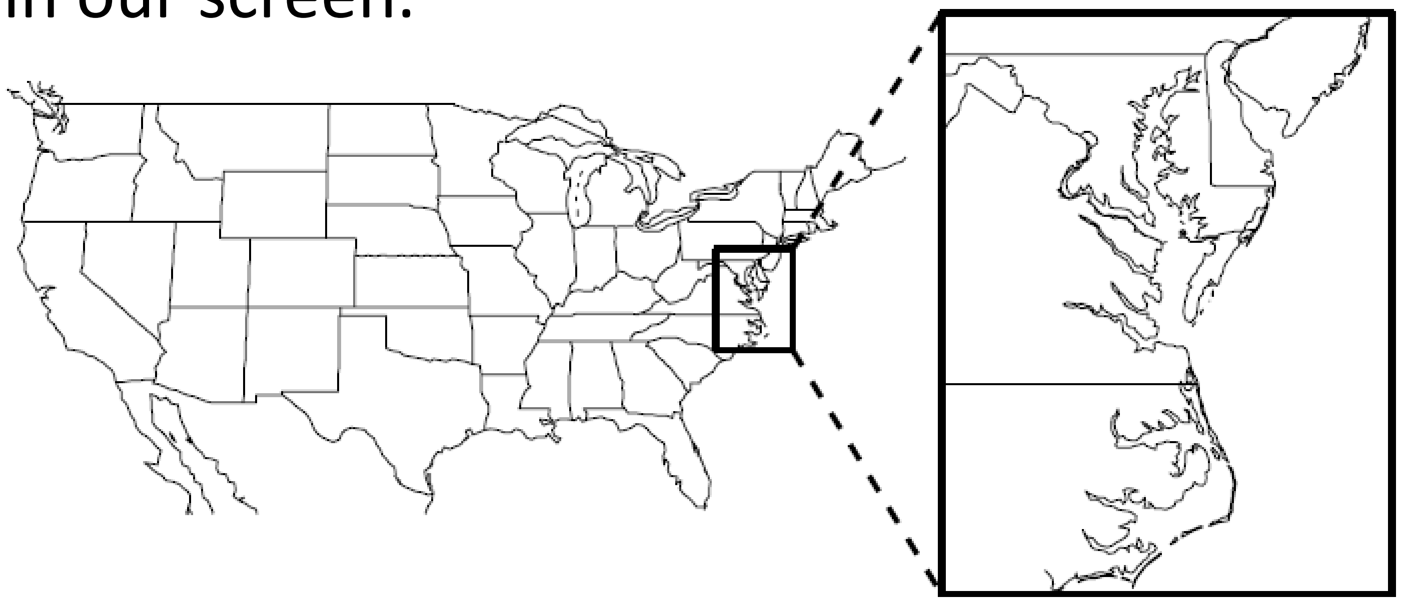
More Geometric Data Structures

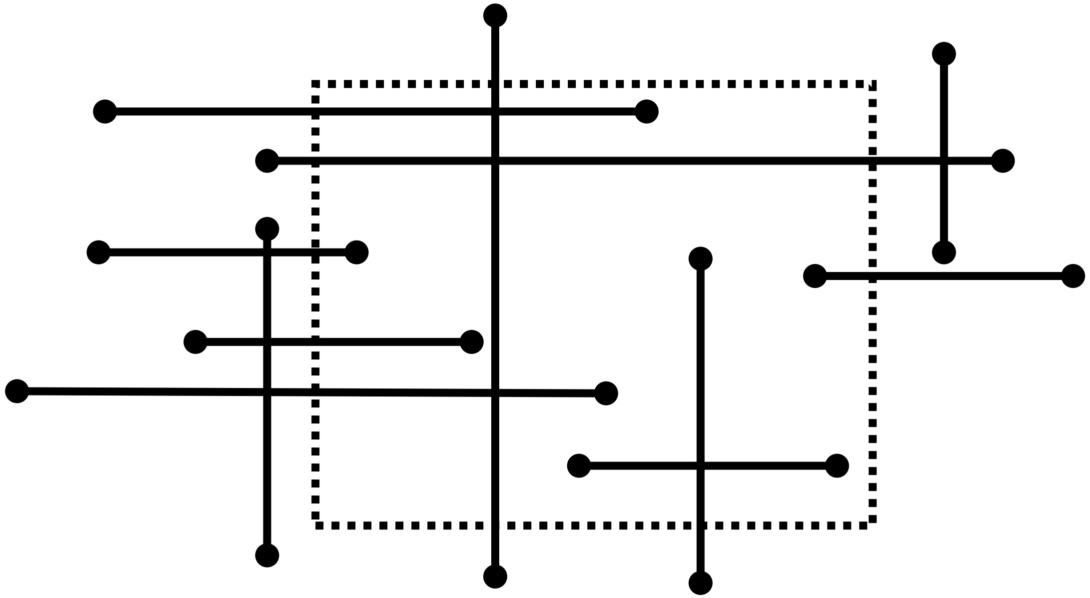
Windowing

- Consider a mapping application (Waze for example)
- The entire map contains huge amount of objects.
- However, at any given time, we need to display a small amount, just the object in our screen.



