String Sorting

N = # strings

D = Total length of distinguished prefix

- variable length = ℓ_{max}
- $\sigma = |\Sigma|$ alphabet size.

qsort (A, N, sizeof (char *), strcmp)

↳ thus deploying the power of a cmp-based sort procedure

PROBLEM Time cost = $\Theta(N \ell \log N)$ with many RANDOM mem accesses.
ℓ = avg-len

long strings, same diff ⇒ BAD ANALYSIS

cmp from scratch, always.

NOTE $\Sigma \xrightarrow{\text{mapped}} \{0, 1, 2, \dots, \sigma - 1\}$

strings ⟶ numbers in base σ.

INT. Keys

BASIC BLOCK

- K Sort ≅ Counting sort (sort using lists)
  1) Keys are integers (SMALL) in $\{0, 1, \dots, K-1\}$ ($K = \sigma$ in strings)
  2) Key k goes to bucket $b[k]$.

  $\frac{\text{Time}}{\text{Space}} = O(n + K)$ $\boxed{\text{OK if } K = O(n)}$ ⇒ small alphabet

Property STABILITY + optimal if

- Radix Sort
  - recursive → LSD (needs all keys have equal length for logical padding)
  - split & recurse → MSD

IDEA ① strings as binary sequences 01011010 1

4

16

256

base

② strings as sequence of groups of bits, and apply K sort

LSD

d = # groups

K = max digit = symbol

$d(K + n)$ Time and $(K + n)$ space

$S = \langle 017, 042, 666, 007, 111, 911, 999 \rangle$

111 < 017 first round
017 < 111 third round

<u>proof.</u> (induction) S_{i-1} = sorted wrt $(0, \dots, i-1)$ digits

We sort by the i-th digit, if two numbers have it different it implies the sorted order and we are done; if it is equal, the stability preserves the order.

Can we choose d and K? We can control them!!

$K \Rightarrow$ groups of $\log_2 K$ <u>bits</u> $\Rightarrow d = \dfrac{\ell}{\log_2 K}$

$$\text{Time} = \frac{\ell}{\log_2 K}(K + n)$$

large $K \Rightarrow$ few groups $\Rightarrow$ costly Ksort
small $K \Rightarrow$ many groups $\Rightarrow$ many phases

breakeven point $K = \Theta(n)$, because this is in any case the cost of one-phase of Ksort.

$\text{Time} = \dfrac{n\ell}{\log_2 n}$ $\quad$ Space $= n\ell$

$\boxed{\text{Comparison-based } LB = n\ell \log n}$

~~Proceeds left→right by distributing keys according to their symbols.~~

$\boxed{\text{MSD RADIX}}$ see next
counted as a function of D
$O(D + n\sigma)$

~~Problem 1: Homogeneous buckets in the recursion~~
~~Problem 2: When buckets are small Ksort is inefficient~~

$|\sigma\text{-partition}|$

$\boxed{\text{Multi-key Quicksort}}$ optimal bound (in the comp-model)

(LB) $n \log n + D$

$\boxed{\text{Key idea}}$ Ternary partition
↳ bucket formation is made easy

al	p	habet
al	i	gnment
al	l	ocate
al	g	orithm
al	t	ernative
al	t	ernate
=		← SORTED

yet potentially unsorted

MultiKey QS (R, ℓ) — R = set of strings of common prefix of length ℓ

① if $|R| \leq 1$ return R

② choose pivot $p \in R$

③ $R_< = \{ s \in R \mid s[\ell+1] < p[\ell+1] \}$;

$R_= = \{ s \in R \mid s[\ell+1] = p[\ell+1] \}$

$R_> = \{ s \in R \mid s[\ell+1] > p[\ell+1] \}$

④ A = MultiKey QS $(R_<, \ell)$

B = MultiKey QS $(R_=, \ell+1)$ advance

C = MultiKey QS $(R_>, \ell)$

⑤ Return $A \cdot B \cdot C$

- Correctness is obvious
- Cost = Count the number of comparisons to execute Step ③
  - Case $s[\ell+1] \neq p[\ell+1]$
    - assume perfect choice of pivot → R is halved → total charge on s is $\log n$

