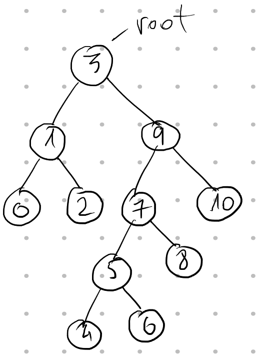
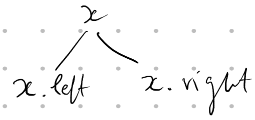
Augmented trees

Binary Search Trees (BST)

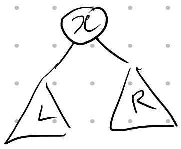
e.g.





search path

key
x.key



everything in L < x < everything in R

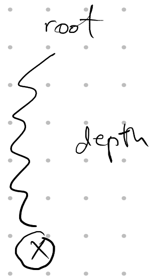
Purpose

Store a set $S \subseteq U$ → ordered

(Dictionary)

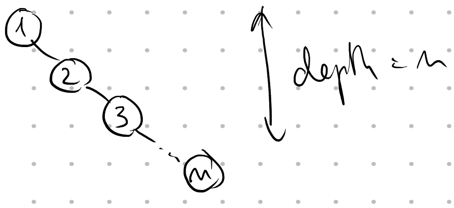
support operations
- search(x) : is $x \in S$? (if yes, recover data stored in x)
- insert(x)
- delete(x)

→ $O(\log n)$ $n = |S|$

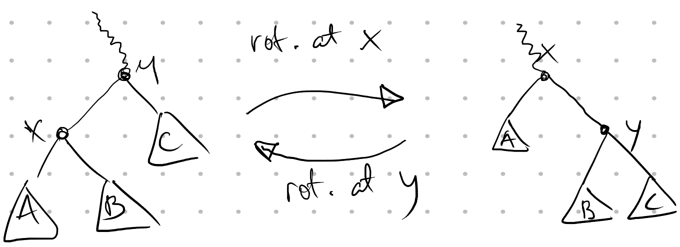


- balanced tree : depth $\leq O(\log n)$
- many strategies known: AVL, red-black trees, treaps, etc.

without balancing



balancing based on rotation operation



can use to restructure tree, make it balanced

Why use BSTs as a dictionary (instead of e.g. hash tables)

- ordering is essential, e.g. can use keys only in comparisons
- support more advanced operations using ordering of keys.
  - traverse all stored keys in order (in BST inorder traversal)
  - predecessor / successor
  - interval queries

$|S \cap [a,b]|$

Augmented trees

- store some additional structural data in nodes
- e.g. exercise sheet 2.

Example Applications

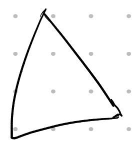
① Rank/Select dictionary

Given a set S (stored in BST)

operations:
- insert/delete an item
- search for an item
- select(k) : outputs k^{th} smallest key in S $\quad 1 \le k \le n$
- rank(x) : output rank of x (how many elements in S are $\le x$)

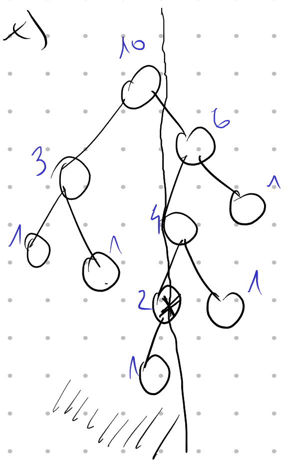
$O(\log n)$ time

recall
select(k) in heap



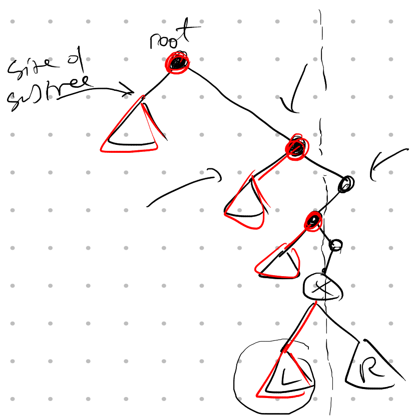
$O(k)$ time (using soft heaps)

Now:
select(k) in BST in $O(\log n)$ time.
rank(x)



rank(x)

where are elements $\le x$?



