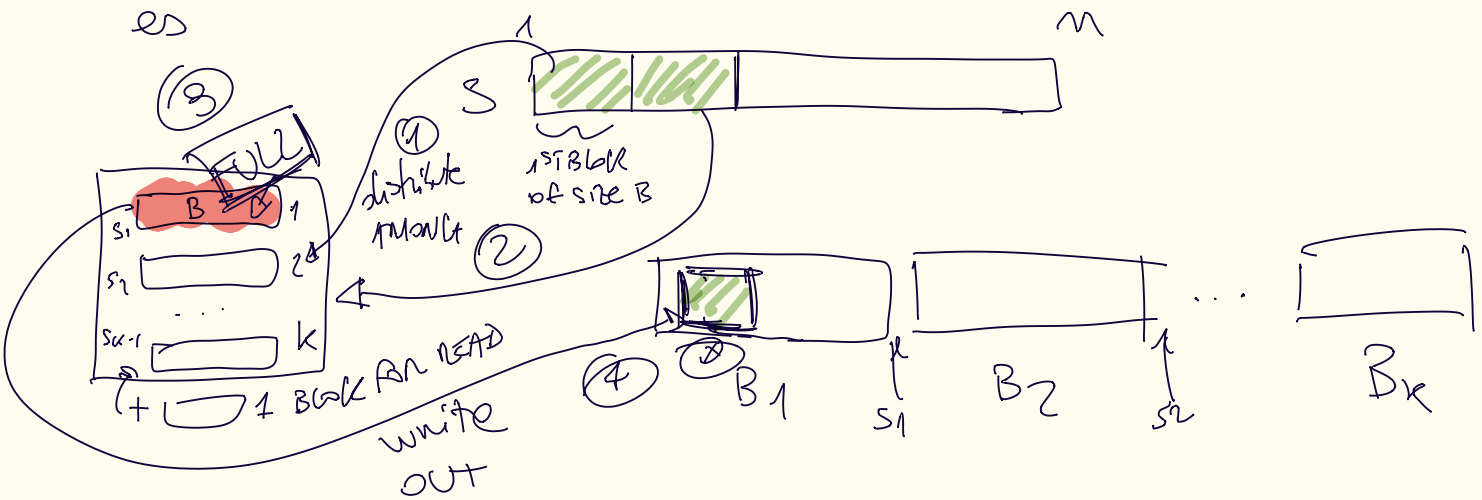


=> Buckets may be very large: ex. B_1 may Not fit in the 1ST BUCKET of the MEMORY

• So, SCAN, pay $(\frac{N}{B})$ IOs

- during a SCAN you:
  - Pick 1ST Block of S of size B, fetch in M, take the item, distribute items among buckets in memory
  - then take next block
  ...

- When a block of the memory is full
=> WRITE OUT



(5) then 1st page of memory is freed => continue...

=> JUST PARTITIONING, NOT SORTING NOW

=> I care just that the page written out is $< S_1$ (X)

- cost: $O\left(\frac{M}{B}\right)$ I/Os to SCAN S

- write out when a page if full => and I write out $O\left(\frac{M}{B}\right)$ blocks

COMPLEXITY I/Os — SCAN — RECURSION $B_1, B_2, \dots B_k$

$$T(n) = O\left(\frac{n}{B}\right) + \sum_{i=1}^{k} T(|B_i|) \quad \text{I/Os}$$

if partitions are balanced $|B_i| = \frac{n}{k}$

$$\Rightarrow O\left(\frac{n}{B}\right) + k \cdot T\left(\frac{n}{k}\right) =$$

$$= O\left(\frac{n}{B} \log_k \frac{n}{B}\right) \simeq O\left(\frac{n}{B} \log_k \frac{n}{M}\right)$$

cuz memory is $B \cdot k \le M$ — $k \le \frac{M}{B}$

SETTING $k = \Theta\left(\frac{M}{B}\right)$

$$\Rightarrow \boxed{O\left(\frac{n}{B} \log_{M/B} \frac{n}{B}\right)}$$

• To get BALANCED PARTITION, NOT k-pivots, but EXTRACT MORE (oversampling)