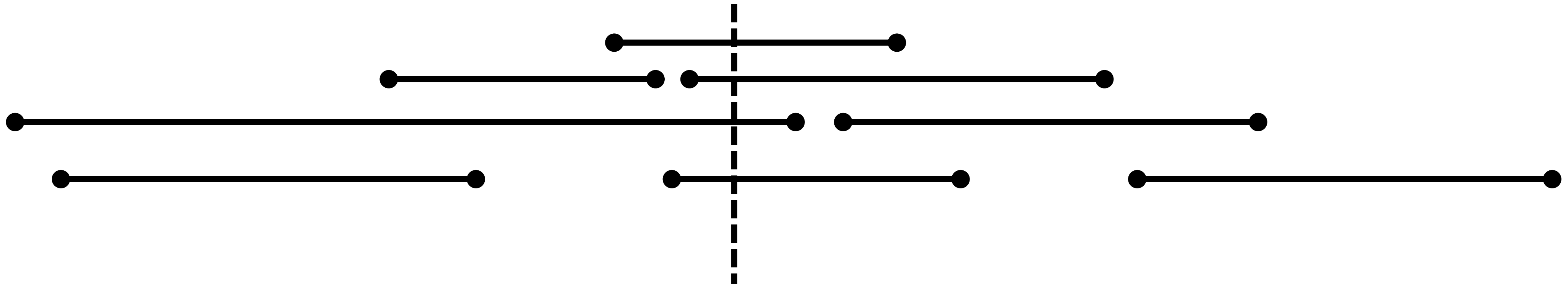
Windowing

- We have seen how to find points in a region, but what about other objects?
- We will begin with a simpler case, only axis-aligned segments.
- We can handle segment with endpoints inside the window easily.
- How can we handle segments that cut the window with no end point inside it?



