CS 225

Data Structures

October 12 – Disjoint Sets and kD-tree

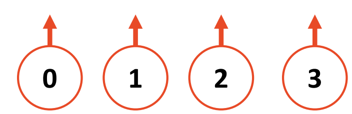
G Carl Evans

Implementation #2

- We will continue to use an array where the index is the key

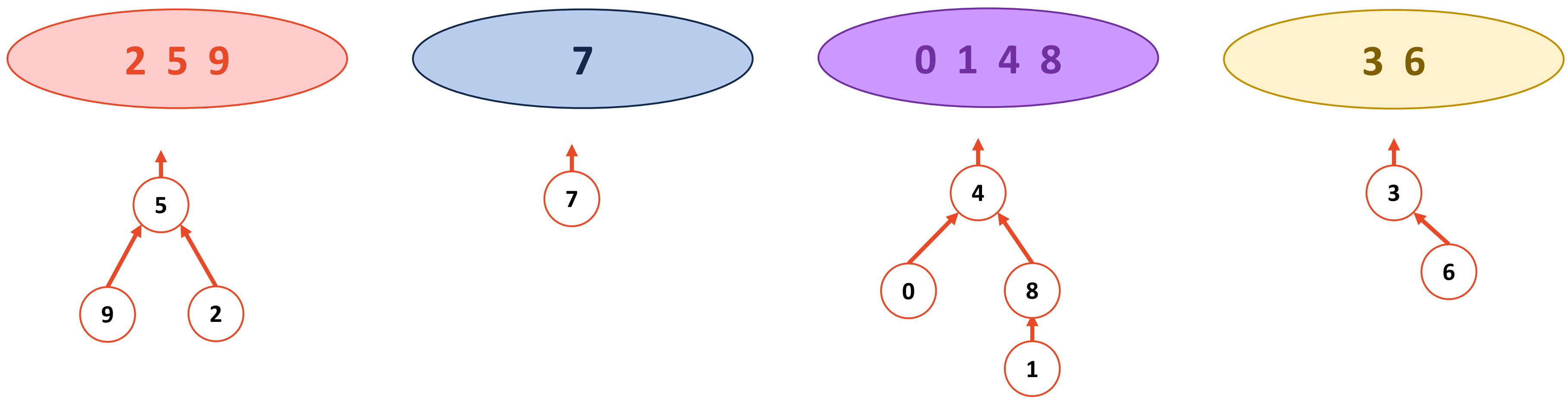
- The value of the array is:
  - **-1,** if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element

- We will call theses **UpTrees**:



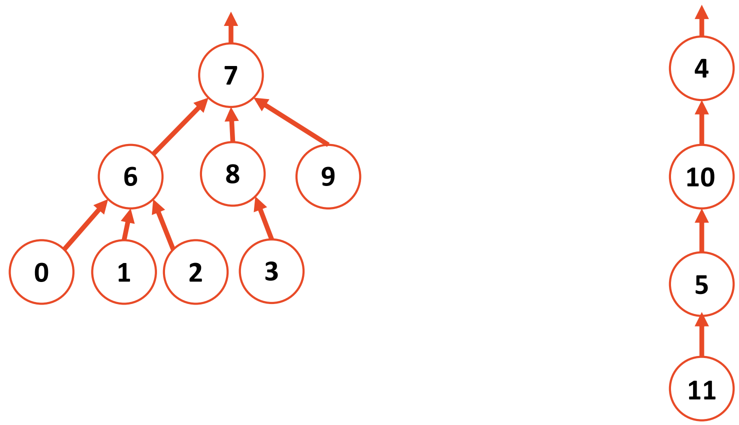
0	1	2	3
-1	**-1**	**-1**	**-1**

Disjoint Sets



0	1	2	3	4	5	6	7	8	9

Disjoint Sets – Smart Union



Union by height

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

***Idea**: Keep the height of the tree as small as possible.*

Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

***Idea**: Minimize the number of nodes that increase in height*

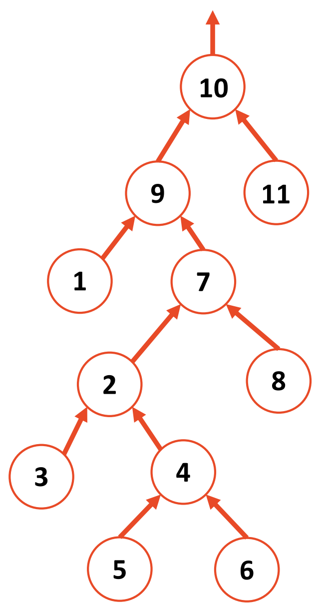
Both guarantee the height of the tree is: _____________.

Disjoint Sets Find

```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return find( s[i] ); }
}
```

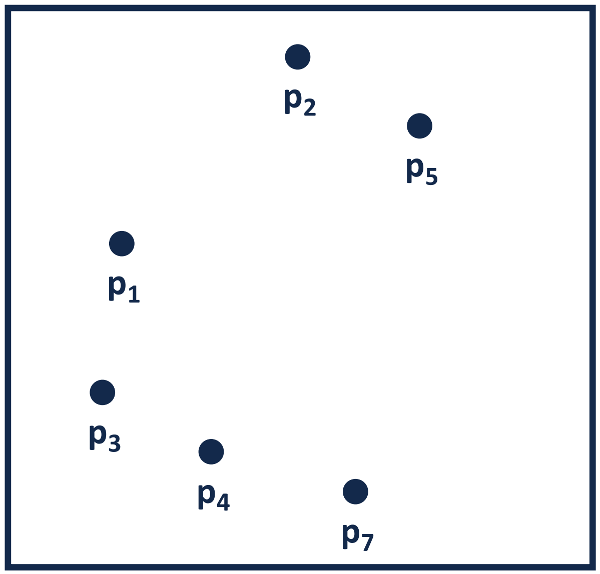
```
void DisjointSets::unionBySize(int root1, int root2) {
  int newSize = arr_[root1] + arr_[root2];

  // If arr_[root1] is less than (more negative), it is the larger set;
  // we union the smaller set, root2, with root1.
  if ( arr_[root1] < arr_[root2] ) {
    arr_[root2] = root1;
    arr_[root1] = newSize;
  }

  // Otherwise, do the opposite:
  else {
    arr_[root1] = root2;
    arr_[root2] = newSize;
  }
}
```

