Adv. Data Structures

Tiered Bit Vectors (cont)

Recap

- Schedule page is fixed
- HW – still coming
- No class on Friday

Next data structure:

What if we restrict inputs?

Goal: Have a bounded set of possible elements, & want to store which ones are in my set.

ie: subset of 32-bit integer

or list of names (all $\leq$ 30 chars)

Operations

- insert(x)
- find (x)
- delete(x)
- max / min
- successor(x)
- predecessor (x)

Tiered Bitvector:

Put a summary on top of the vector. u/B

OR the bits

1	0	1	0	1	1	0	0
00100010	00000000	00011000	00000000	00000100	11110111	00000000	00000000

B

How to search/update:

succ: check for next value in x's block
if none, move up + scan upper tier (until 1)
Move down + find min in low block

Runtime: $B + \frac{u}{B} + B$
$= O(B + \frac{u}{B})$

How to find "best" value for B?

Calculus!

Minimize $O(\frac{U}{B}+B)$:

$$\frac{d}{dB}(UB^{-1}+B) = 0$$

$$\Rightarrow -UB^{-2}+1=0$$

$$1=UB^{-2} \Rightarrow B^2=U$$

Solve for B:

$$B=\sqrt{U}=U^{1/2}$$

Runtime: $O(B+\frac{U}{B})$

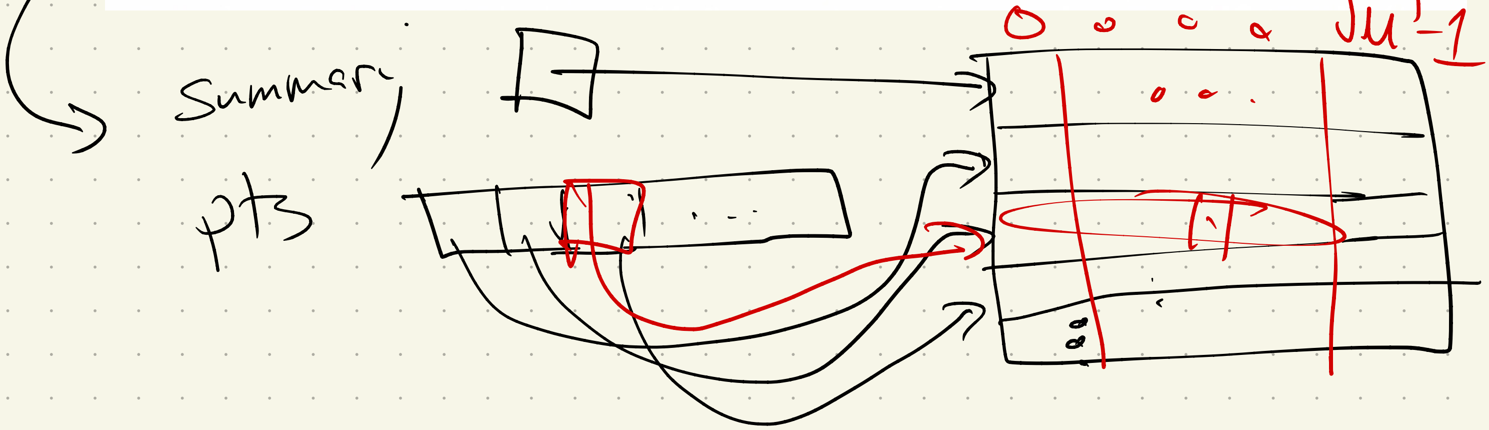
$$=O(\sqrt{U}+\frac{U}{\sqrt{U}})$$

$$=O(\sqrt{U})$$

Helpful view:

Think of this as a main vector & a "summary" vector.

