General Data Structures

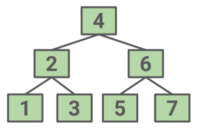
Data structures allow us to avoid looking at all of the data all of the time.

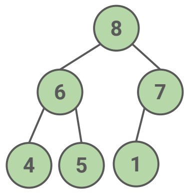
Binary search tree. Make a decision to ignore data based on key comparison.

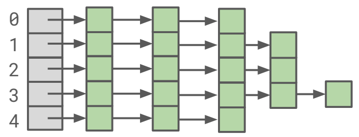
Binary heap. Optimize for access to the smallest or largest items.

Hash table. Make a decision to ignore data based on hash code, bucket index.

Invariants help us to ensure consistency.





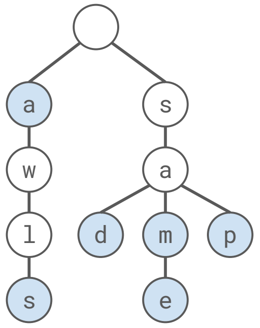


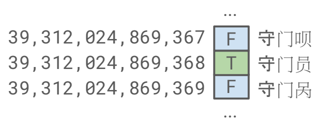
Specialized Data Structures

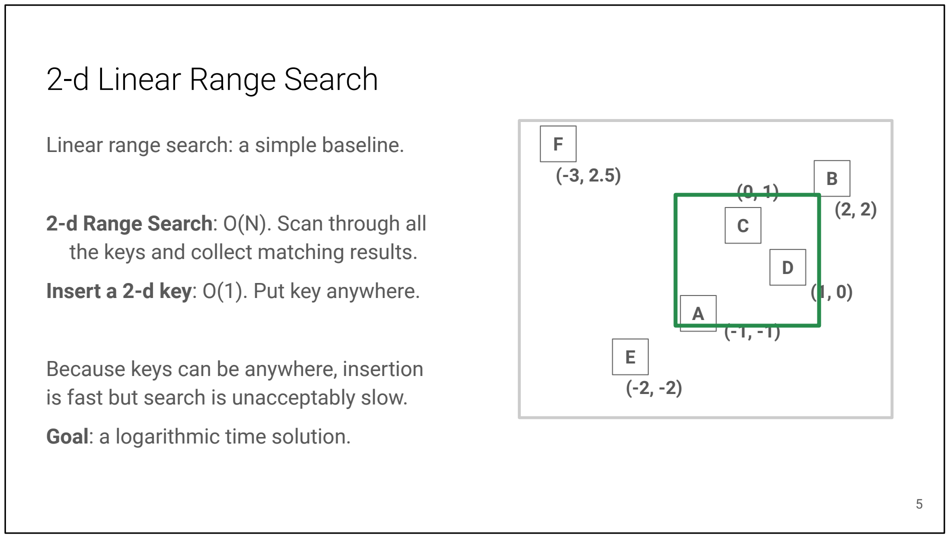
Data-Indexed Array. Keys must be small.

Tries. Keys must be subdivisible (strings).

Today: **multi-dimensional keys**.




...
39,312,024,869,367 F 守门呗
39,312,024,869,368 T 守门员
39,312,024,869,369 F 守门呙
...



2-d Linear Range Search

Linear range search: a simple baseline.