Path Compression

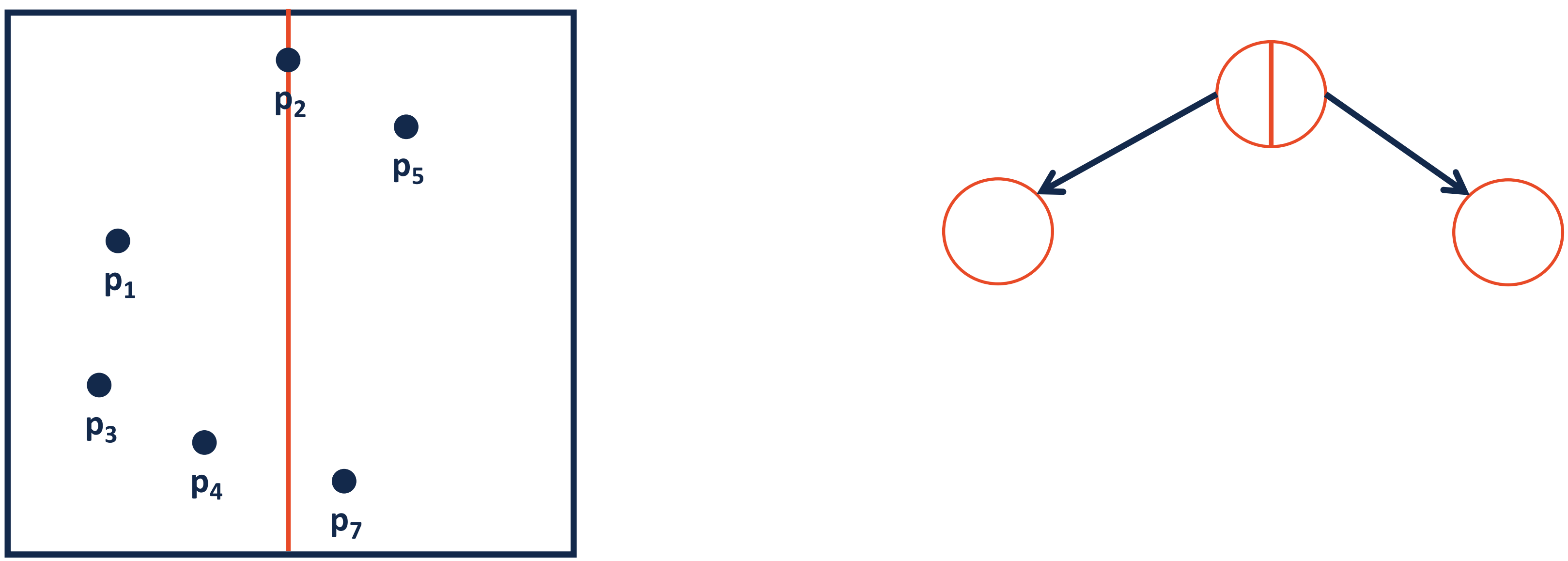


MP Mosaics and One More Tree

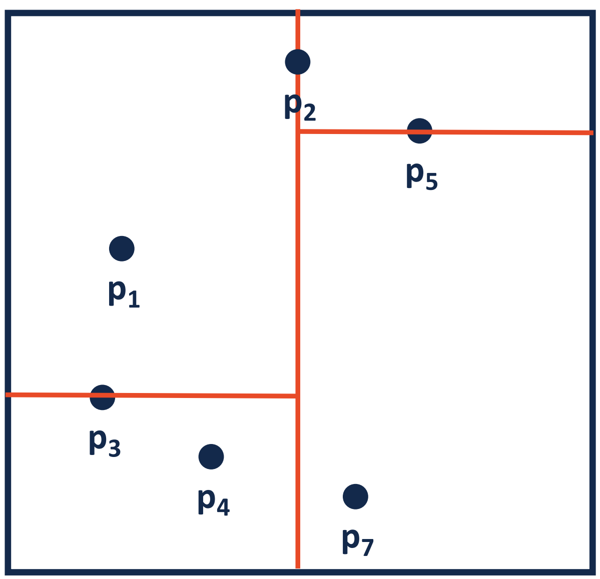
kD-Trees

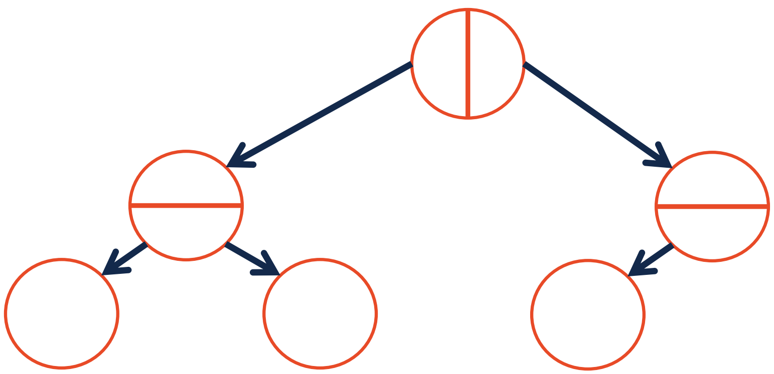


kD-Trees



kD-Trees





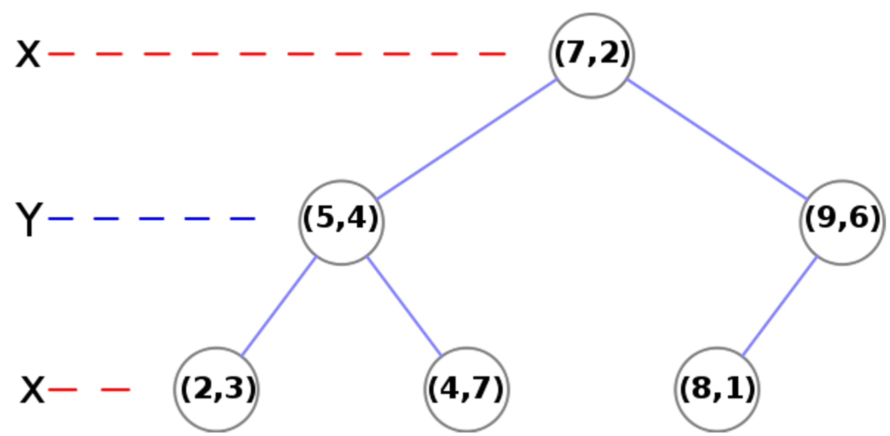
kD-Tree Constructor

How to construct your own kD-tree?

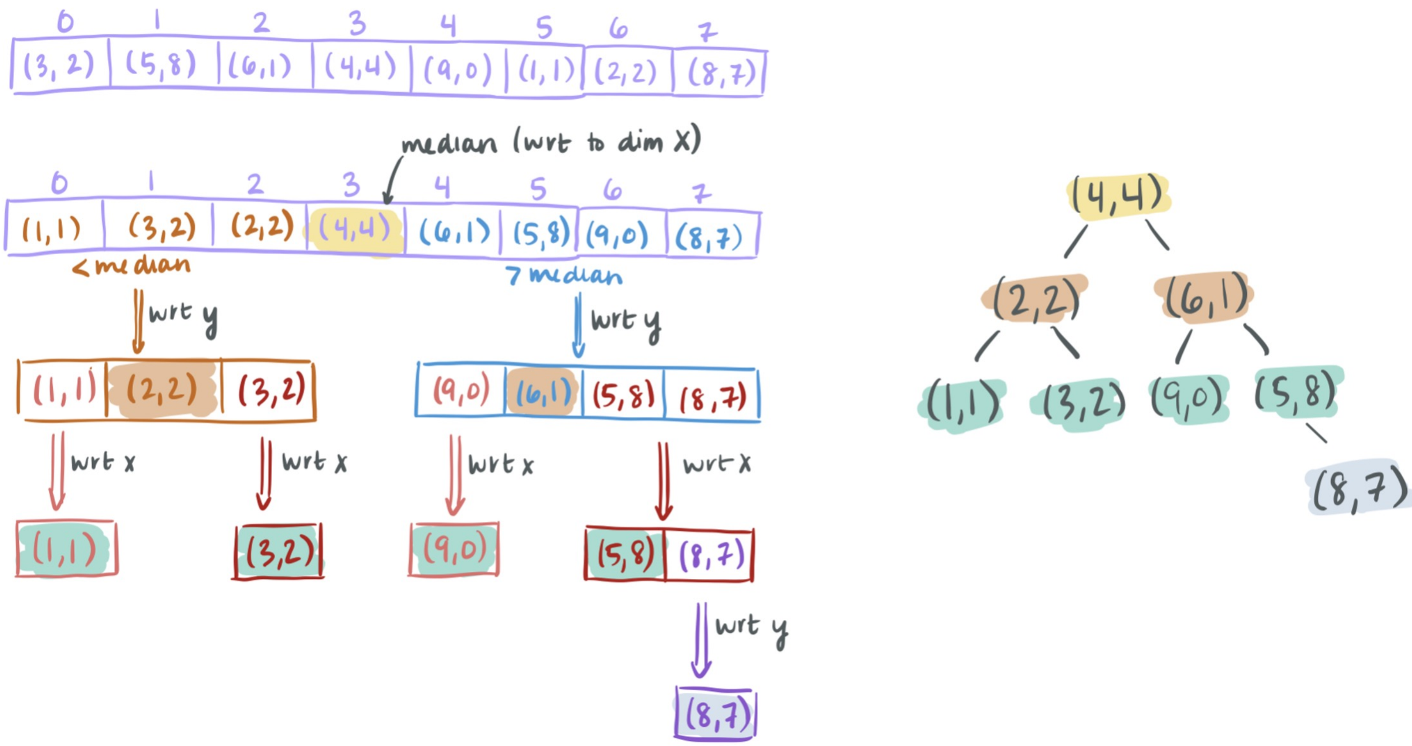
kD-Tree

- Data structures (trees) which are often used to find the *nearest neighbor* of a *k*-dimensional point
  - *Why you should care:* actually very applicable to real world scenarios!
- **kD-Trees** are used to organize Points in *k*-dimensional space, for any $k > 0$

$$\begin{bmatrix}5\\4\end{bmatrix} \begin{bmatrix}2\\3\end{bmatrix} \begin{bmatrix}8\\1\end{bmatrix} \begin{bmatrix}9\\6\end{bmatrix} \begin{bmatrix}7\\2\end{bmatrix} \begin{bmatrix}4\\7\end{bmatrix}$$