$|L|$

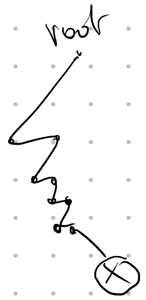
Idea: store in node x, size of subtree rooted at x, denoted $s(x)$ (including x)

Task: maintain $s(x)$ under all update operations.

insert/delete/search in time $O(\log n)$

use AVL-tree

insert(x)



need to increment all sizes on path

$O(\log n)$

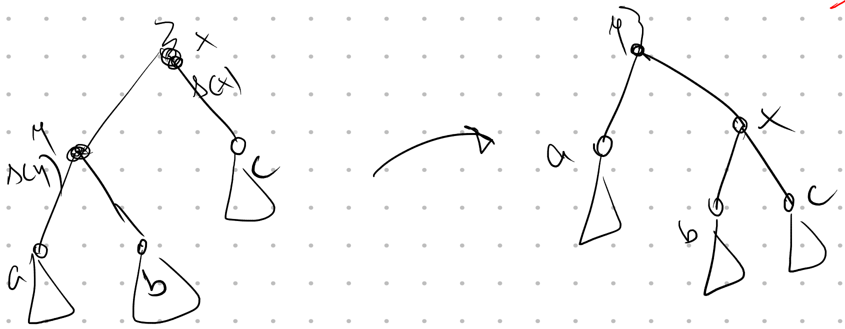
delete(x)

need to decrement all sizes on path

$O(\log n)$

AVL restructuring uses rotations

enough to show for a single rotation



update:

$s(a), s(b), s(c)$ unchanged

$s'(x) = s(x) - s(a) - 1$

$s'(y) = s(y) + s(c) + 1$

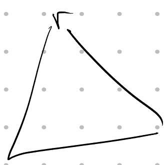
$O(1)$ time

charge it to rotation cost

We can assume AVL-tree, with augmentation $s(x)$

---

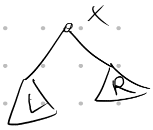
select(k, root)



select(k, r)

```
if k < s(r.left)+1
    return select(k, r.left)
elif k > s(r.left)+1
    return select(k - s(r.left) - 1, r.right)
else if k = s(r.left)+1
    return r.
```

rank (x, root)



rank (x, r)

```
if (x<r)
  return(rank (x, r.left))
elseif (x>r)
  return(rank (x, r.right) + s(r.left)+1)
else (x=r)
  return(s(r.left)+1)
```

---

② Interval queries

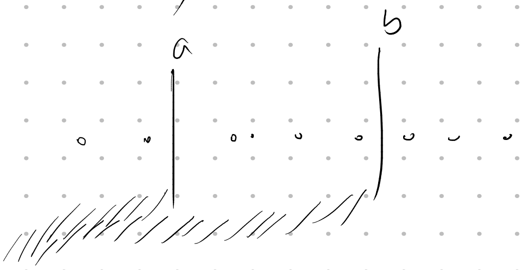
$S = \{a_1, \dots, a_n\}$

insert, delete, search

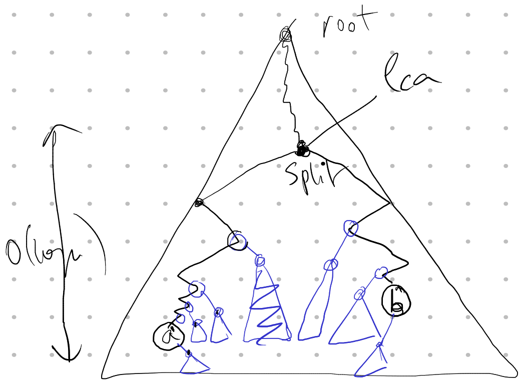
count (a, b) → return $|S \cap [a,b]|$ (Interval queries)