OVERSAMPLING => create BALANCED PARTITIONS by take more pivots than needed

(1) Pick at random: $(a+1)\cdot k - 1$ (more / required) samples from S via scan (cost $O(\frac{N}{B})$ #IOs)

• $a = \Theta(\log_2 k)$

=> so OVERSAMPLING by a factor a ← $a \cdot k$ pivots

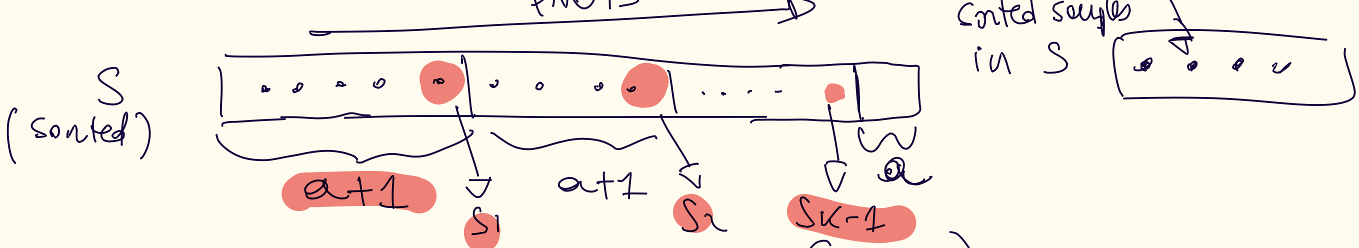
(2) Sort the SAMPLED PIVOTS

(3) WANT $k-1$ PIVOTS so:

generic pivot (s_i) → $s_i := S[(a+1)\cdot i]$

array S sorted increasingly PIVOTS

$i = 1, 2, \ldots, k-1$



=> pick 1 sample ogni $(a+1)$

=> last one

• LAST BLOCK (FONDAMENTALE): in the penultimo pivot last item e l'ultimo è grande a (altrimenti avrei un altro pivot se fosse $a+1$!)

=> so I've sampled

$$(a+1)(k-1) + a = (a+1)\cdot k - 1$$

=> I've sampled items totally equality spaced also for last one

=> SAMPLES are samples of items distribution

- if $a = 0$ => # samples = $k-1$ => no oversampling (some parts of I want to take)
  (sort is $O(k \log_2 k)$)

  if $a > 0$ => we increase sort of sorting a few IN M but $O(ak \log_2 ak)$ => but cool for balancement of partition

- How close a?

  $a = \Theta(\log_2 k)$ => we need only a logaritmic OVERSAMPLING to get BALANCED PARTITIONS (good!)

  ACTUALLY we set $a+1 = 12 \ln k$

  => for the proof

- since in our case $k = \frac{N}{B}$ $\quad a = \Theta(\log_2 \frac{N}{B})$

PROOF: TO GET BALANCED PARTITION oversample by ~~factor $\Theta(\log_2 k)$~~ SET $a+1 = 12 \ln k$

- BALANCED means: $|B_i| < 4\frac{N}{k}$

- proof by contradiction

  estimate $\mathbb{P}(\exists \text{ bucket } B_i : |B_i| \geq 4\frac{N}{k}) \leq \frac{1}{2}$ (NOT that one will slow)

  (will occur with prob. $1/2$)

=> sufficient to do twice the extraction

=> so on AVG 2 EXTRACTIONS are enough to provide that 1 is GREAT

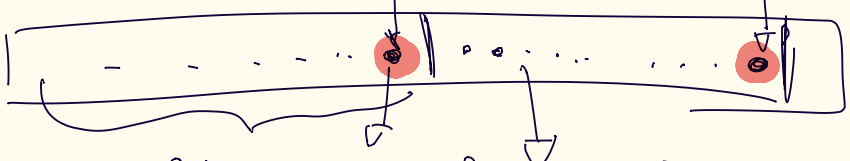
- coins: on avg every 2 I get 1 head

(like head / tail in coins) TAKEN on avg

FUNDAMENTAL PRINCIPLES

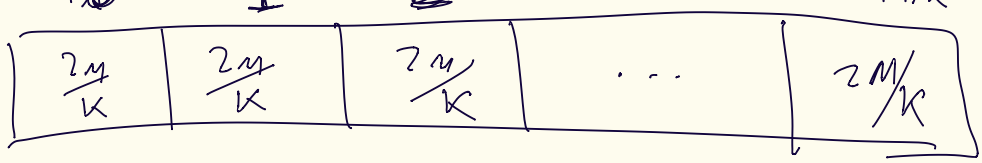
What is B_i? => (1) $B_i = \{ S[j] : S_{i-1} < S[j] \leq S_i \}$ (set of items $S[j]$, the PIVOT S_{i-1})