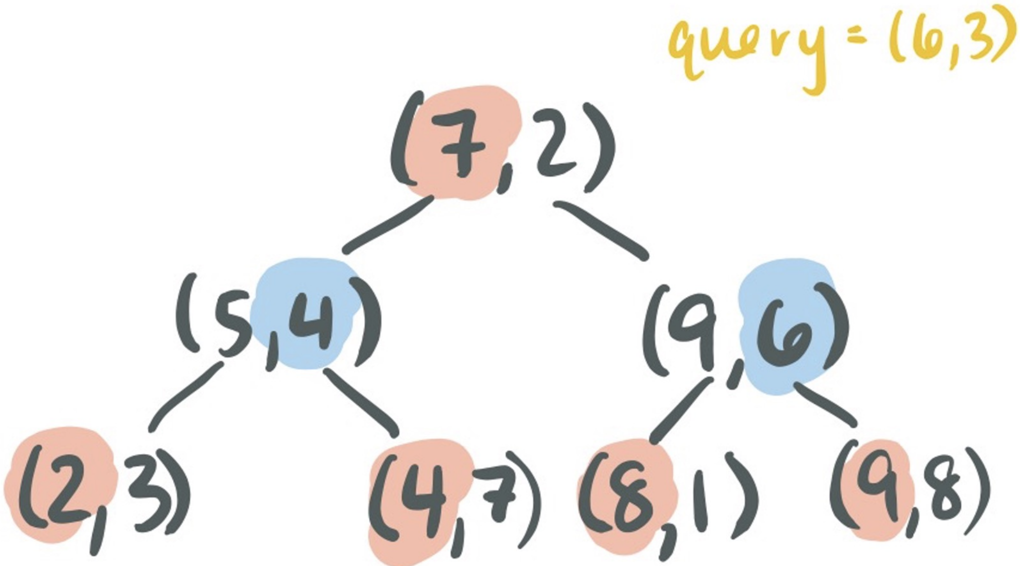
kD-Tree Constructor

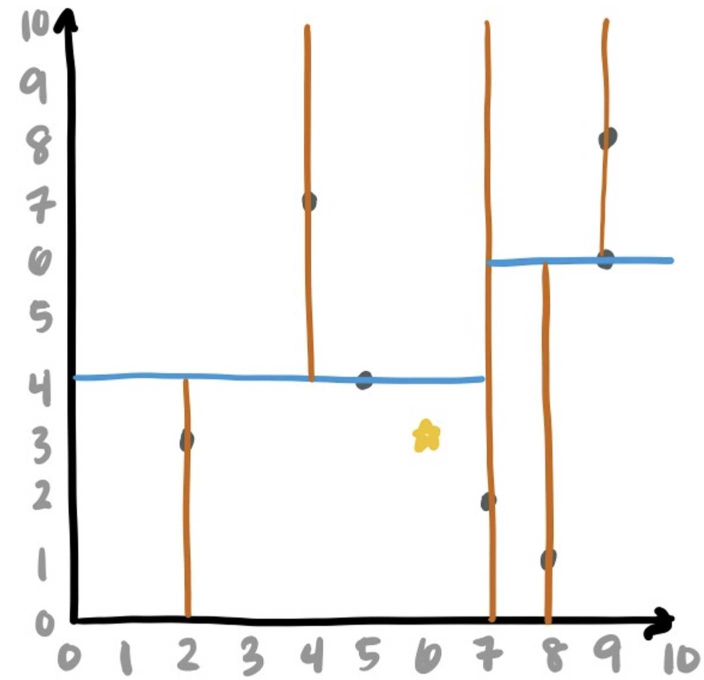


Nearest Neighbors

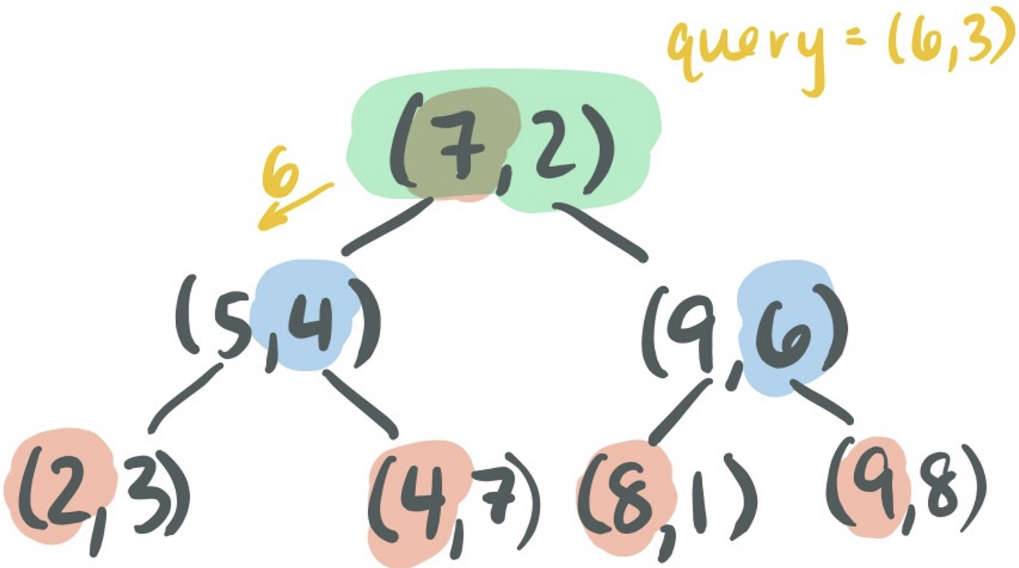
Some suggestions to keep in mind