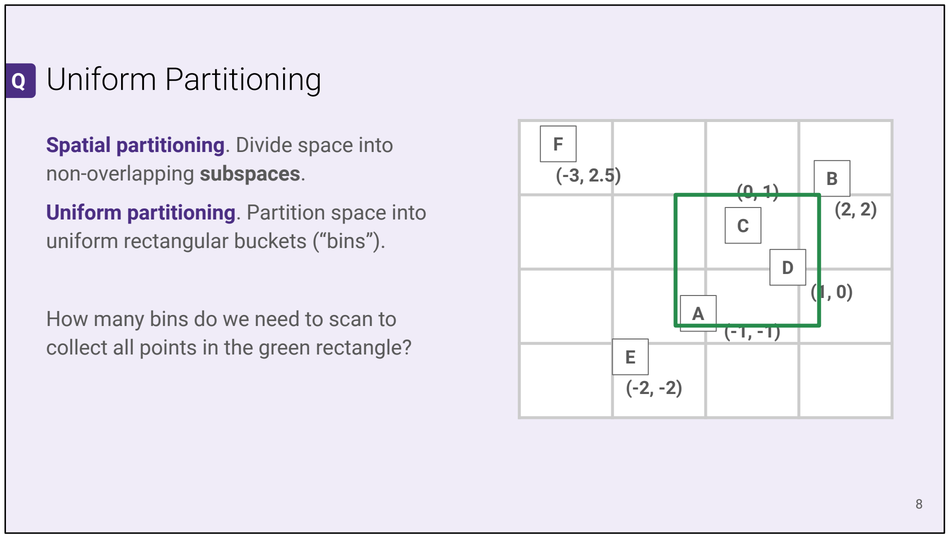
2-d Range Search: O(N). Scan through all the keys and collect matching results.

Insert a 2-d key: O(1). Put key anywhere.

Because keys can be anywhere, insertion is fast but search is unacceptably slow.

Goal: a logarithmic time solution.

F
(-3, 2.5)
B
(2, 2)
(0, 1)
C
D
(1, 0)
A
(-1, -1)
E
(-2, -2)



Uniform Partitioning

Spatial partitioning. Divide space into non-overlapping **subspaces**.

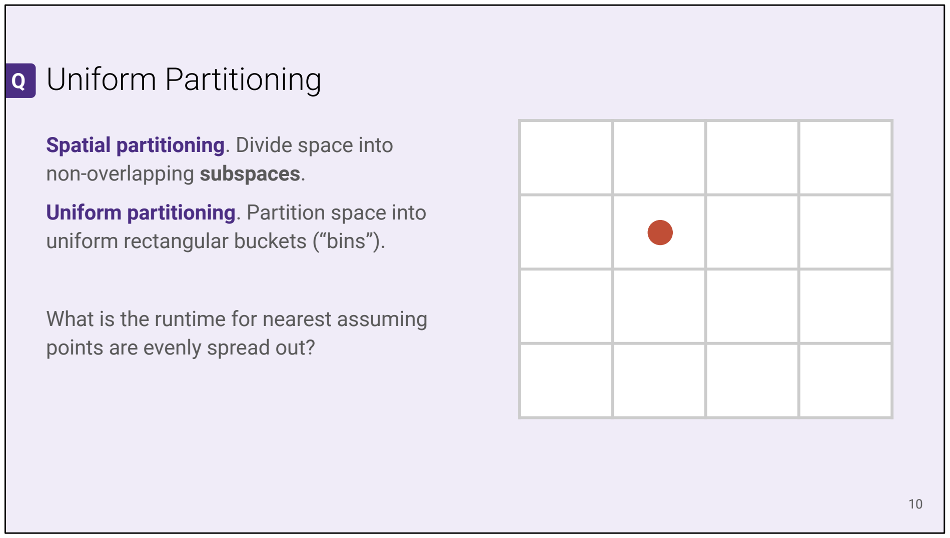
Uniform partitioning. Partition space into uniform rectangular buckets (“bins”).

How many bins do we need to scan to collect all points in the green rectangle?

Q
F
(-3, 2.5)
B
(2, 2)
(0, 1)
C
D
(1, 0)
A
(-1, -1)
E
(-2, -2)

Data structures optimize for certain operations on data by coming with organizational schemes that allow us to ignore large portions of the data. These organizational schemes are implemented with algorithms that respect the data structure invariants.

Q1: How many bins do we need to scan to collect all the points in the red rectangle?