Too strong ⇓ Random

→ # Total comparisons = $n \log n$

  - Case $s[\ell+1] = p[\ell+1]$
    - advance of one char → no more than D total adv

Integer Arithmetic
MSD Radix Sort

What about using σ buckets, and not just 3?

deploy INTEGERS

① Distribution takes $(n_i + \sigma)$ and involves all strings for at most D steps $(\Sigma$ over $n_i \Rightarrow D)$ ok if $n_i > \sigma$ Recall for next consideration

② Every non-trivial partitioning (not all equal chars) takes $O(\sigma)$ time and this occurs no more than n times. (Non trivial partition is a tree with fan-out ≥ 2 and n leaves).

$D + n\sigma$

$$\boxed{\begin{array}{l} \text{Multikey QS} = D + n\log n \\ \text{MSD RadixS} = D + n\sigma \end{array}}$$

Understand algorithms to "combine" them:

① LSD = Ksort + repeat

<u>now</u> when the #strings to partition is $n_i \le \sigma$ use Multikey since it is Σ-indep.

$\Longrightarrow$ $D + n\log\sigma$

- We are avoiding the σ-cost of small-bucket formation which is too much when $n_i \ll \sigma$.

cmp-based	$D + n\log n$	multikey QS	OPTIMAL
integer	$D + n\log\sigma$	MSD + Multikey	
constant	D // $D + n\sigma = O(D)$	MSD Radix	OPTIMAL

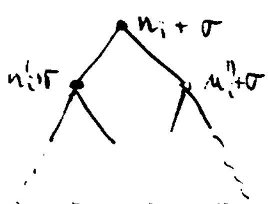
OBSERVATION

Cost Multi-key → #distinguishing chars $\Longrightarrow D$
→ #n log#n

when using COMBINATION. 2° term is $\sum n_i \log n_i \le n \log n$ (σ)

Cost MSD Radix

- When $n_i \ge \sigma \Rightarrow$ optimal cost $\Theta(n_i) \Rightarrow D$
- When $n_i < \sigma \Rightarrow$ poor cost $\Theta(\sigma)$ which we pay at each visit



$n_i + \sigma \le (1+1)\,\sigma$

$\Rightarrow n_i + 2\sigma \le (1+2)\,\sigma$

$\Big\}\ (h + \Theta(n_i))\,\sigma \Downarrow \Theta(n_i\,\sigma)$

n_i leaves $\Rightarrow \Theta(n_i)$ nodes

SUMMING OVER ALL n_i

$\Downarrow$

$O(n\,\sigma)$

Lecture #3

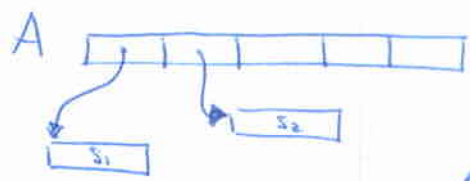
Strings in a set

- Sorting → search data structures
  → duplicate removal
  → join in DBs

OTHER NOTES

- Search + Storage $D = \{abaco, abba, paolo, pattern, patty\}$

① Array of string pointers, randomly allocated strings



∘ $P \times \log N$ Time cost / $\log N$ I/Os / Space: $L_{TOT} + N + N \log N$

random memory accesses

∘ Interpolation Search ⇒ $\log \log N$ → $mid = low + \frac{(K - A[low])}{A[high] - A[low]} * (high - low)$

Assumes random distribution of keys in the range $[A[low], A[high])$, given that there are $(high - low)$ keys

- String ≡ number in base σ
- recent Δ: gap ratio $= \frac{\max (x_i - x_{i-1})}{\min (x_i - x_{i-1})}$

① bin B_i represents a range of size $\frac{x_n - x_1}{n}$ (equally part.)

② binary search on B_J where $J = \frac{K - x_1}{x_n - x_1}$

Cost is $O(\log \Delta)$ → $\Delta = $ polylog n per unif distr.