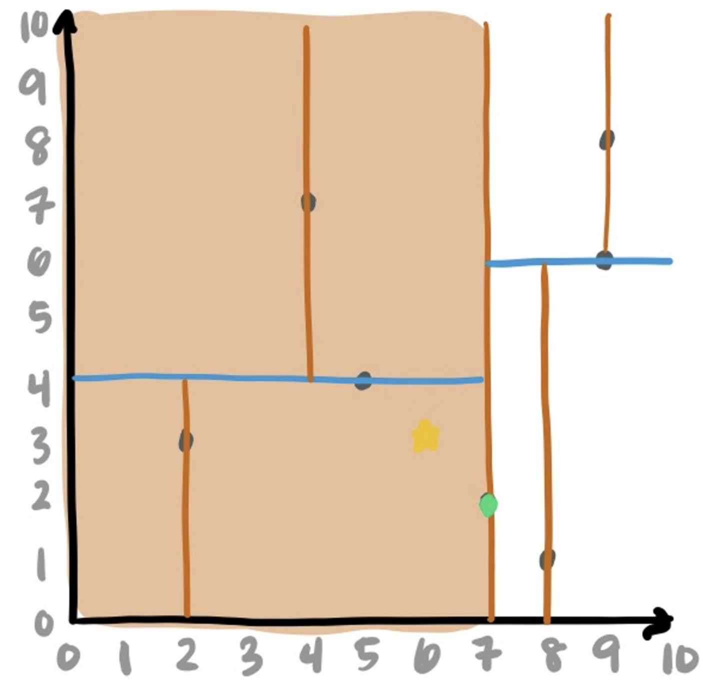
Nearest Neighbor - demo



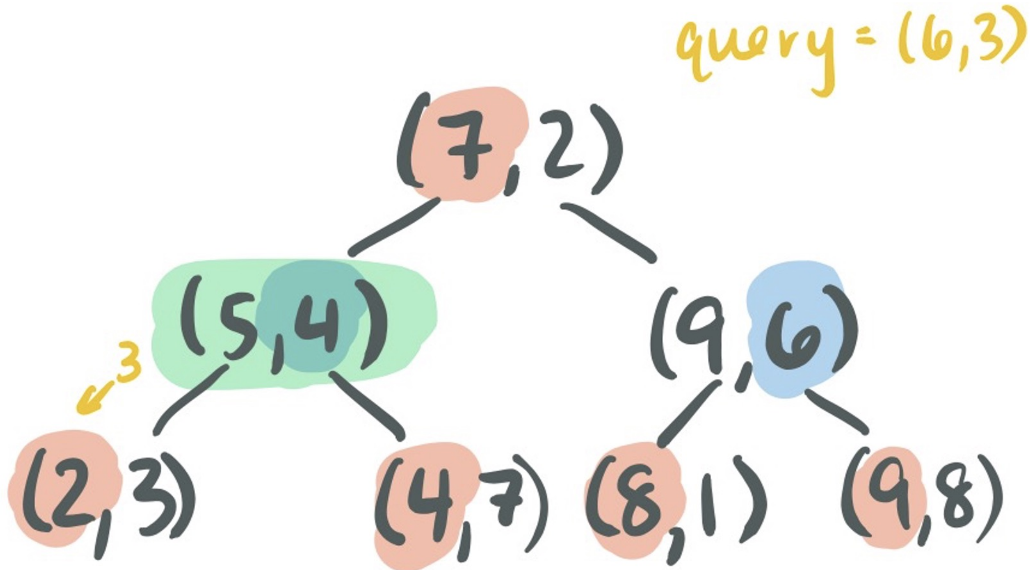


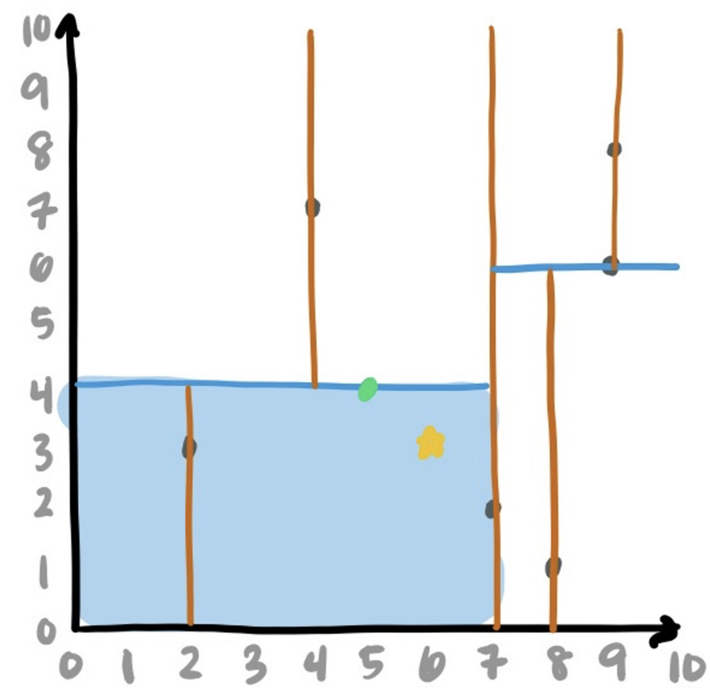
Nearest Neighbor - demo