(2) the S_{sorted}



& other items sampled are not TAKEN

- the skipped once are: $a+1$ from S



S_{sorted} logically

ASSUM PARTITIONED logically of block $\frac{2M}{K}$

=> blocks are $\frac{M}{\frac{2M}{K}} = M \cdot \frac{K}{2M} = \frac{K}{2}$

- $|B_i| \geq \frac{4M}{K}$ ITEMS

- we set $|B_i| \geq \frac{4M}{K}$

OBSERVATION => so the Probability that a Bucket has size $\frac{4M}{K}$

=> $\exists$ some piece of t_0, t_1, t_2

TOTALLY COVERED BY a BUCKET

---

out SIZE IS $\frac{4M}{K}$ IS NOT POSSIBLE THAT B_i is included in one of $t_0, t_1, \ldots$

(at least B_i covers 1 t_j

=> $P\left(\exists B_i : |B_i| \geq \frac{4M}{K} \right) \leq P(\exists T_j$ is covered entirely by some $B_i)$ ⇓

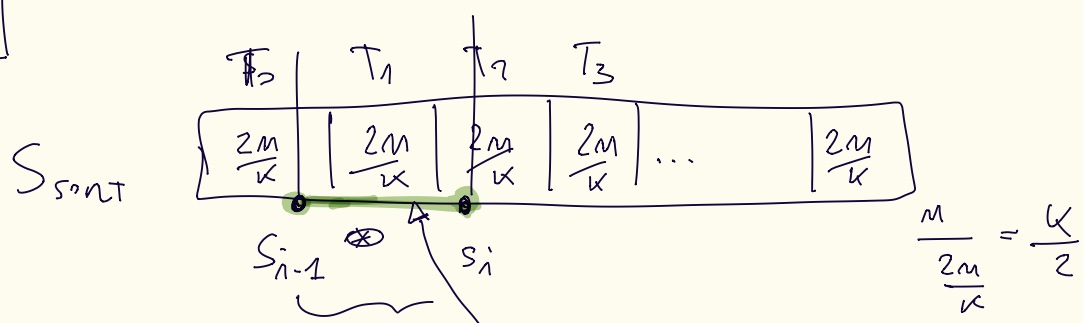
cut the other event is unique

=> B_i could be so ~~big~~ LARGE that does just 2 blocks T_j

↙ VICEVERSA IS NOT TRUE

More cuz not receiving the core ~~each~~ that exist one of the T_j (B_i maybe so long that does more than 1 block (more that exist at least one) 1, 2, 3 ...

So



(*) Let $B_i = \{ |B_i| : S_{i-1} < S[j] \leq S_i \}$

a+1 including S_i or a without

=> So samples skipped are in part in T_0, T_1, T_2

=> if T_1 is covered by B_i => for sure I've the event: (T_j in general)

#samples inside T_1 (T_j) to generate this event are $< a+1$

(2)

$$\Rightarrow \mathbb{P}\left(\exists B_i : |B_i| \geq \frac{4n}{k}\right) \leq \mathbb{P}\left(\exists t_j : t_j \text{ totally covered by } B_i\right) \leq$$ (same)

(3) $$\leq \mathbb{P}(\exists t_j : t_j \text{ contains less than } (a+1) \text{ samples}) \leq$$

UNION BOUND: P of existence is upper BOUNDED B/C:

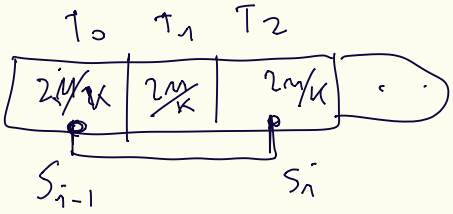
# of possibilities (I've $\frac{k}{2}$ T_j) · 1 of the event

$$\leq \frac{k}{2} \cdot \mathbb{P}(T_1 \text{ contains less than } (a+1) \text{ samples})$$