list (a, b) → return $S \cap [a,b]$

rank (b) − rank (a) → (a, b] $O(\log n)$



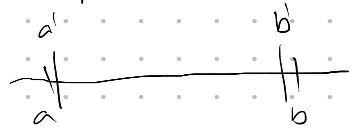
$O(\log n + k)$ → # items listed



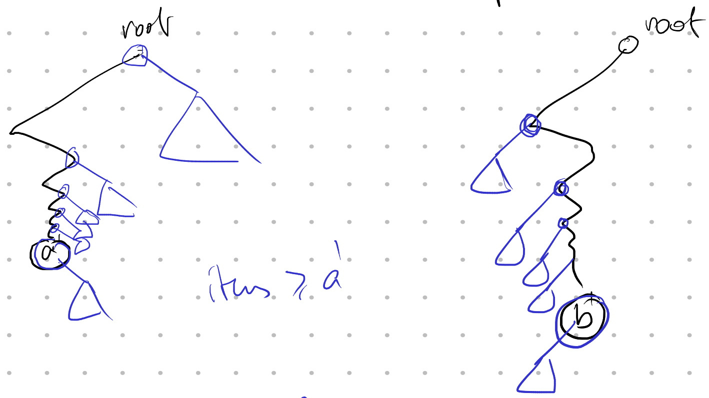
$[a, b]$

a' smallest item in the tree s.t. $a' \geq a$

b' largest item in the tree s.t. $b' \leq b$



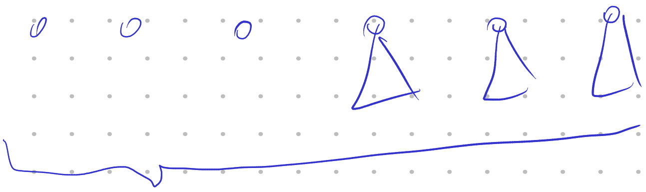
report all items in $[a', b'] \cap S$



items $\geq a'$ items $\leq b'$

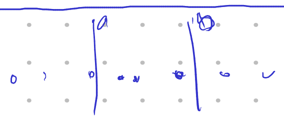
$\cap$

$O(\log n)$ elements and subtrees

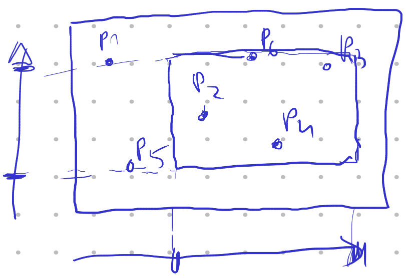


$O(\log n)$

output all elements in subtrees $O(\log n + k)$



extend to 2D



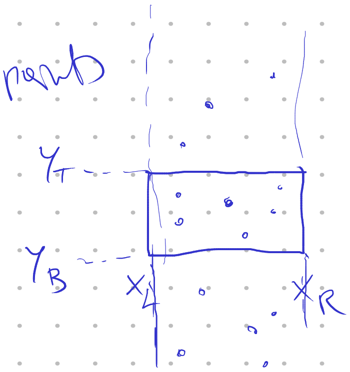
Query: axis-parallel rectangles

- list points that fall within rectangle
- how many points?