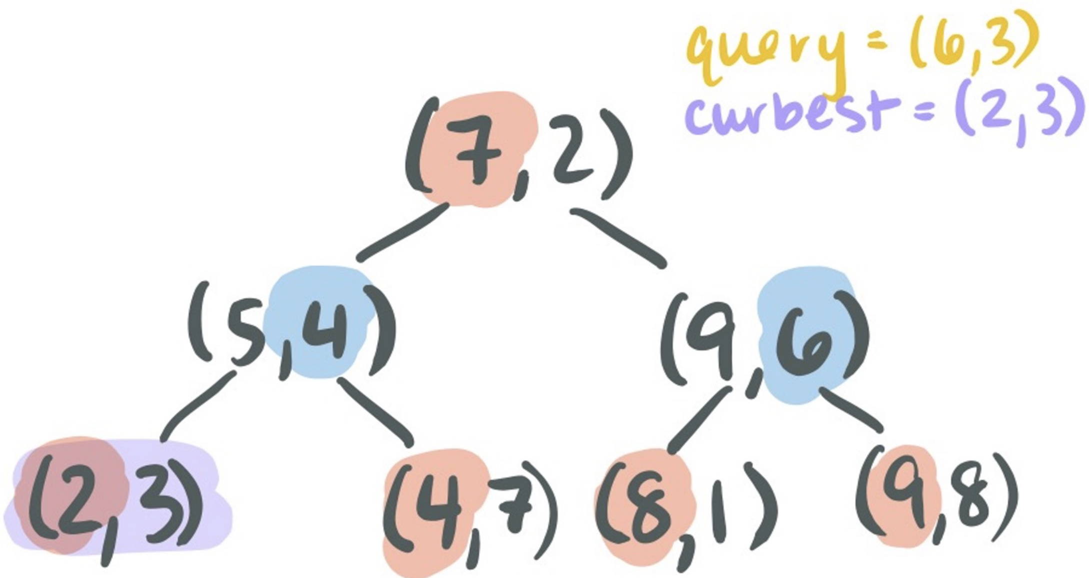


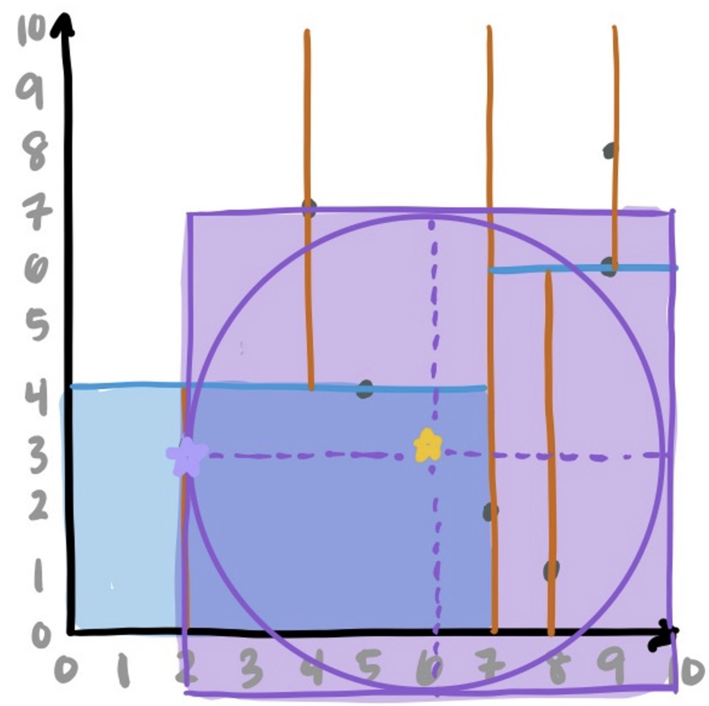
Nearest Neighbor - demo





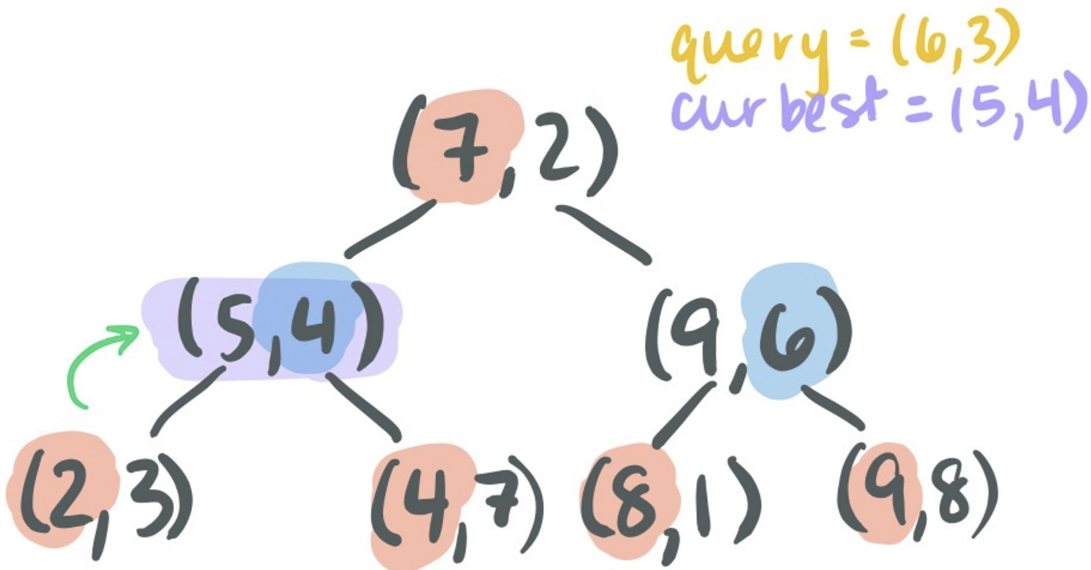
Nearest Neighbor - demo

