6.851 ADVANCED DATA STRUCTURES

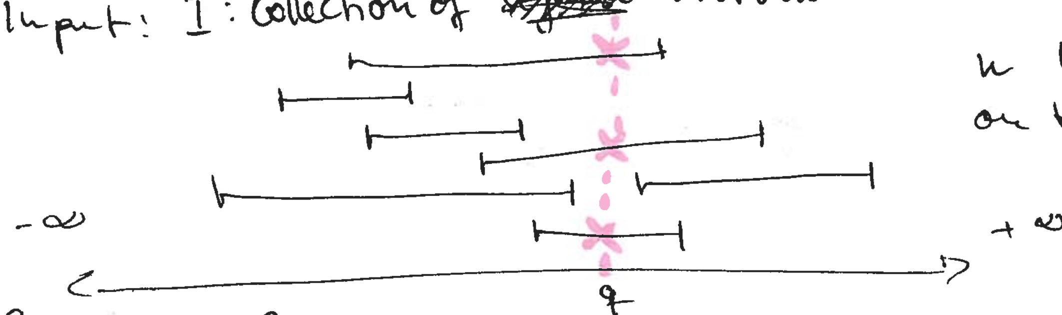
(ANDRÉ SCHULZ' NOTES)

LECTURE 4

STABBING QUERIES FOR SEGMENTS

(Vertical line Stabbing)

Input: I : collection of ~~segment~~ intervals

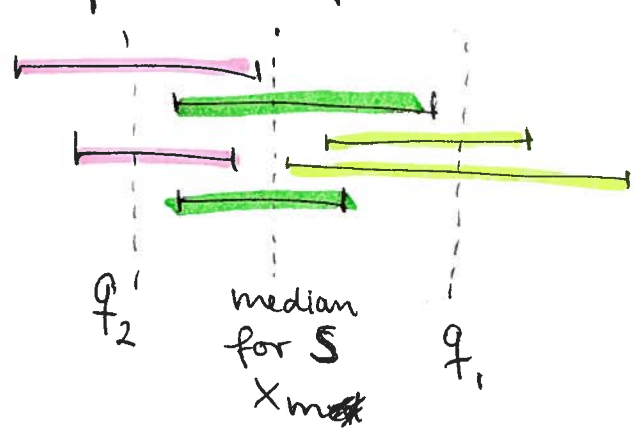


n intervals I on the real line

Query : Given a point $q \in \mathbb{R}$, report all ~~segments~~ intervals containing q

1st Approach : Interval-Trees

Let $S = \{x_1, x_2 \ldots\ldots x_{2n}\}$ denote the endpoints of the n segments



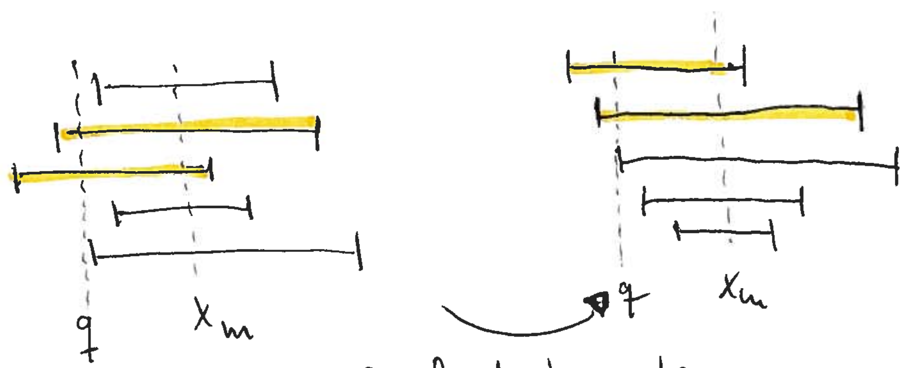
I_ℓ: red: complete interval left of x_m

I_m: yellow: complete intervall right of x_m

I_r: green: the rest

- If q is left of x_m (e.g. q_2) we can ignore the yellow segments
- If q is right we can ignore the red segments

We could use this behavior to search for the stabbed intervals, but how to handle I_m?