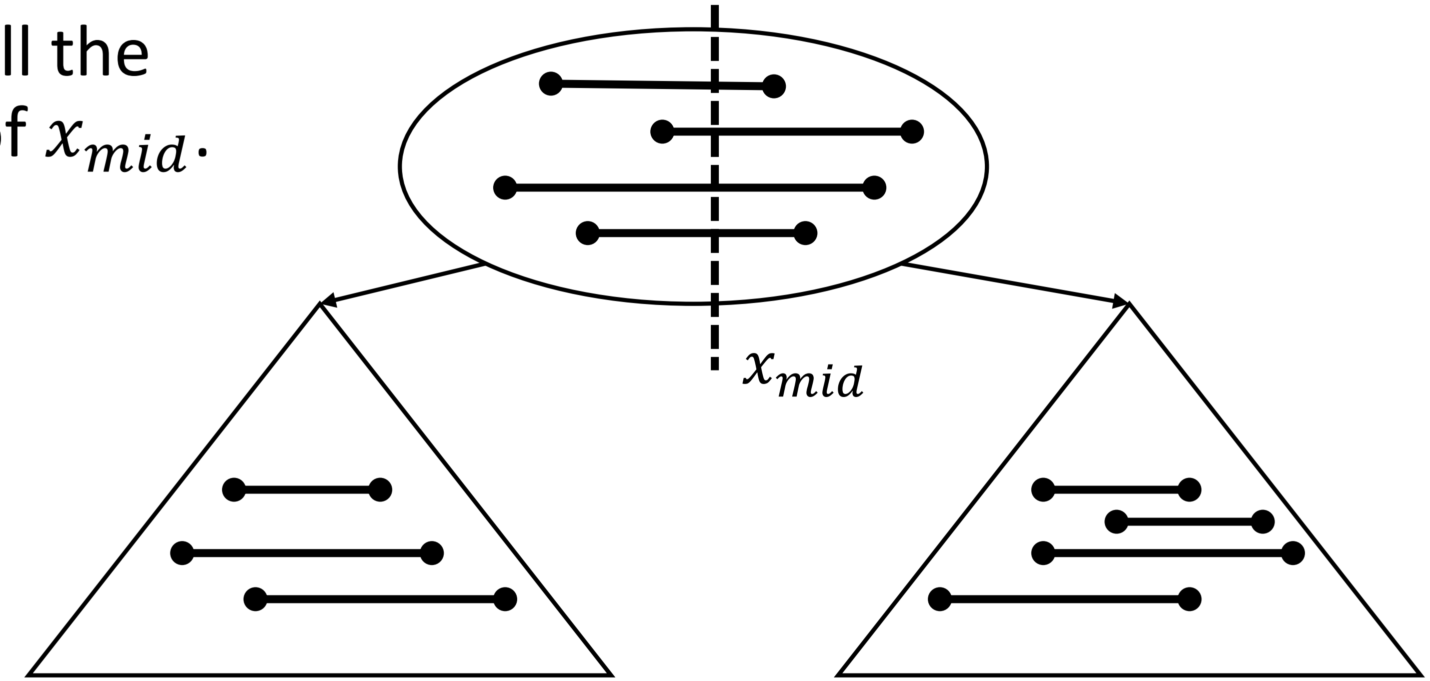
Interval Trees

- Lets simplify the problem:
- Given a set of horizontal intervals, find the set of intervals that contain the point x.
- Trivial solution: $O(n)$, surely we can do better.
- Can we use a tree? When does one interval is smaller than another?



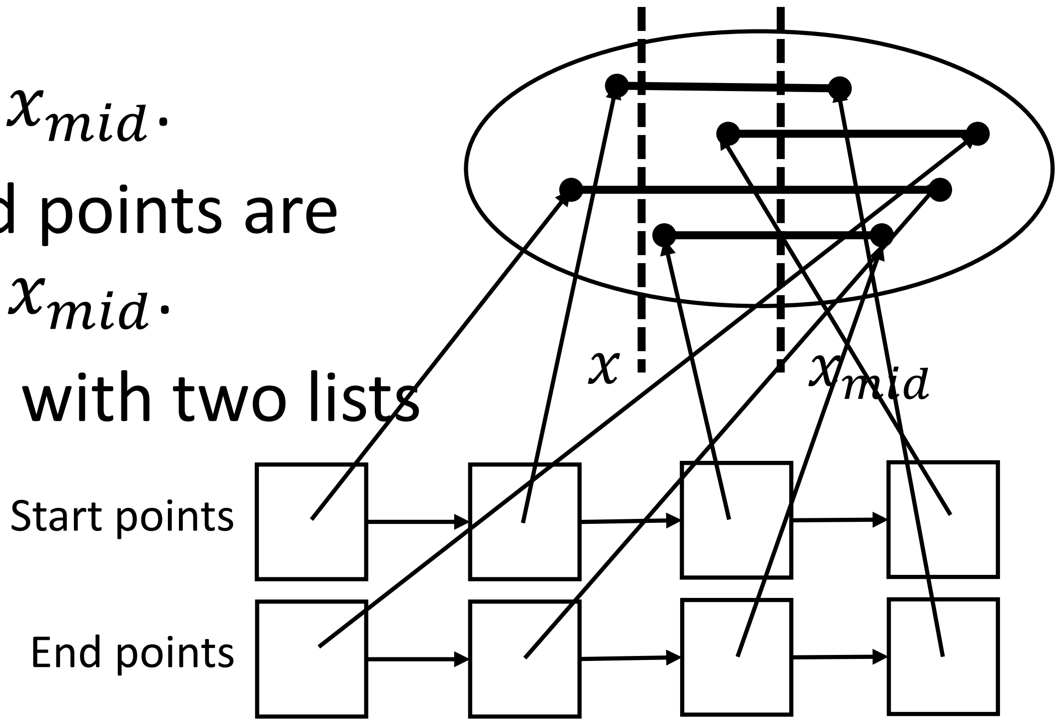
Interval Trees

- Idea: the root will contain the intervals which are roughly in the middle.
- Formally, let x_{mid} be the median of all interval end points.
- In the root we will have all the intervals intersecting x_{mid}
- To the left, a sub tree with all the intervals strictly to the left of x_{mid}.
- The same to the right.