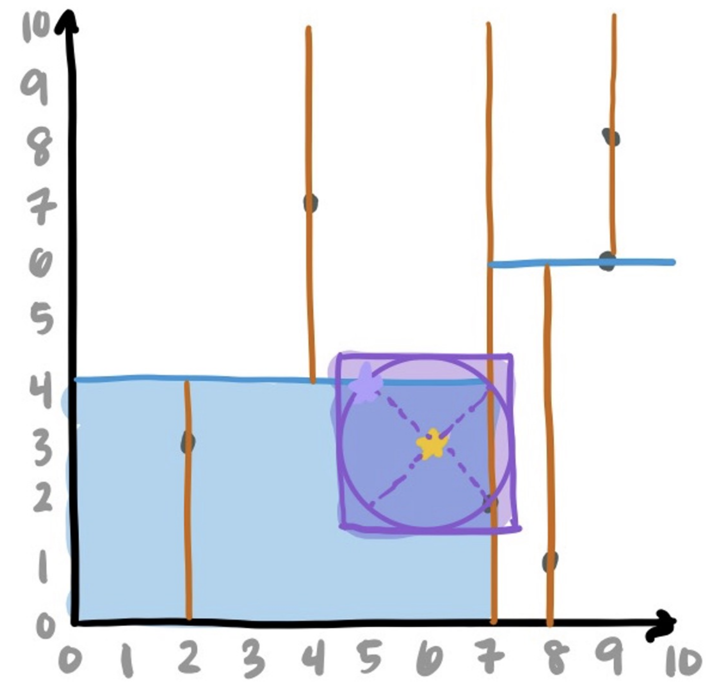




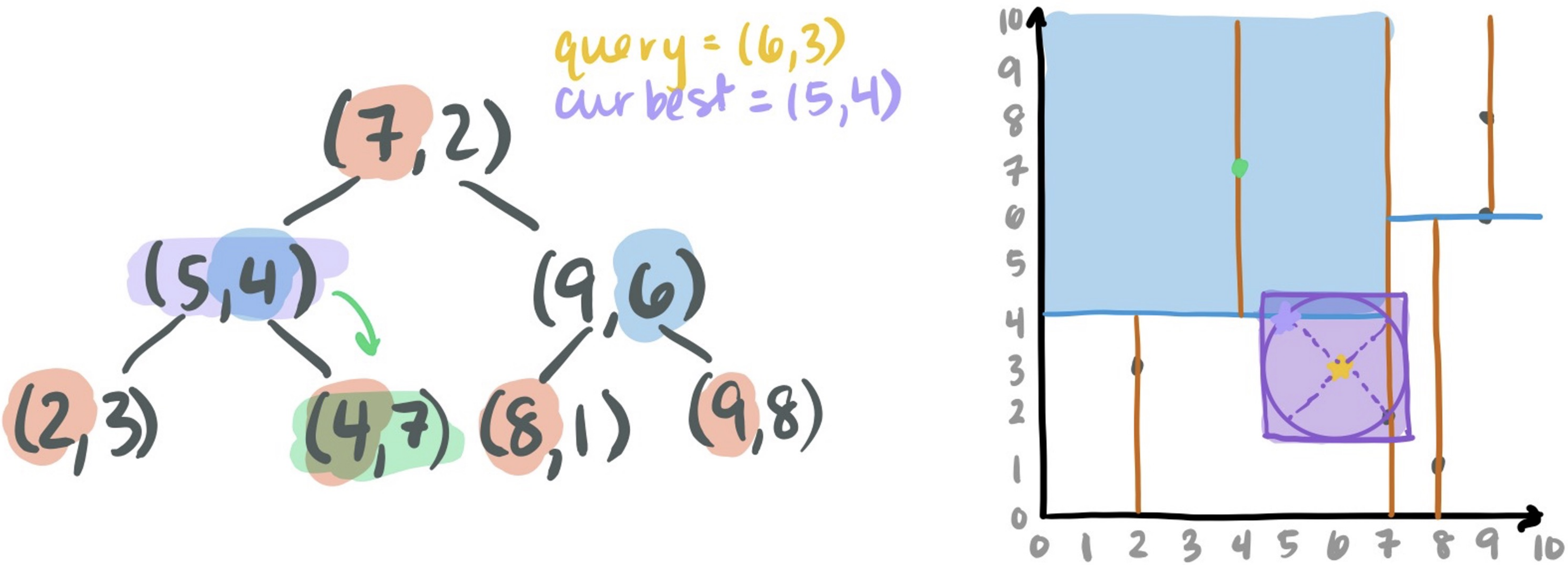
Nearest Neighbor - demo

Backtracking: start recursing backwards -- store “best” possibility as you trace back