Assume $q < x_m$

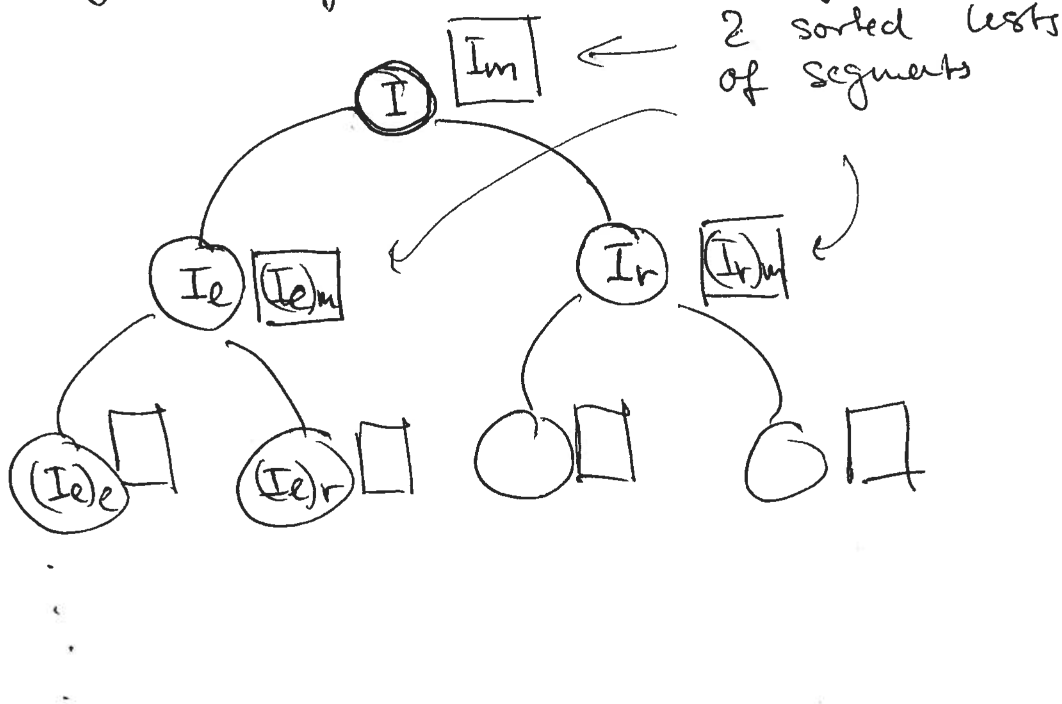
Sort intervals by the left endpoints!

q stabs the first k elements and that is it.

→ reporting alle (sorted) segments and stop when $q <$ ~~a~~ , (~~a~~, b) segment

→ do the same with the segment endpoints if $q \geq x_m$

This way we get the following DS



Query : • Search for q and check all the sorted lists on the nodes of the search path

Query time : searching : $O(\log n)$, reporting : $O(k)$ $\Big\}\ O(\log n + k)$

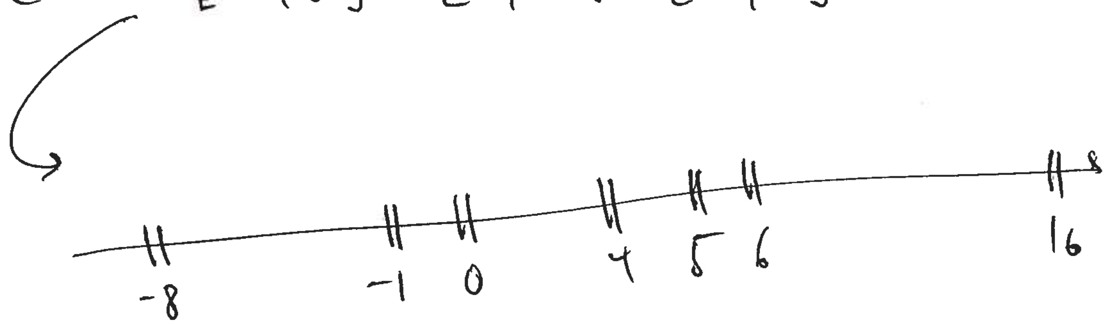
Space : • Every segment is stored ~~once~~ twice
• the tree needs $O(n)$ space
$\Rightarrow O(n)$ space