$=$

- $\mathbb{P}$ (a sample occurs in T_1) $= \frac{2n/k}{n}$ (positive events / total event) $= \frac{2}{k}$

by random sampling from S in S_{sort}



- WHAT AVG # of samples that s in T_1:

INDICATOR VARIABLE

$X_i = \begin{cases} 1 & \text{if } i\text{-th sample is in } T_1 \\ 0 & \text{otherwise} \end{cases}$

RANDOM VAR $X = \sum_{i=1}^{x} X_i =$ $x = (a+1)(k-1)$ samples

$= \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{x} X_i\right] = \sum_{i=1}^{x} \mathbb{E}[X_i] = x \cdot \frac{2}{k} =$ (items we sample $= (a+1)\cdot k - 1$)

AVG of the indicator VAR = = AVG FIRST IT occurs

$= \left[(a+1)\cdot k - 1\right]\frac{2}{k}$

$= 2(a+1) - \frac{2}{k} \geq 2(a+1) - 1 \geq \frac{3}{2}(a+1)$ ($k \geq 2$ cert pivots are $k-1$)

items $2n/k$ sampling (k increase $\Rightarrow \frac{2}{k}$ goes to 0 and all grows)

$\Rightarrow$ AVG # of items in T_1 is at least this

(*)

$\Rightarrow \mathbb{E}[X] \geq \frac{3}{2}(a+1)$ $\Rightarrow$ $a+1 \leq \frac{2}{3}\mathbb{E}[X]$

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

come back

$\mathbb{P}$ (T_1 contains $< a+1$ samples) $= \mathbb{P}(X < a+1)$

X

$\Rightarrow$ ~~apply~~ Chernoff Bound $\left(\mathbb{P}(X < (1-\delta)\mathbb{E}[X]) \leq e^{-\frac{\delta^2}{2}\mathbb{E}[X]} \right.$

$$\mathbb{P}(X < a+1) \overset{(*)}{\leq} \mathbb{P}\left(X < \left(1 - \underbrace{\frac{1}{3}}_{\delta}\right)\mathbb{E}[X]\right) \overset{\text{Chernoff}}{=}$$

$\Leftarrow \mathbb{E}[X] \geq \frac{3}{2}(a+1)$

$$= e^{-\frac{\left(\frac{1}{3}\right)^2}{2} \cdot \mathbb{E}[X]} = e^{-\frac{1}{18} \cdot \mathbb{E}[X]} \leq$$

$$\leq e^{-\frac{1}{18} \cdot \frac{3}{2}(a+1)} = e^{-1/12\,(a+1)}$$

CONDITION

- $(a+1) = 12 \ln k \quad \Rightarrow e^{-\frac{1}{12} \cdot 12 \ln k} =$

$$= e^{-\ln k} = \frac{1}{e^{\ln k}} = \boxed{\frac{1}{k}}$$

$\Leftarrow$

a (1) $\mathbb{P}(T_1 \text{ contains } < a+1 \text{ samples}) \leq \frac{1}{k}$

b 2) $\mathbb{P}(\exists B_i : |B_i| > \frac{4m}{k}) \leq \frac{k}{2}\,\mathbb{P}(T_1 \text{ contains } < a+1 \text{ samples})$

$$a+b = \mathbb{P}\left(\exists B_i : |B_i| > \frac{4m}{k}\right) \leq \frac{k}{2} \cdot \frac{1}{k} = \frac{1}{2}$$

OVERSAMPLING

- Want balanced partitions: $|B_i| < \frac{4n}{k}$ how?