1	0	1	0	1	1	0	0
00100010	00000000	00011000	00000000	00000100	11110111	00000000	00000000



To lookup, check $\left\lfloor \frac{x}{u^{1/2}} \right\rfloor^{\text{th}}$ bit vector in spot $x \bmod u^{1/2}$

Ex: slot 43 — 5th vector, spot 3

here, $B = 8 = \sqrt{u}$ so $u = 64$

Insert(x):

- insert $x \bmod U^{1/2}$ into $\lfloor \frac{x}{U^{1/2}} \rfloor^{th}$ vector
- and $\lfloor \frac{x}{U^{1/2}} \rfloor$ into summary

Similarly:

Min():

- call min on summary
- call min on result

Delete(x):

set $x \bmod U^{1/2}$ in $\lfloor \frac{x}{U^{1/2}} \rfloor^{th}$ vector to 0

if $\lfloor \frac{x}{U^{1/2}} \rfloor^{th}$ vector is empty,

set $=0$ in summary

What about deleting?

1	0	1	0	~~1~~ 0	1	0	0
001000~~1~~0	00000000	00011000	00000000	00000~~1~~00	11110111	00000000	00000000

10   0

- 1 delete in bottom
  $O(1)$
- Is-empty
  - if empty, delete top
    $(0 \to 1)$

Runtime:

$O(\sqrt{u})$

Analyze:

isEmpty: Need to call
isEmpty on summary

u

$\sqrt{u}$

Lookup:

To lookup, check $\left\lfloor \frac{x}{u^{1/2}} \right\rfloor$ th

$\sqrt{u}$ bit vector

in spot $x \bmod u^{1/2}$

Both give recurrence:

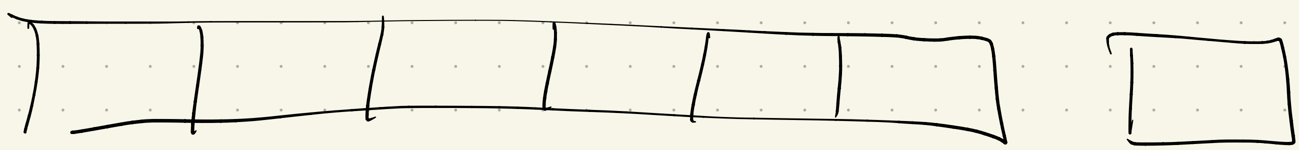
Let $T(x)$ = runtime on
universe of size x

Then

$$T(u) = T(\sqrt{u}) + 1$$

$$T(2) \leq 1$$

So: tiering helped! ($U \to \sqrt{U}$)
Can we improve even more?



$\sqrt{U}$ blocks each $\sqrt{U}$ size

summary, size $\sqrt{U}$

Recurse!

For each block of size $\sqrt{U}$, apply the same construction:

$U^{1/4}$ size blocks, plus summary

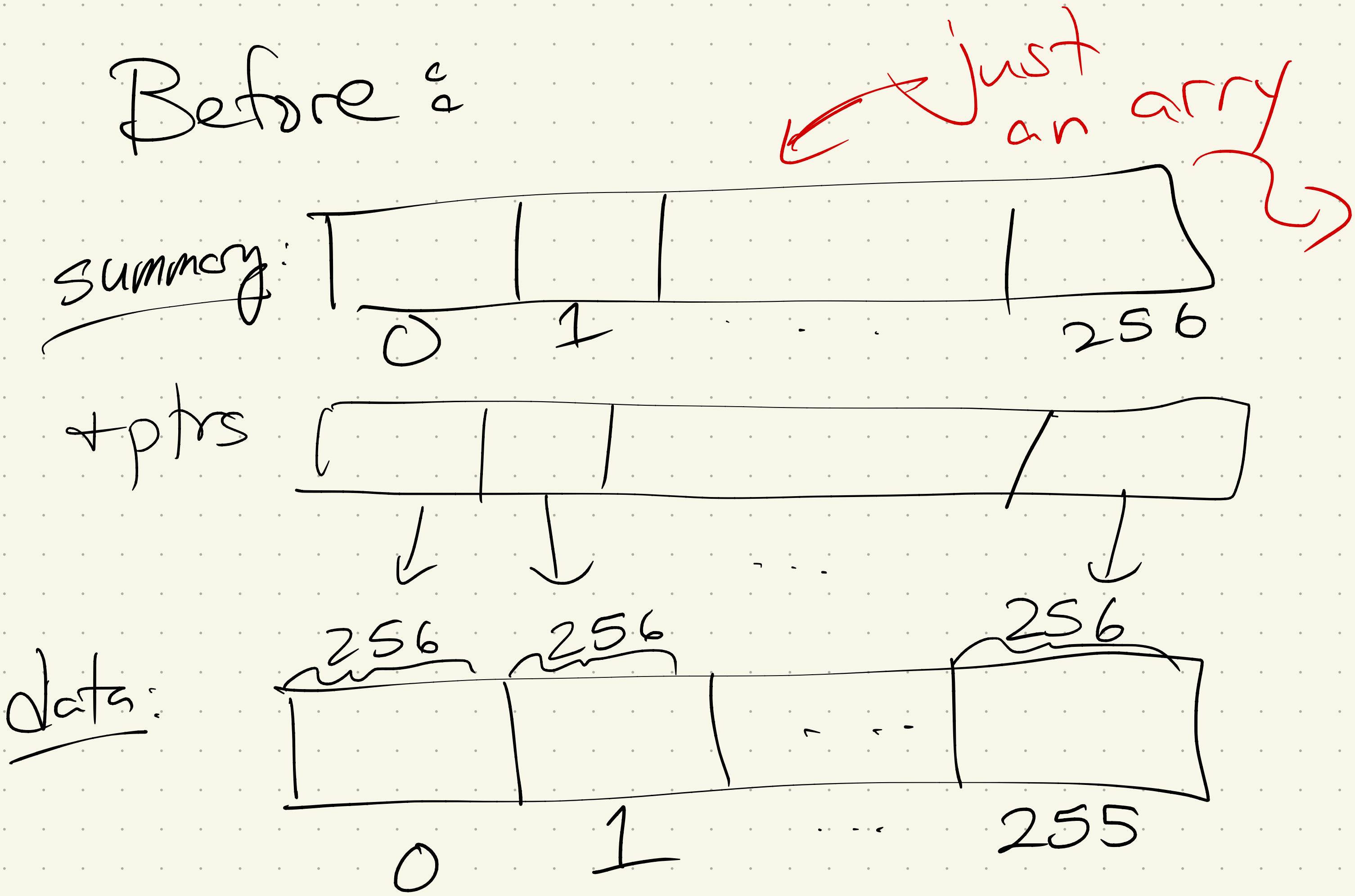
Picture:

Suppose we have ASCII!

$U = 65{,}536$

$\sqrt{U} = 256 \;(\& \; U^{1/4} = 16)$

Before:

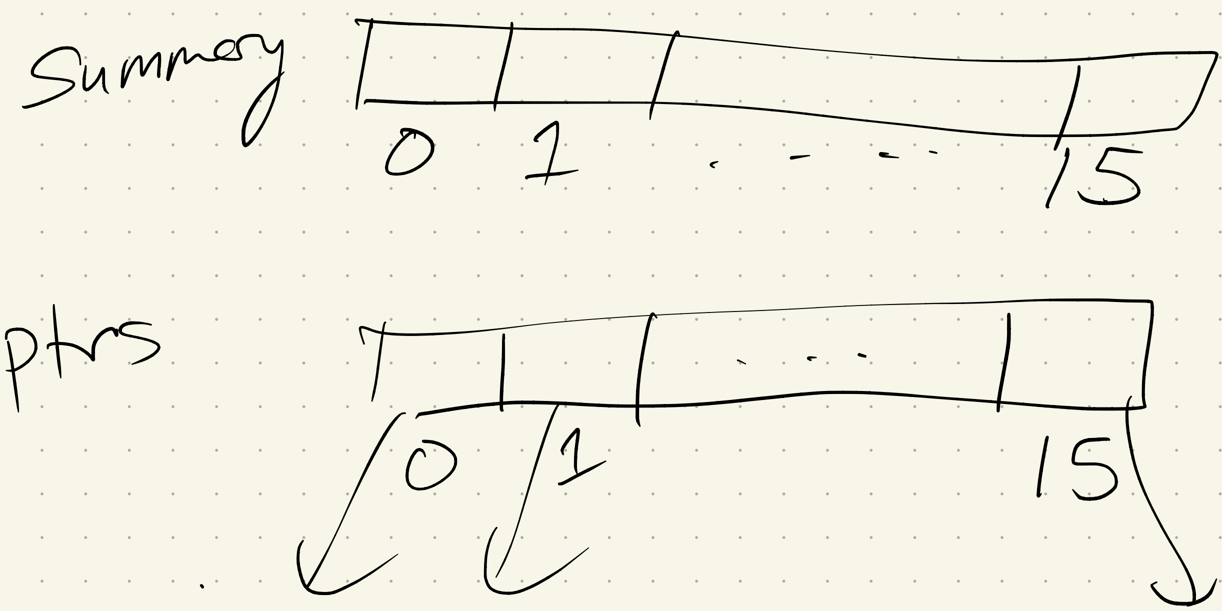


Change: recursively store summary & each level

Summary + data blocks:
each size 256

Apply same construction:

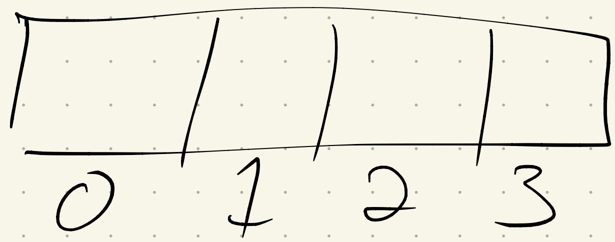
$$\sqrt{256} = 16$$

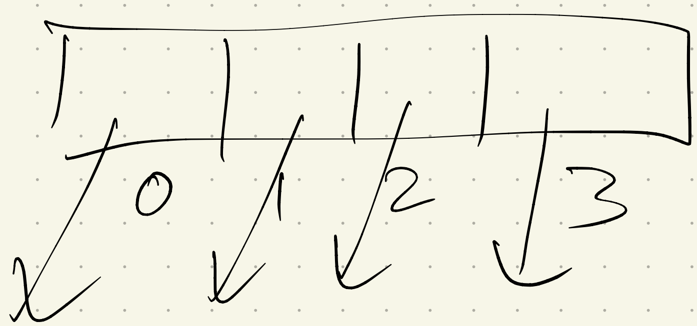


Each of <u>those</u> is size 16.

$$\sqrt{16} = 4$$

So:





(& stop when $\underline{\leq 2}$)

Master Thm

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

$$n^{\log_b a}$$

or $f(n)$

To solve: domain transformation trick

$$T(u) = T(u^{1/2}) + 1$$

Set $u = 2^k$

$\Rightarrow k = \log_2 u$

$T(\sqrt{2^k})$

$(2^k)^{1/2}$

Then: Let $S(k) = T(2^k)$

$$S(k) = S(k/2) + 1$$

solves

$= O(\log_2 k)$

$= \log \log u$

for lookup + IsEmpty

Insert & Min/Max:

2 recursive calls on smaller size

$$I(u) = 2I(u^{1/2}) + 1$$

Substitute again:

$$u = 2^k \Rightarrow k = \log_2 u$$

$$J(k) = I(2^k)$$

$$J(k) = 2J\left(\frac{k}{2}\right) + 1$$

$$= k$$

$$= O(\log_2 u)$$

Delete:

≤2 recursive delete calls, plus one isempty:

$$D(u) \le \underbrace{2D(u^{1/2})}_{\text{recursive calls}} + 1 + \underbrace{O(\log\log u)}_{\text{isempty}}$$

dominates

see 2 slides ago

$$= O(\log u)$$

Successor/Pred:

Each time:

- One recursive call
- One min/max call

$$P(U) = P(U^{1/2}) + 1 + O(\log U)$$

$$\Rightarrow P(U) = \log U$$

So takeaway:

$O(\log U)$ worst case

$\underline{O(\log \log U)}$ lookups

vs: $O(U) + O(1)$ / $O(\sqrt{U})$

but:

U is size of universe!

If $n \geq \log U$,

we beat BST in lookups!

(since $\log n \geq \log \log U$)

<u>van Emde Boas tree:</u>

A slight modification of our tiered bitvectors.

Besides summary & $\sqrt{u}$ pointers to next level, we'll also store min & max separately. (at each level)

Lookups are unchanged (except we also check if target is min or max)

Important: min & max are <u>only</u> stored in special field.

Insert :

Delete:

Runtimes :