preproc : Sorting for the endpoints first gives $O(n \log n)$ preproc. time

Alternative : Segment Trees

Idea: Split the real line in elementary intervals

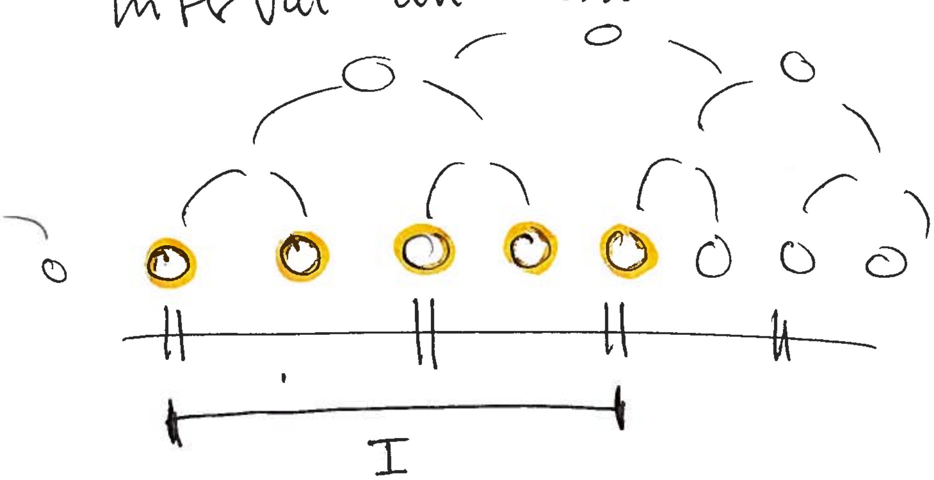
Example $[-8,6]$ $[0,16]$ $[4,5]$ $[-1,2]$



$(-\infty,-8)$ $[-8,-8]$ $(-8,-1)$ $[-1,-1]$ $(-1,0)$ $[0,0]$ . . . . .

treat a single number as interval

→ Build a BST over the elementary inter-vals and store for each ~~inter~~ elementary interval all associated intervals



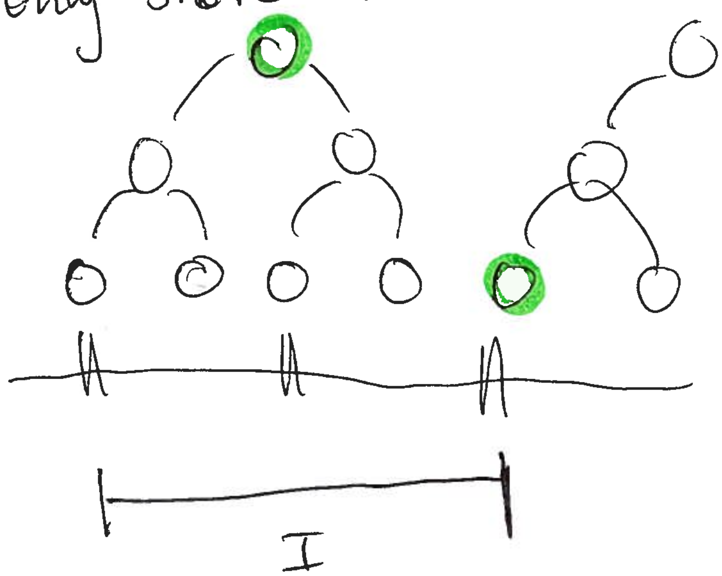
I is stored in the orange nodes

Query: Search for q and report all segments stored in the "leaf"

↳ Fast $O(\log n + k)$, but needs a lot of space !!!

How to fix this?

Only store an interval in a few nodes



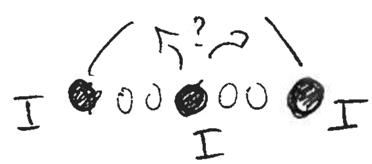
Store I only in the green vertices!

