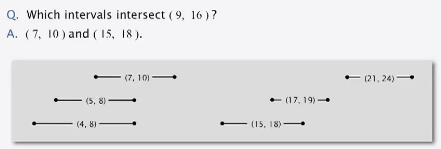
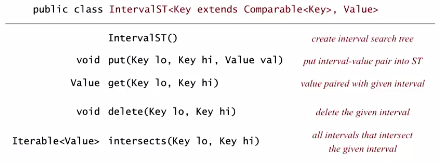
Interval search trees

1d interval search: data structures to hold set of overlapping intervals

* Insert an interval (lo, hi)
* Search for an interval (lo, hi)
* Delete an interval (lo, hi)
* Interval intersection query: given an interval (lo, hi), find all the intervals (or one) in the data structure that intersect(s) (lo, hi)



Interval search API:

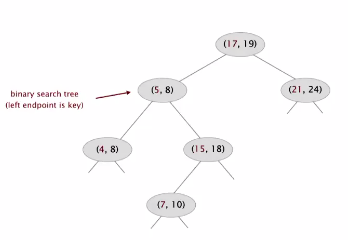


*\*Nondegeneracy assumption*

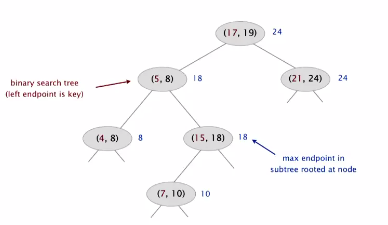
Interval search tree

Create BST where each node stores an interval (lo, hi):

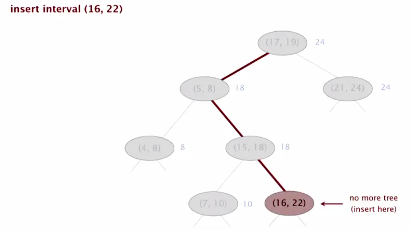
* Use left endpoint of interval as BST key

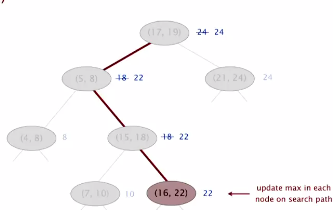


* Store max endpoint in subtree rooted in node



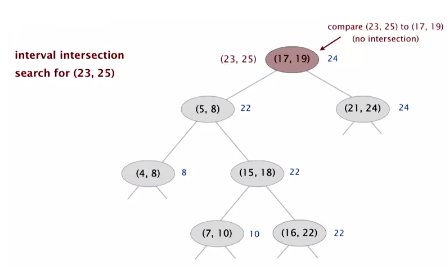
Insert an interval (lo, hi):

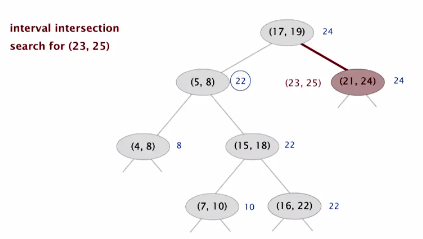
* Insert into BST, using lo as key
* Update max in each node on search path



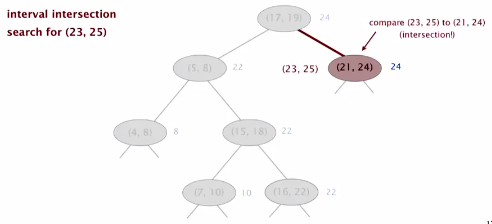
Search for intersecting intervals:

* If interval in node intersects query interval, return it

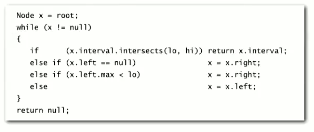


* Else if left subtree is null go right
* Else if max endpoint In left subtree is less than lo go right
* Else go left

Result:

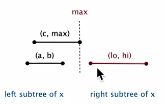


Amazingly simple implementation:



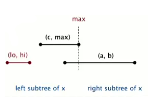
Proof:

* If left subtree is empty, no intersection
* If max endpoint is less than low, every interval in left subtree has a max less than low



Max endpoint *max* in left subtree is less than *lo* ->   
for any interval (a, b) in left subtree of *x*, we have *b* <= *max* < lo

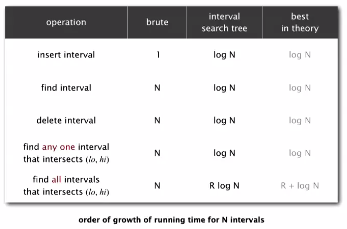
* If you go left there is either an intersection or no intersection at all (including right)



* Since we go left, we have lo <= max
* Then for any interval (a, b) in right subtree of x, hi < c <= a -> no intersection in right

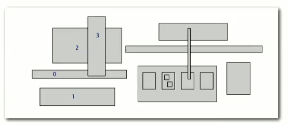
Use a red-black BST to guarantee performance

Run time?



Everything takes logN time, except finding all intervals, which is R log N (R is num intersecting intervals)

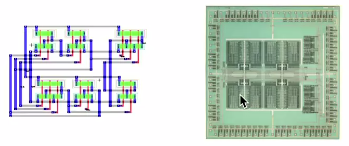
Rectangle intersection

Problem: find intersections among a set of N orthogonal rectangles

Quadratic algorithm: check all pairs of rectangles for intersection

\*Nondegeneracy assumption

This problem is very important for a number of applications:

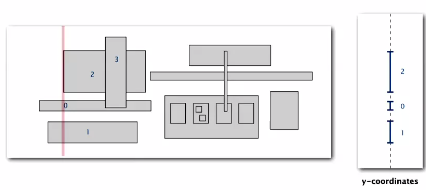
Design rule checking for microprocessor chip creation

* ensure certain elements did not intersect
* certain spacing is needed between different types of wires
* Debugging was orthogonal rectangle intersection search

Quadratic algorithm would be problematic because as per Moore’s law, processing power doubles every 18 months. If we use a quadratic algorithm with such increases, the run time will dramatically increase (double based on Moore’s law) for each new iteration of processors. Therefore, linearithmic algorithm is necessary to keep pace with Moore’s law.

Sweep line algorithm (use for interacting rectangles instead of lines):

* X coordinates of left and right endpoints define events   
  (left side of rectangle == interval, interval is added to interval search tree)



* When we reach the right, we remove

Proposition is that **sweep line algorithm takes N log N + R log N**  time to find **R intersections** among a set of **N rectangles** Proof:

* Put x coordinates on PQ (or sort) -> N log N
* Insert y-intervals into ST -> N log N
* Delete y-intervals from ST -> N log N
* Interval searches for y-intervals -> N log N + R log N

Sweep line reduces 2d orthogonal rectangle intersection search to 1d interval search

**Summary of geometric applications of BSTs**