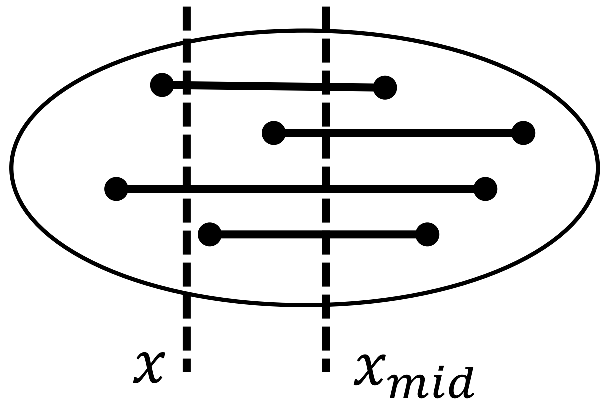
Interval Trees

- Problem: how do we find which intervals in a node intersects x?
- Maybe all the intervals intersects x_{mid}, thus all are in the same node.
- Do we have the same problem again?
- No, we know all the intervals intersects x_{mid}.
- In the example we know that all the end points are to the right of x, since x is to the left of x_{mid}.
- Knowing this, we can solve the problem with two lists in the node, one for each direction.



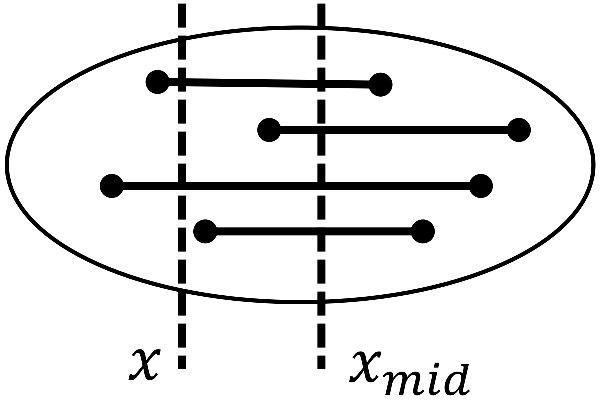
Interval Trees

- What is the complexity of constructing an interval tree?
- We need to sort the intervals - $O(n \log n)$.
  - Once for all the tree.
- Finding the median takes - $O(n)$.
- Constructing the root node - $O(n)$.
- Constructing the left and right subtrees - $2T\left(\frac{n}{2}\right)$.
  - Since we split by the median there are at most $\frac{n}{2}$ intervals in each tree.
- $T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$.



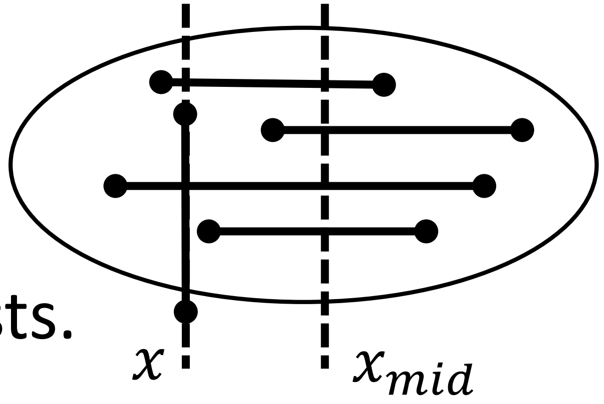
Interval Trees

- Query – find the relevant nodes (as in a BST), and in each node report the intersecting intervals.
- Query time – $O(\log n + k)$.
    - Where k is the number of reported intervals.
- Space complexity – $O(n)$.



Interval Trees

- Until now we asked for the intervals intersecting a line.
- But what if instead of a line we have a segment?
- We look for start **points** in the area $[-\infty, x] \times [y, y']$.
- We know how to handle points:
- In each node we will have $2d$-Range trees instead of lists.
- The query time in the Range trees is $O(\log n + k)$, so $O(\log^2 n + k)$ in total.
- Space complexity $O(n \log n)$.