More precisely: If two children of a node store the same interval, store the interval in the node, not in the children

This induces a subdivision of the intervals into canonical subsets

Claim The segment tree needs $O(n \log n)$ space

proof: Every interval is stored in at most 2 vertices of the same depth



Either way 3 nods/level contradicts our convention

↳ $O(\log n)$ levels $\Rightarrow$ $O(n \log n)$ total space $\square$

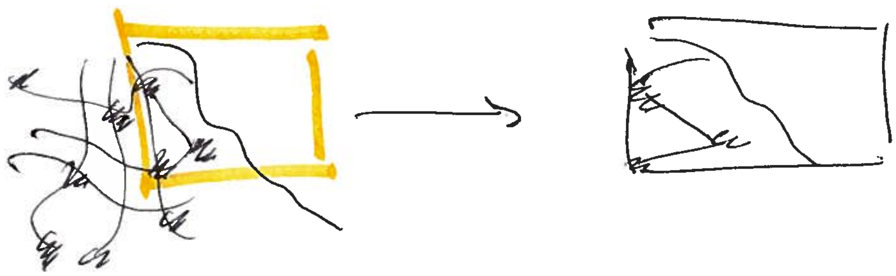
How to construct a segment tree?

1. Buit the tree over the elementary intervals
2. Add the intervals - TOP DOWN recursively

Because we have at most 2 nodes/level where the interval is stored we ~~have~~ need in total $O(n \log n)$ time

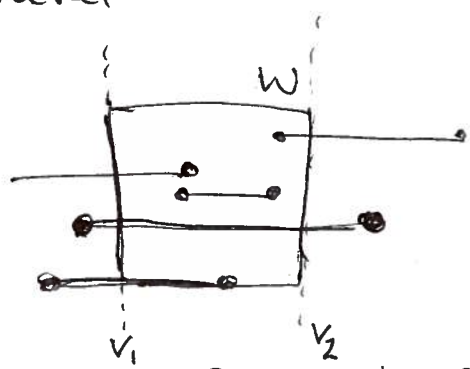
WINDOWING

Problem: Large Map, Line segments, but we want to access only a "window"



Variant: all line segments are horizontal or vertical (Chip layout)

↳ 1st horizontal



1 Endpoint in W } Part I
2 Endpoints in W }
No Endpoints in W } Part II

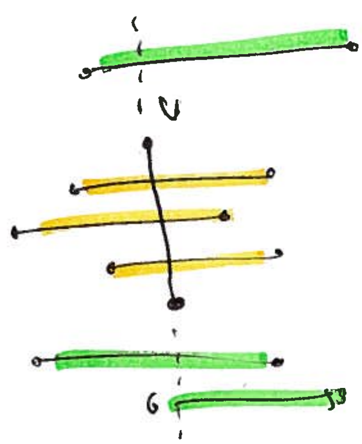
Part I: ~~Bo~~ • Use Range tree to find all endpoints in W

• For each endpoint $\in W$:
  if 2nd endpoint $\notin W$: report segment
  if 2nd endpoint $\in W$: report segment if 1st endpoint is left

Part 2: These segments are stabbed by v_1 & v_2

Idea find all segments stabbed by v_1 and check if the 2nd endpoint lies outside W