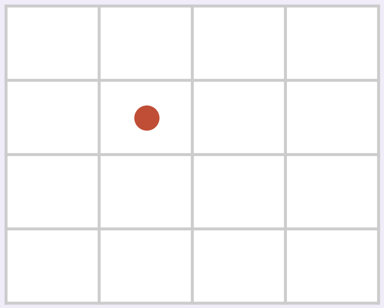


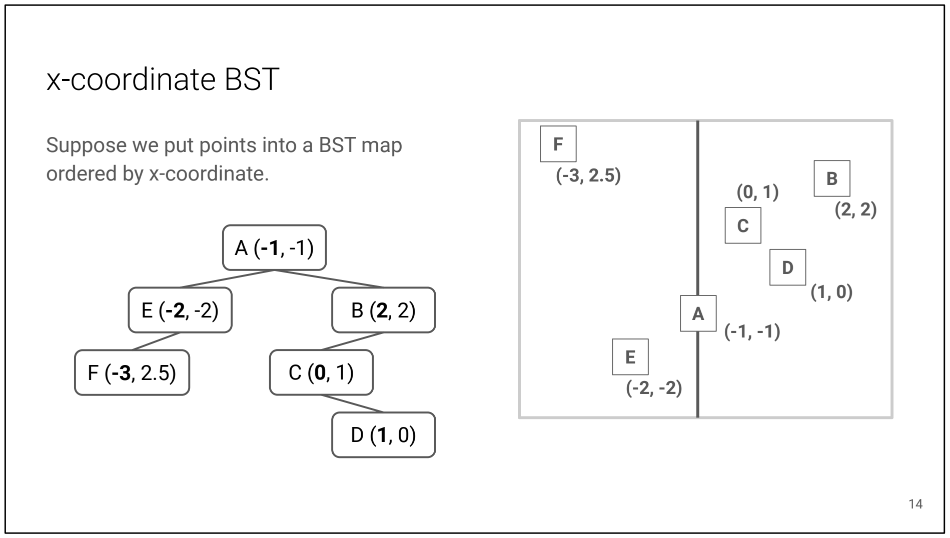
Q Uniform Partitioning

Spatial partitioning. Divide space into non-overlapping **subspaces**.

Uniform partitioning. Partition space into uniform rectangular buckets ("bins").

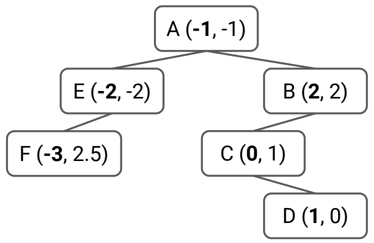
What is the runtime for nearest assuming points are evenly spread out?

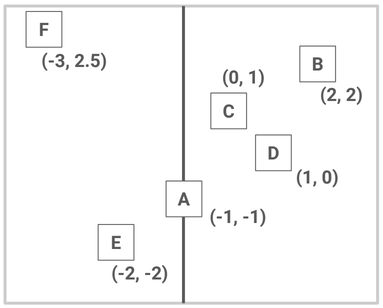




x-coordinate BST

Suppose we put points into a BST map ordered by x-coordinate.





Q1: What is the runtime for nearest assuming points are evenly spread out?

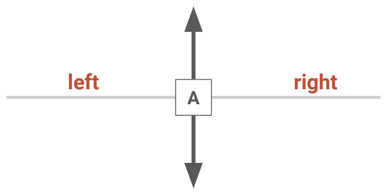
More general theme inspired by binary search trees vs. ordered linked nodes: recursive subdivision leads to logarithmic behaviors, while uniform subdivision leads to linear behaviors.

Recursive Partitioning

1-dimensional data (BST)

Keys are ordered on a line.

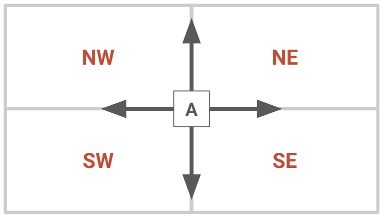
Recursive decision: **left** or **right**.



2-dimensional data (Quadtree)

Keys are located on a plane.

Recursive decision: **NE**, **SE**, **SW**, or **NW**.