  - extract more pivots then needed

    $(a+1)\cdot k + 1$ $\quad a = \Theta(\log_2 k)$

    (actually $a+1 = 12 \ln k$

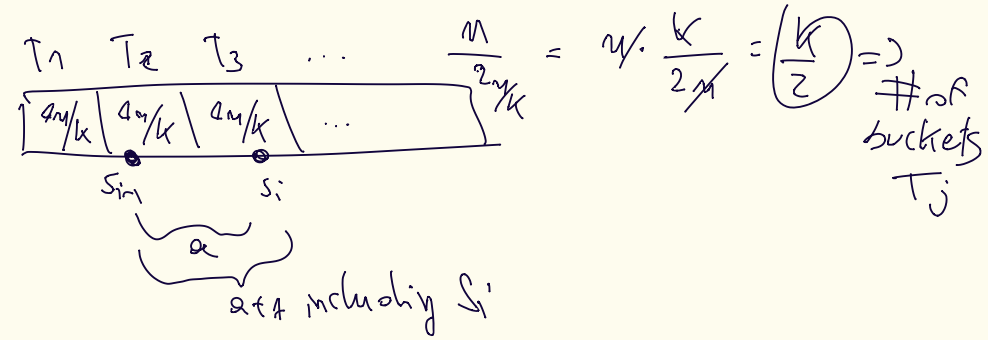
  - sort pivots (pay $ak\log_2 k$)

  - extract pivots: $S[(a+1)\cdot i]$

proof by contraddiction:

$$P(\exists\, B_i : |B_i| \geq \frac{2n}{k}) \leq \frac{1}{2}$$

idea:



$\Rightarrow |T_j| < |B_i|$

$$P(\exists B_i : |B_i| \geq \frac{2n}{k}) \leq P(\exists T_j : B_i \text{ covers entirely } T_j) \leq$$

$$\leq P(\exists T_j : T_j \text{ contains} < a+1 \text{ samples}) \leq \text{union bound}$$

$$\leq \left[\frac{k}{2} \cdot P(T_j \text{ contains} < a+1 \text{ samples})\right] \quad *$$

$\mathbb{E}[\text{sample}]$ => to evaluate this, need to evaluate

$P(\text{sample occurs}) = \frac{2n}{k} / n = \frac{2n}{k} \cdot \frac{1}{n} = \frac{2}{k}$

$\mathbb{E}[\text{sample}] = \mathbb{E}[X] = \mathbb{E}[X_i]$ where $X_i = \begin{cases} 1 & \text{if a sample occurs in } T_j \\ 0 & \text{otherwise} \end{cases}$

$$\mathbb{E}[X_i] = \mathbb{E}\left[\sum_{i=1}^{x=(a+1)\cdot k - 1} (X_i)\right] = \sum_{i=1}^{x} \mathbb{E}[X_i] =$$

$$= \left[(a+1)\cdot k - 1\right] \cdot \frac{2}{k} = 2(a+1) - \frac{2}{k} \geq 2(a+1) - 1 \geq \frac{3}{2}(a+1)$$

$\Rightarrow \mathbb{E}[X] \geq \frac{3}{2}(a+1)$

$\frac{2}{3}\mathbb{E}[X] \geq (a+1)$

$(a+1) \leq \frac{2}{3}\mathbb{E}[X]$

$\otimes \Rightarrow \dots \leq \frac{k}{2} \cdot \mathbb{P}(\underbrace{T_j \text{ contains}}_{X} < a+1 \text{ samples}) =$

$= \frac{k}{2} \cdot \mathbb{P}(X < a+1 \text{ samples}) = \frac{k}{2} \cdot \mathbb{P}(X < \frac{2}{3}\mathbb{E}[X]) =$

$= \frac{k}{2} \cdot \mathbb{P}(X < 1 - \frac{1}{3}\mathbb{E}[X]) \overset{\delta}{=} \leq \frac{k}{2} \cdot e^{-\frac{(\frac{1}{3})^2}{2} \cdot \mathbb{E}[X]} =$

(Chernoff bound)

$= \frac{k}{2} \cdot e^{-\frac{1}{18}\cdot \mathbb{E}[X]} \leq \frac{k}{2} \cdot e^{-\frac{1}{18}\cdot\frac{3}{2}(a+1)} =$

$= \frac{k}{2} \cdot e^{-\frac{1}{12}(a+1)} = \frac{k}{2} \cdot \frac{1}{e^{\frac{1}{12}(a+1)}} = \frac{k}{2} \cdot \frac{1}{e^{\frac{1}{12}\cdot 12 \ln k}} =$

$= \frac{k}{2} \cdot \frac{1}{k} = \frac{1}{2}$ ✓