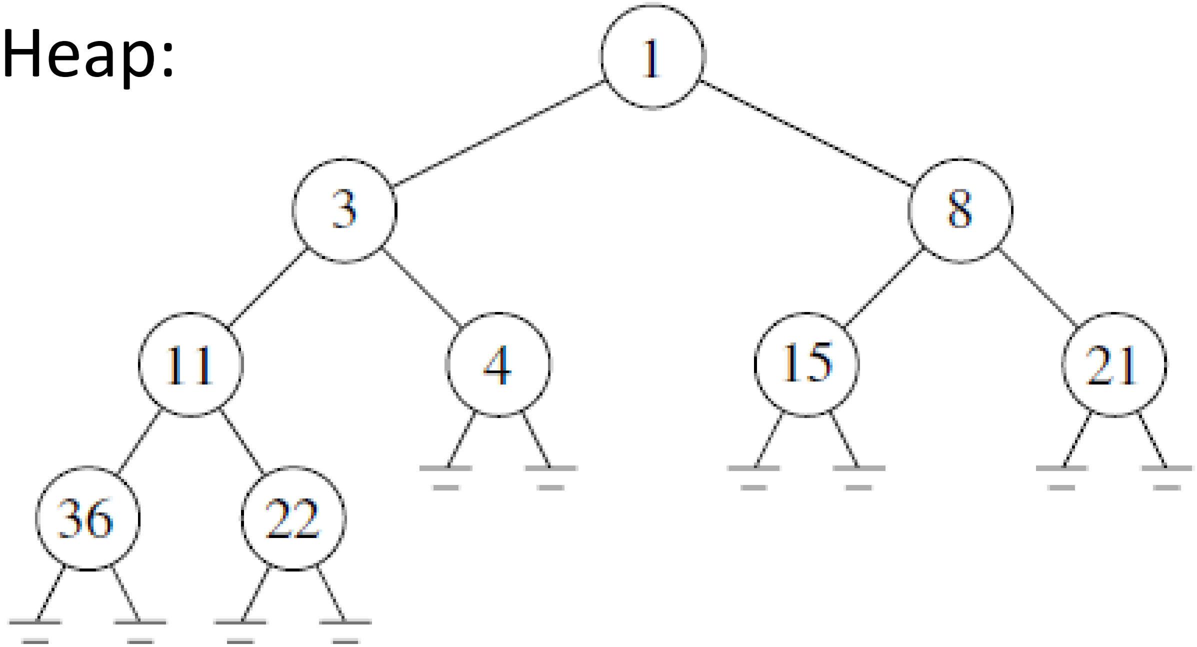


Priority Search Trees

- Recall our last problem:
- Given a set of points find those inside $[-\infty, x] \times [y, y']$.
- The area is not bounded, can we do better than $2d$-Range tree?
- We have seen that without the y range we can simply use lists and report the points starting from the minimum one until reaching x.
- This means that we don't need to be able to search on the x-axis.
- What data structure will allow us to have the y data searchable and the x data traverseable from the minimum value until x?

