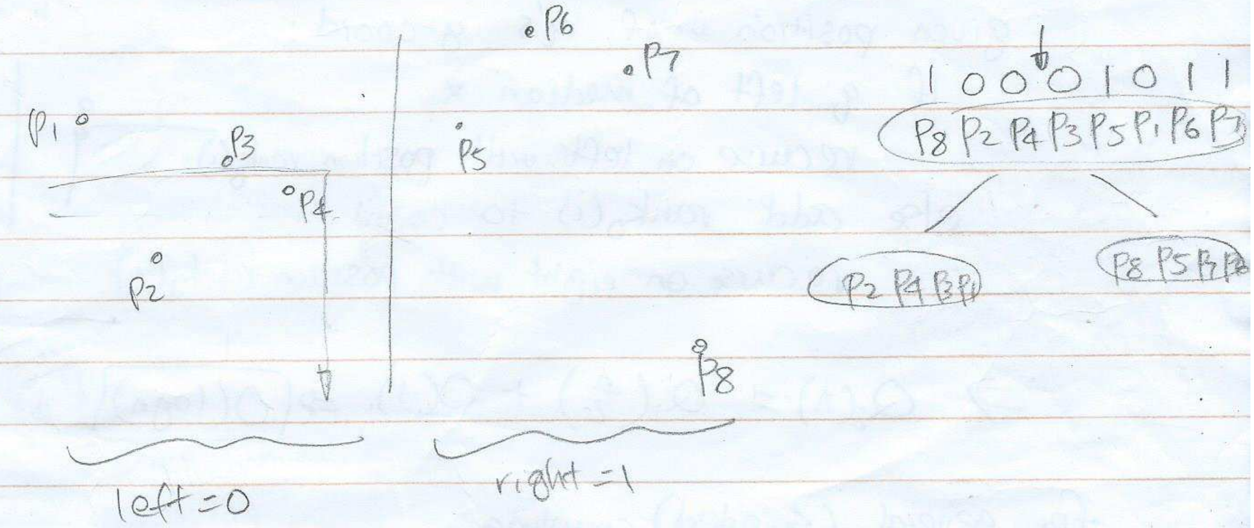
<u>Improving Space by Bit Packing</u> (Chazelle '88)

for 2D Counting

idea - modify range tree

idea - replace list of y-coords by list of bits



<u>Standard Assumption</u> RAM model with fixed word size w where $w \geq \log n$ bits

<u>Fact</u> can store array A of n bits using $O(n/w)$ space (in words)

s.t. can compute $\text{rank}_0(i) = \#\ 0\text{'s in } A[1..i]$

$\text{rank}_1(i) = \#\ 1\text{'s}$ " "

Pf. divide array into n/w words

$$\underbrace{\boxed{}\ \boxed{}\ \boxed{}}_{c_j}\ \boxed{}\ \cdots\ \boxed{}$$

i

Precompute $c_j = \#\ 0\text{'s in first } j \text{ words}$

Count # 0's in a word by a nonstandard word op.

(if $w = \log n$, can build table with all answers in $O(n)$ space.)

□

Rmk: can further reduce space to $n/w + o(n/w)$ by <u>succinct DS</u>

Space for $S(n) = 2\,S(\frac{n}{2}) + O(\frac{n}{w})$

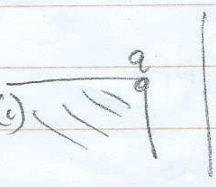
$\Rightarrow O(\frac{n}{w}\log n) = \boxed{O(n)}$ by setting $w = \log n$

Query Alg'm, for dominance (2-sided) counting:

given position i of q's y-coord:

if q left of median x

recurse on left with position $rank_0(i)$



else add $rank_0(i)$ to count

else recurse on right with position $rank_1(i)$

$\Rightarrow Q_2(n) = Q_2(\frac{n}{2}) + O(1) \Rightarrow \boxed{O(\log n)}$

for general (4-sided) counting:

reduce to dominance by subtraction/addition

$\Rightarrow \boxed{O(\log n)}$

(Rmk - related to wavelet trees

- reporting is costlier $(O(\log n + k\log n))$)

Improving query time, by bit tricks?

Assumption: input values are integers in $[U] = \{1, \ldots, U\}$

word RAM model where each word has fixed size w.

with $w \geq \log n$ (index/ptr fits in a word)

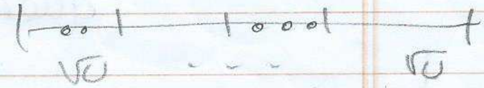
$w \geq \log U$ (input value " " ")

Warm-Up: 1D Pred/Succ Search

Method 1: van Emde Boas (vEB) tree

divide $[U]$ into $\sqrt{U}$ chunks of size $\sqrt{U}$

recurse inside each chunk



let $D = \{$ (indices of) all nonempty chunks $\}$

recurse in D

store min & max of each nonempty chunk

Query alg'm, given pt q:

$i =$ index of chunk containing q ← $O(1)$ time by arithmetic

if $i \in D$, recurse inside chunk

if $i \notin D$, recurse in D.