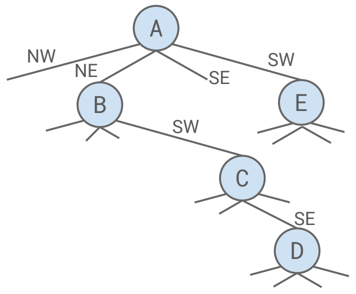


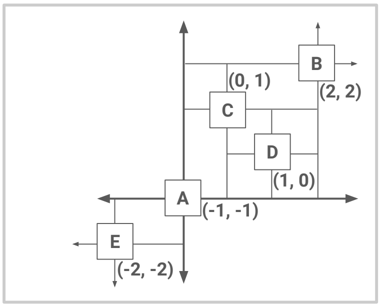
?: What does a quadtree look like? Each node has how many children?

Quadtree

Demo

5 objects in 2D space.





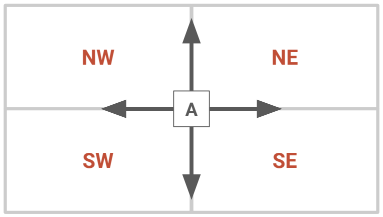
?: Does insertion order affect the balance of a quadtree?

Recursive Partitioning

2-dimensional data (Quadtree)

Keys are located on a plane.

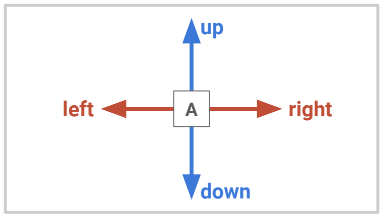
Recursive decision: **NE**, **SE**, **SW**, or **NW**.



2-dimensional data (2-d tree)

Recursive decision 1: **left** or **right**.

Recursive decision 2: **up** or **down**.



2-d Tree

Demo

Idea. Root node partitions entire space
left and **right** (by x-coordinate).

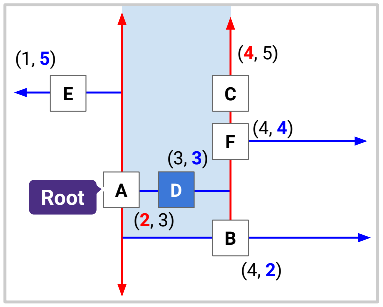
All depth 1 nodes partition subspace into
up and **down** (by y-coordinate).

All depth 2 nodes partition subspace into
left and **right** (by x-coordinate).

...

Each point owns **2 subspaces**.

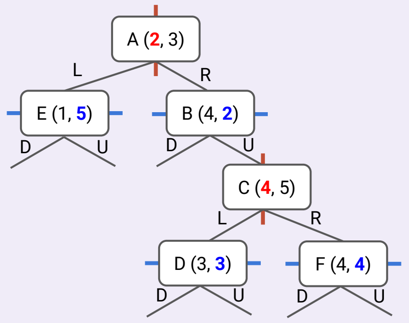
The subspace above **D** is infinitely large.

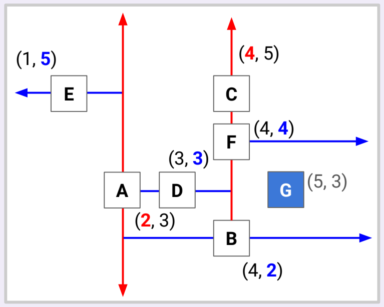


?: Does insertion order affect the balance of a k-d tree?

Q 2-d Tree Insertion

Where would **G** go in the 2-d tree?





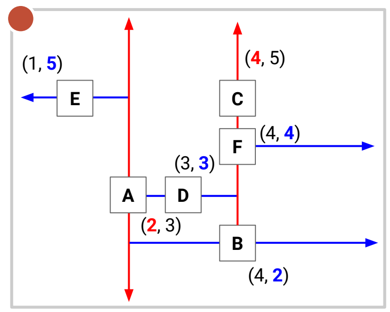
Q1: Where would **G** go in the 2-d tree?

2-d Tree Nearest Neighbors

Demo

Optimization. Do not explore subspaces that can't possibly have a better answer than the current best.

Find the nearest point to (0, 7).



There's a more advanced and subtle pruning rule that we'll see in the homework.