Priority Search Trees

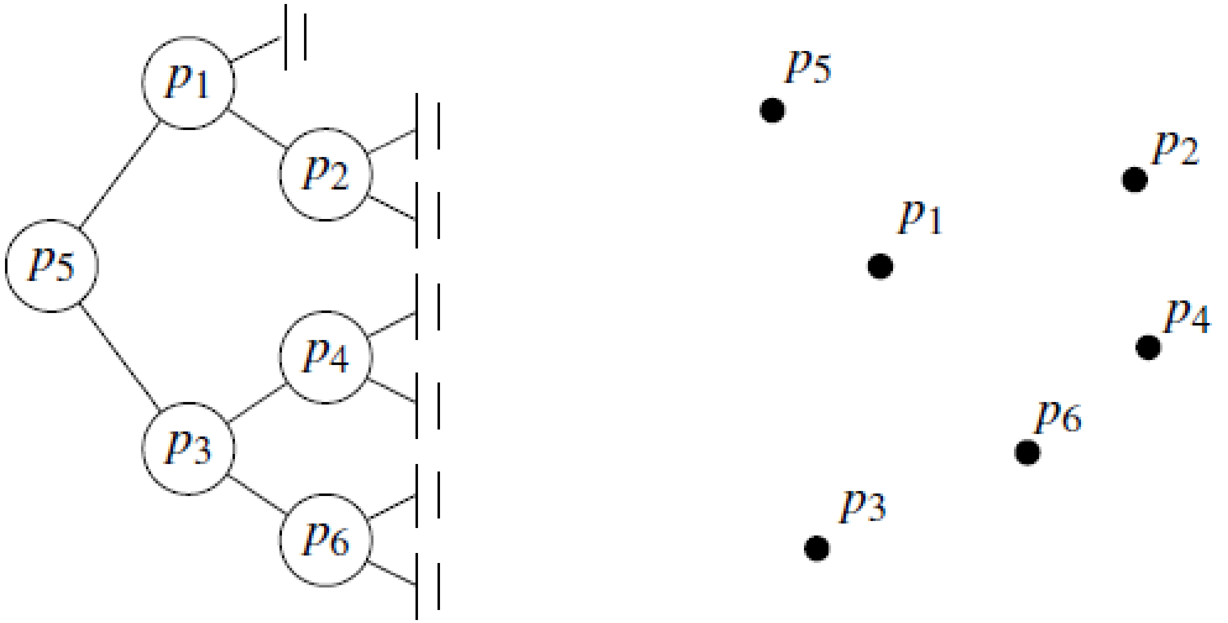
- Reminder - Min-Heap:



- Can we find all the elements smaller than some value x in $O(k)$ time?
- Yes, start in the root, and traverse each sub tree with root smaller than x.

Priority Search Trees

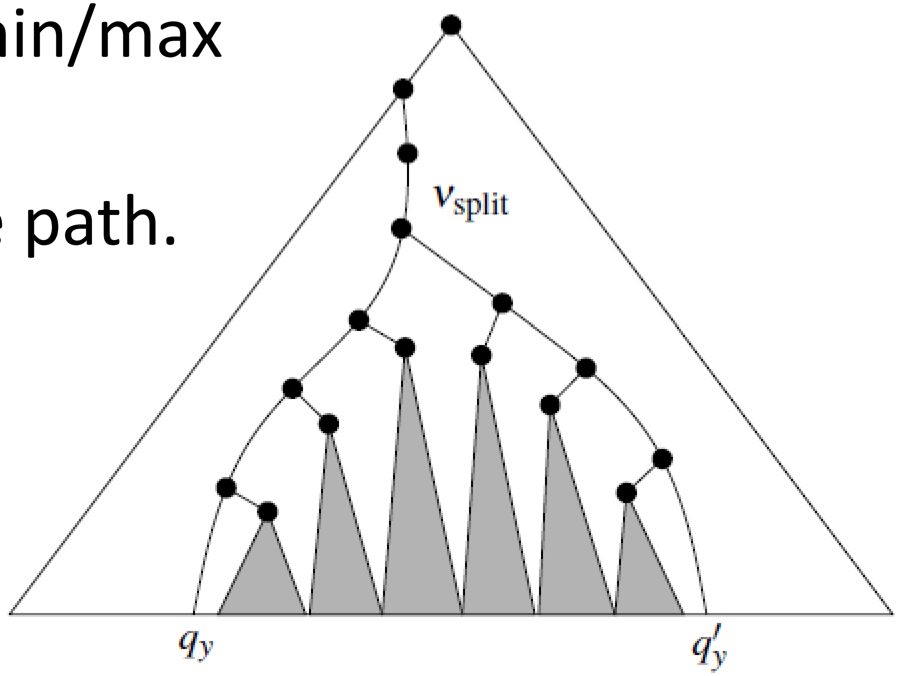
- Our full data structure will be a hybrid between a search tree and a heap:



- Heap according to the x axis, and all the elements in the left sub tree are smaller than the elements in the right sub tree (but not necessarily smaller than the root).

Priority Search Trees

- Using this data structure we can look for subtrees fully contained in $[y, y']$, and inside them look for all the elements inside $[-\infty, x]$ according to the heap.
- In order to search for y and y' store the min/max in each sub tree in each node.
- We also need to check all the nodes in the path.
- Query complexity – $O(\log n + k)$.
  - Without fractional cascading.
- Space complexity – $O(n)$.
  - Reducing the interval tree space complexity to $O(n)$.



Non Axis-Aligned segments?

- What about general segments, that is, not axis-aligned?
- We'll see next week.