$O(1)$ time by hashing

$$Q(U) = Q(\sqrt{U}) + O(1)$$

let $U = 2^\ell$ $\Rightarrow$ $Q'(\ell) = Q'(\ell/2) + O(1)$

$(\sqrt{U} = 2^{\ell/2})$ $\Rightarrow$ $Q'(\ell) = O(\log \ell)$

$\Rightarrow$ $Q(U) = \boxed{O(\log\log U)}$

Space: each element stored in ≤ 2 recursive substructures

$$s(U) = 2\, s(\sqrt{U}) + O(1)$$

space per element

total space $O(n\, 2^{\log\log U}) = O(n \log U)$

Space reduction trick:

select subset containing every b^{th} element

build vEB for subset set $b = \log U$

$\Rightarrow$ space $O(\frac{n}{b}\log U + n) = \boxed{O(n)}$

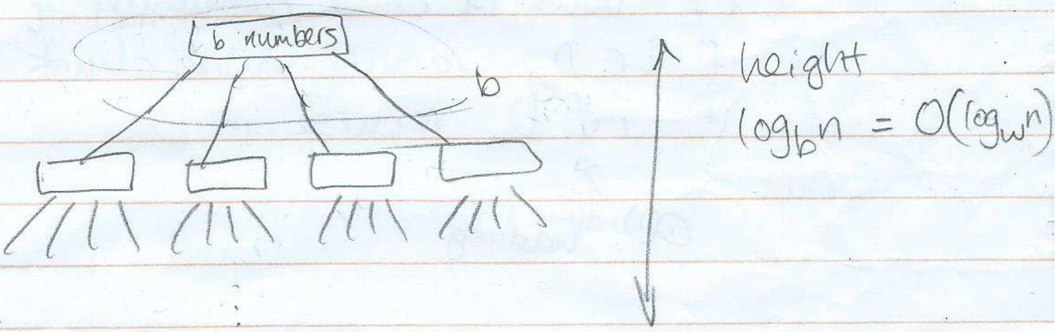
query time $O(\log\log U + \log b) = \boxed{O(\log\log U)}$

finish by binary search

Method 2: Fusion Tree (Fredman, Willard '90)

$O(n)$ space, $O(\log_w n)$ time

rough idea - b-ary search tree with $b \approx \sqrt{w}$

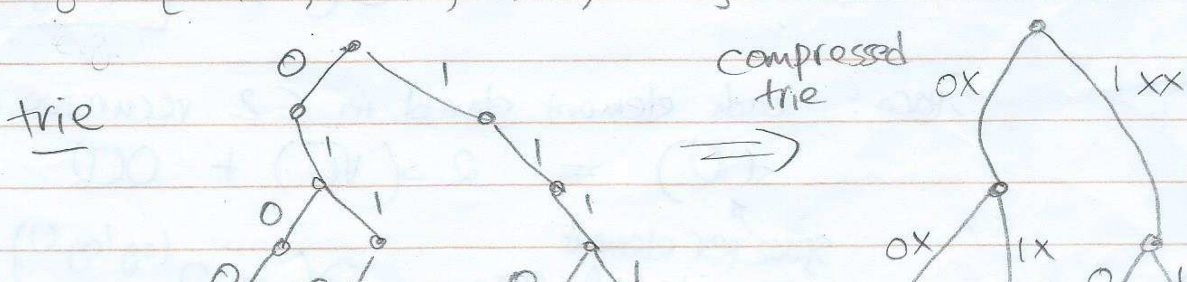


need to search among b numbers in $O(1)$ time.

how? compress b numbers into a single word

then use a nonstandard word op

e.g. {0100, 0110, 1110, 1111}



$O(bw)$ nodes: bad! $O(b)$ nodes

compressed trie encodable in $O(b \log w)$ bits $\ll w$

to query q:

follow path to get element z — may be wrong

e.g. $q = 0011$

$z = 0110$

find first bit where q & z differ

(avoiding nonstandard ops is complicated!)

Method 3: Combine!

$$Q(n) = O(\min\{\log\log U, \log_w n\})$$

$$\leq O\left(\min\left\{\log w, \frac{\log n}{\log w}\right\}\right)$$

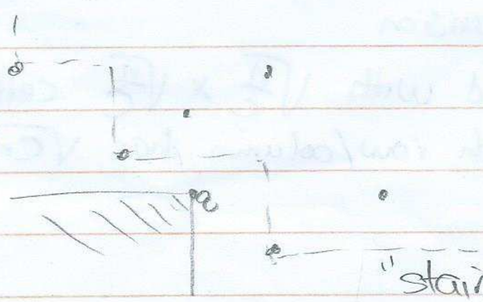
$$\leq \boxed{O(\sqrt{\log n})}$$

beats $\log n$ for any U! (near-optimal)

Back to 2D ...

Advanced Method 1

dominance (2-sided) emptiness



reduce to pred search in x

$\Rightarrow$ $O(n)$ space, $O(\log\log U)$ time