Claim $\max x_i - x_{i-1} \geq \frac{x_n - x_1}{n}$ (avg distance)

$\min x_i - x_{i-1} \stackrel{def}{=} \frac{\max x_i - x_{i-1}}{\Delta} \geq \frac{x_n - x_1}{n \Delta} \Rightarrow |B_i| \leq \Delta$

- How to save $\simeq \log_2 B/\ell$ cache/block misses and be output-sensitive?

  Time/IO Saving

  → write strings contiguously

  → [↳ last $\log_2 B/\ell$ accesses are in the same block, both for pointers & strings.

- How to save space?

  ⇒

  ⇒ Blocking : reduce # pointers $N \longrightarrow L/B$ [1 byte more or IO]

  cons → This requires Block scanning but page already fetched

  PRO → Front-coding (35÷40%)

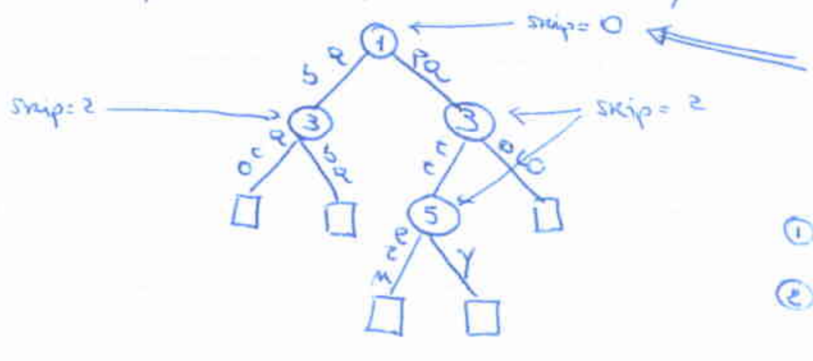
$$\text{Time} = \log_2 \frac{L}{B} + 1 \text{ I/Os} \qquad \boxed{\ell \le B}$$

$$\text{Space} = \underbrace{L_{TOT}}_{\text{strings}} + \frac{L}{B}\underbrace{4}_{\text{pointers}} + \underbrace{N}_{IO}$$

- Can we speed-up prefix-searches? TRIE

  → uncompressed : one node per char

  → compressed : squeeze out the unary path. [need SKIP or DIGIT NUM]



① P I/Os (indep. N)

② 12÷20 N bytes

internal nodes ≤ N-1 , # leaves = N

- branching impacts.

  ① binary impl = leftmost, child + sibl ⇒ 2 pointers × internal node but O(σ) branch time

  BE BACK ON THIS

  ② hash table ⇒ O(1) we avg branch time, O(N) space

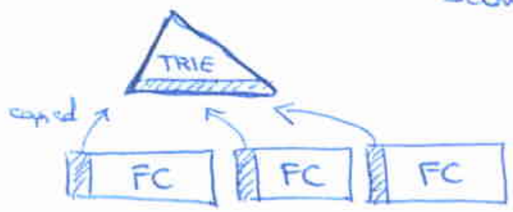
  ③ array ⇒ O(1) branch time, O((N-1) · σ) space !!

What about I/O-issues?

Two-level memory

- ~~Node~~ 1^ level implementation impacts on the I/O-performance
  → larger ~~nodes~~ blocks → fewer cached → more I/O-misses.

- Given 2-levels → Trie on a set sample
  → scan on the rest (buckets)