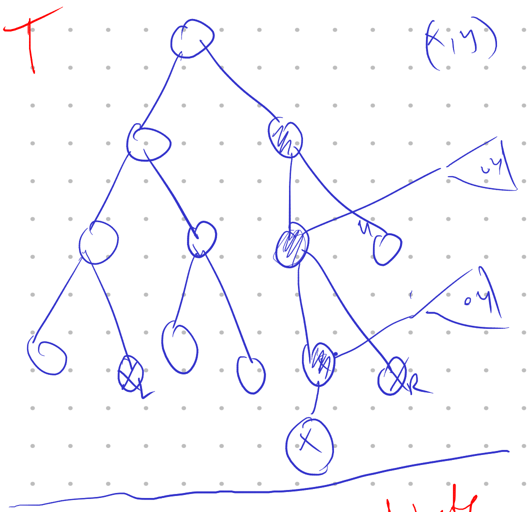
extend solution to 2D

Static case: no insert/delete

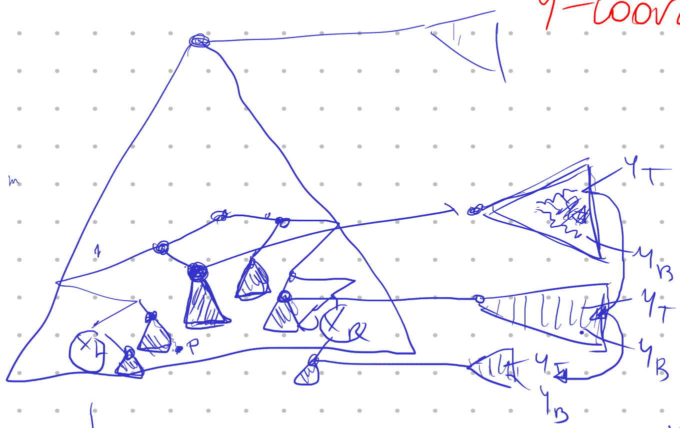
Build BST on x-coordinates of points

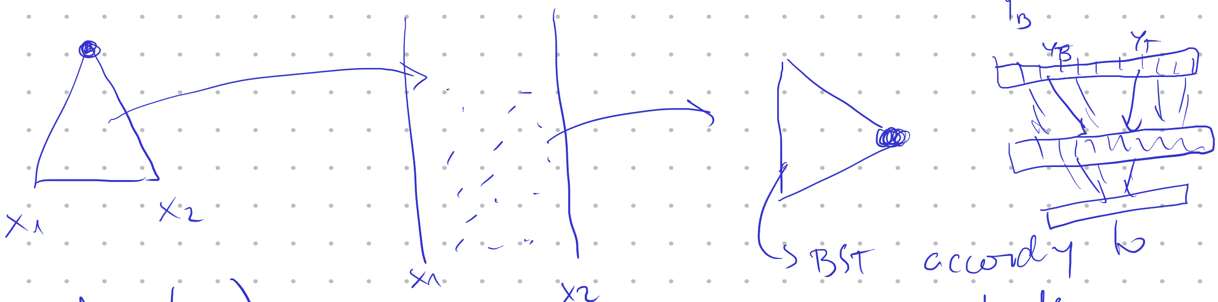


subtrees of T also stored in secondary trees, indexed by y-coord.



index by x-coordinate





BST according to y-coordinates

each point stored $O(\log n)$

total storage $O(n \log n)$

$O(\log^2 n + k)$ → only if we actually report the items.

Range tree DS

- how to make it dynamic?
  (doable but complicated)

- how to get rid of extra log factor (in 2D)

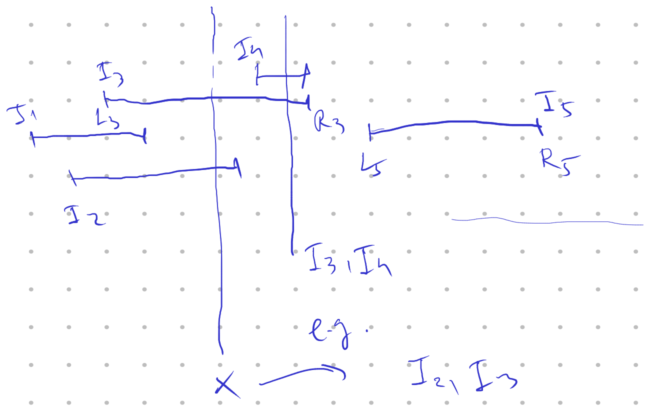
$O(\log n + k)$

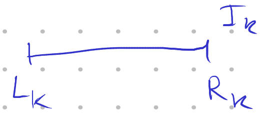
(fractional cascading)