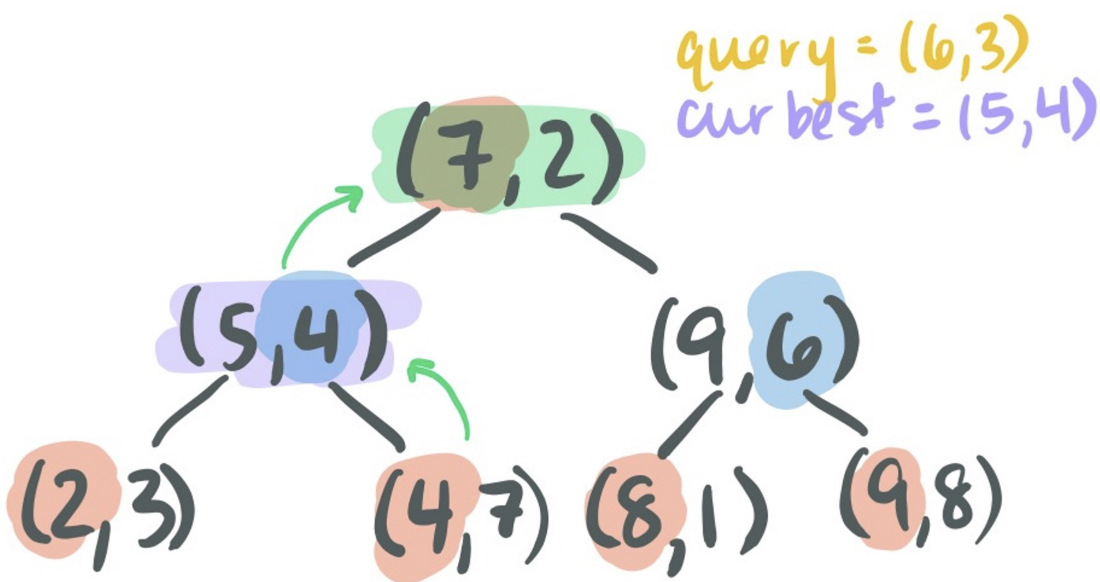


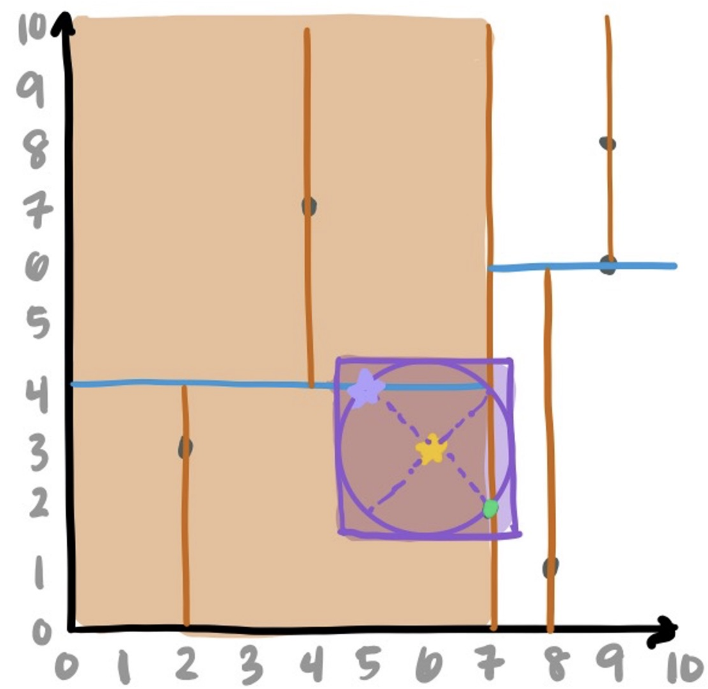
Nearest Neighbor - demo



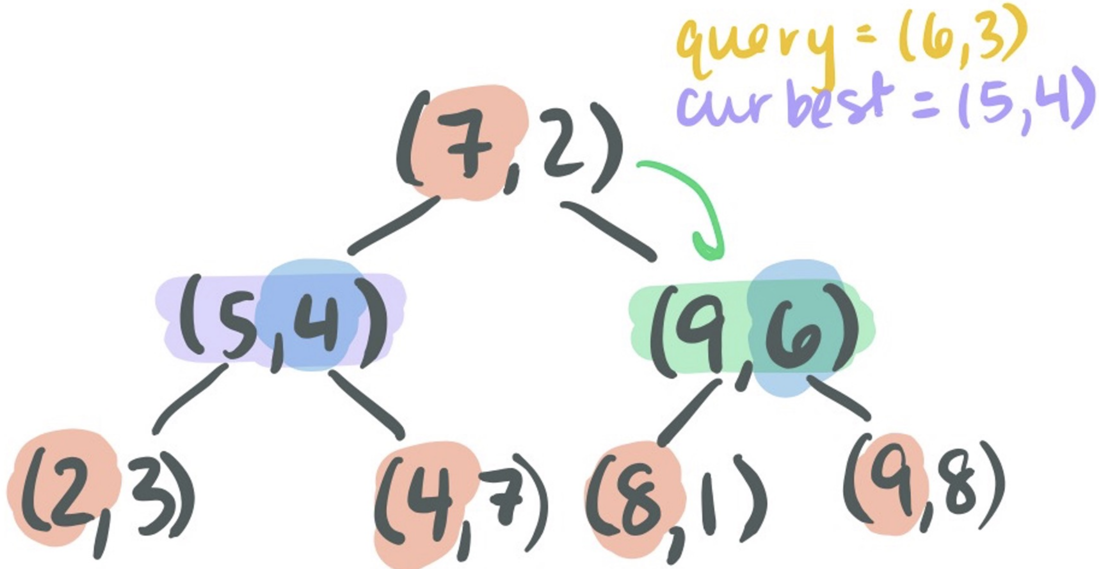
Nearest Neighbor - demo

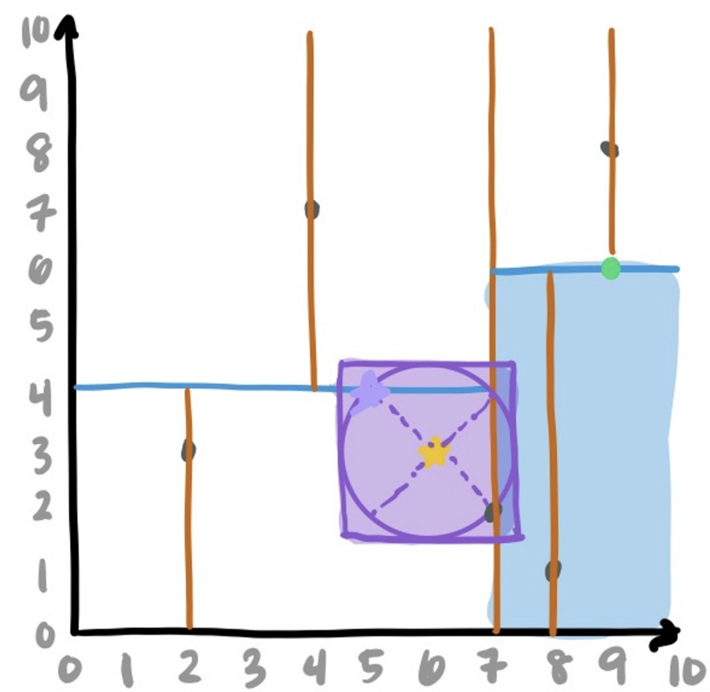
On ties, use `smallerDimVal` to determine which point remains `curBest`





Nearest Neighbor - demo





Nearest Neighbor - demo