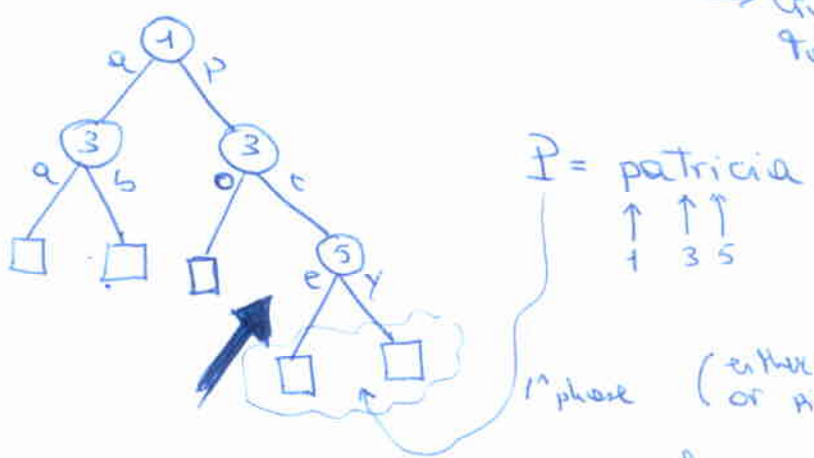


① TRADE-OFF: ↑ Smaller buckets → bad FC but faster search

② COPIED SPACE: Much because we need to keep entire strings to solve edge labels

- Two improvements
  - Trie → drop edge labels + strings → more "copied" strings
    → difficult search
  - Buckets + FC → no bucketing but FC on all (more robust)

BLIND TRIE described in the attached paper but it is not described how to lex-search [MY NOTES]
→ crucial to find where to proceed.



(either we reach a leaf or pick one descending)

$Lcp(P, \text{"patty"}) = \text{pat}\boxed{t} \Longrightarrow P < \text{patty}$

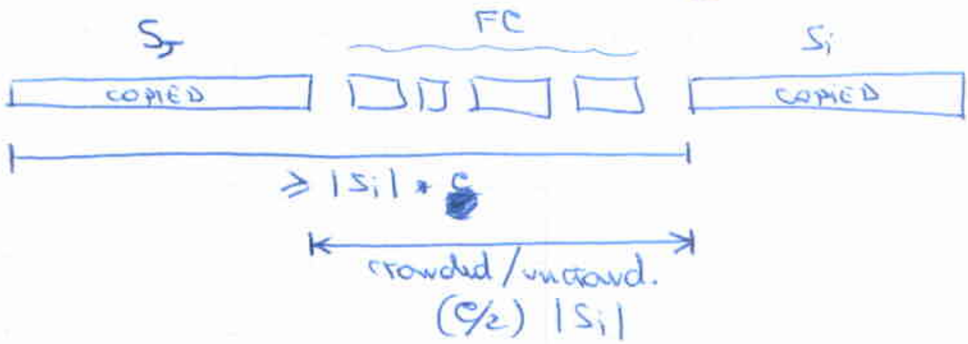
SPACE proportional to sampled number and not to length (thousands of strings in L2)

$\boxed{\text{TIME/IO}}$ 1 string check → buckets (obvious)
→ LPFC (cool idea!!)

① Front-code a string S iff the cost of decoding it $\leq c|S|$
[# previous explicit char to scan in order to decode]

② Decoding procedure is not affected

TEO $\forall c > 2$, space $= (1+\varepsilon)$ FC, Time $\frac{|S|}{\varepsilon}$, where $\varepsilon = \frac{2}{c-2}$



- Decompose the sequence of copied into (UNCROWD) · (CROWD)*
- if S uncrowded, it is preceded by at least $\frac{c}{2}|S|$ chars of FC
- if S_i crowded, then $|S_j| \geq \frac{c}{2}|S_i| \Rightarrow |S_i| \leq |S_j| \frac{2}{c}$ (geometric)

RUN LEN

$$\cancel{\text{crowded}} < |\text{UNCROWD}| \cdot \sum_{x=0}^{\infty} \left(\frac{c}{2}\right)^x = |\text{UNCROWD}| \cdot \frac{1}{1-\frac{2}{c}}$$

- Charge this cost onto the $\frac{c}{2}|\text{UNCROWD}|$ chars that precede the run
  $\Rightarrow \frac{2}{c-2}$ per char cost $\Rightarrow$ this is ε
  (first run has no preceding chars but uncrowded is FC-chars)

$\boxed{\text{NOTE}}$ This allows also to drop completely the bucketing and thus result more robust.