→ trick to speed up simultaneous binary search in multiple sets

③ store intervals
point queries

S : set of intervals $(I_1, \dots, I_n)$



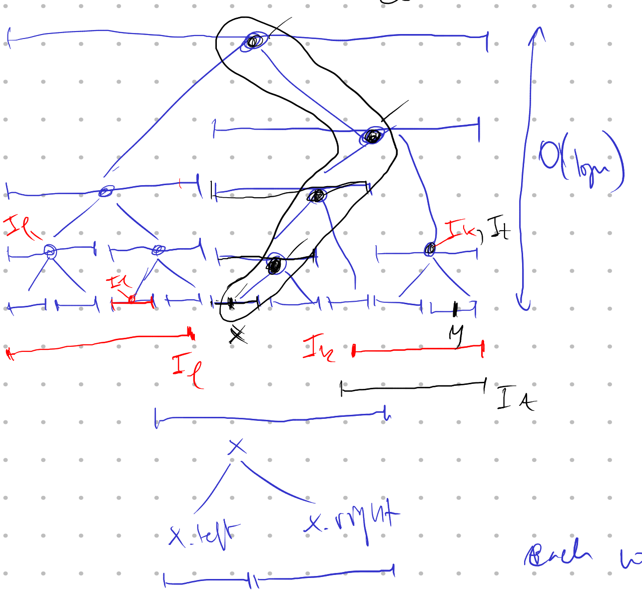


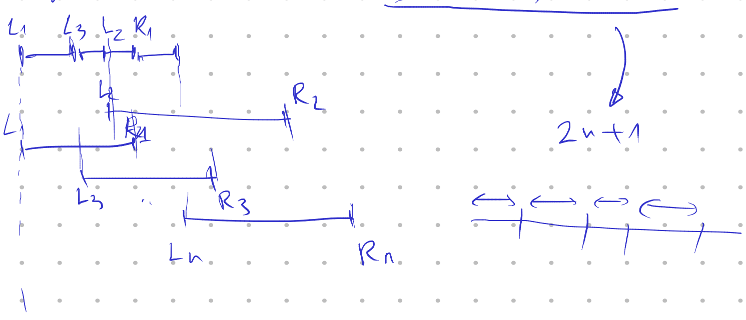
insert (L, R, id)
delete
query (x)

---

query(x) $\longrightarrow$ search for leaf whose elementary interval contains x

- Binary search tree
- leaves store all "elementary intervals" $\rightarrow 2n+1$





$\{L_1, \dots, L_n, R_1, \dots, R_n\}$ sort

Each node stores union of two child-intervals

Input intervals $I_1, \dots, I_n$

I_k stored at a node x if interval of x contained in I_k
but interval of parent(x) not contained in I_k

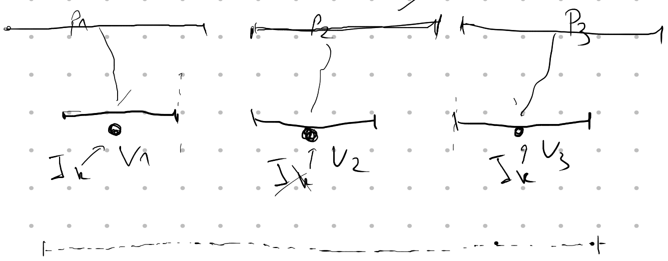
In how many nodes is I_k stored (at most?)

Claim: at most $O(\log n)$ places (nodes) $\longrightarrow O(n \log n)$ storage

Claim 2: at most two nodes of same level in tree store an interval I_k. $\forall k \in \{1, \dots, n\}$

Proof by contradiction

also contained in I_k



$\Longrightarrow$ ↯

Claim 2 $\Rightarrow$ Claim 1

---

query(x) → - find leaf that contains x

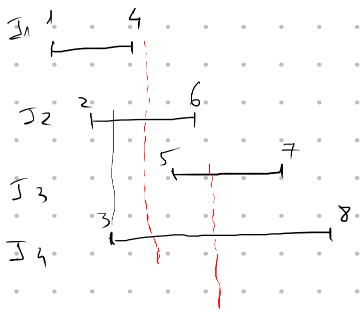
- output all intervals stored in nodes on search path of x

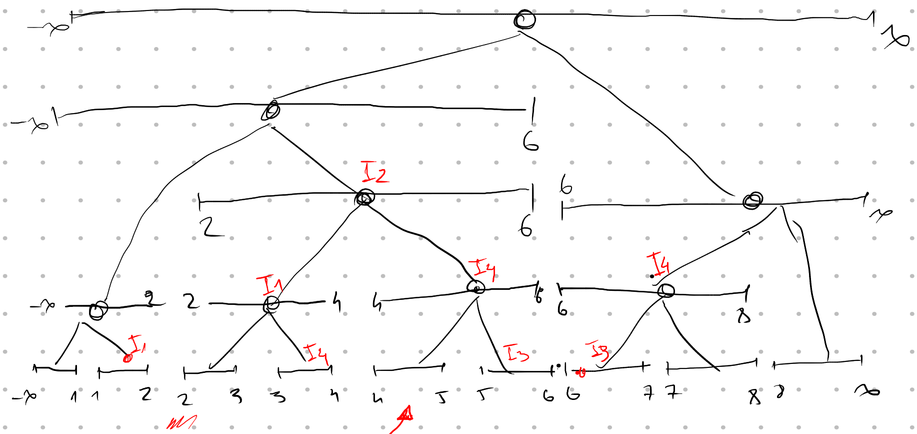
$O(\log n + k)$

{ insert
{ delete

rotation (AVL)

example:





query(4.5) ⟶ return I_2, I_4

query(6.1) ⟶ I_3, I_4

Segment tree