Problem



Not these

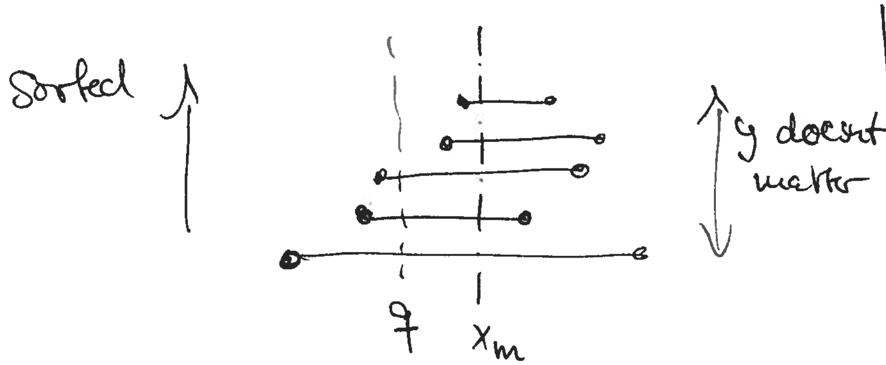
Report only these

Not these

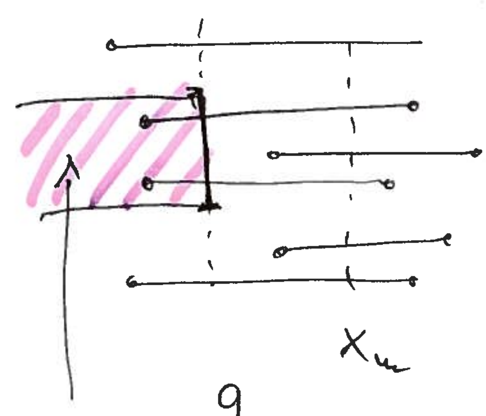
1st Solution

Modified interval tree

(We have to modify how to treat I_{mid}



Old idea



We need to report these endpoints!

Now

We can answer such queries with a 2D range tree

$\Rightarrow$ Query time ~~$O(|I_m| \log |I_m|)$~~.

$O(\log |I_m| + k')$

for each of the $\log n$ nodes touched in the interval-tree

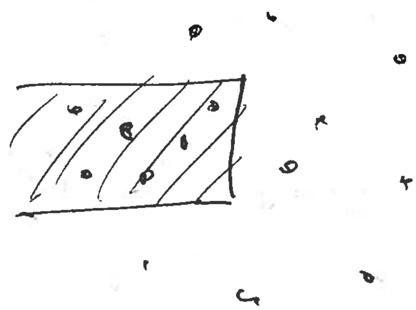
$\Rightarrow$ $O(\log^2 n + k)$ total

Storage goes up to $O(n \log n)$

(because the storage for a 2D range tree exceeds the sorted list by a factor of $\log n$)

<u>2nd Solution</u>: Again Interval-tree but with a new 2nd level DS called <u>priority search tree</u>! (PST)

A PST can answer queries in 2D with one side open:

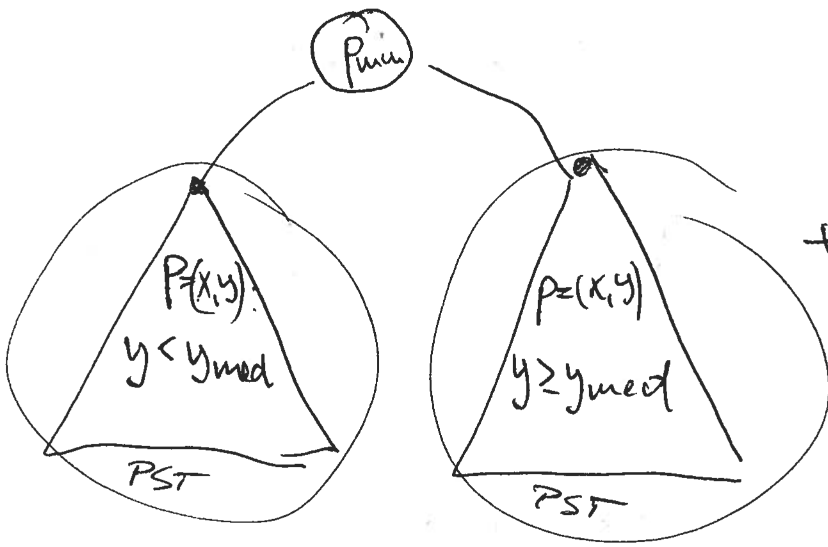


Exactly what we want!

Here is how it works

p_{min} = point with minimum x-coordinate x_{min}

y_{med} = median of the y-coordinates (ignore p_{min})



+ recurse

Query: Search for y_{min} & y_{max} in PST

+ check all points on the search path

Report the shaded subtrees but check if the p_{min}-root entry is too big

if so stop reporting

$\Rightarrow$ Querytime: $O(\log n + k)$ but Storage $O(n)$!!! ...