

Delayed
class start
(No access to projector system)

Data Structures

KD Tree 2 (quickSelect and Lambda)

CS 225

September 25, 2026

Brad Solomon



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Learning Objectives

Review KD Tree construction (and search)



Introduce quickSelect algorithm

Discuss lambda functions

MP - MOSaic

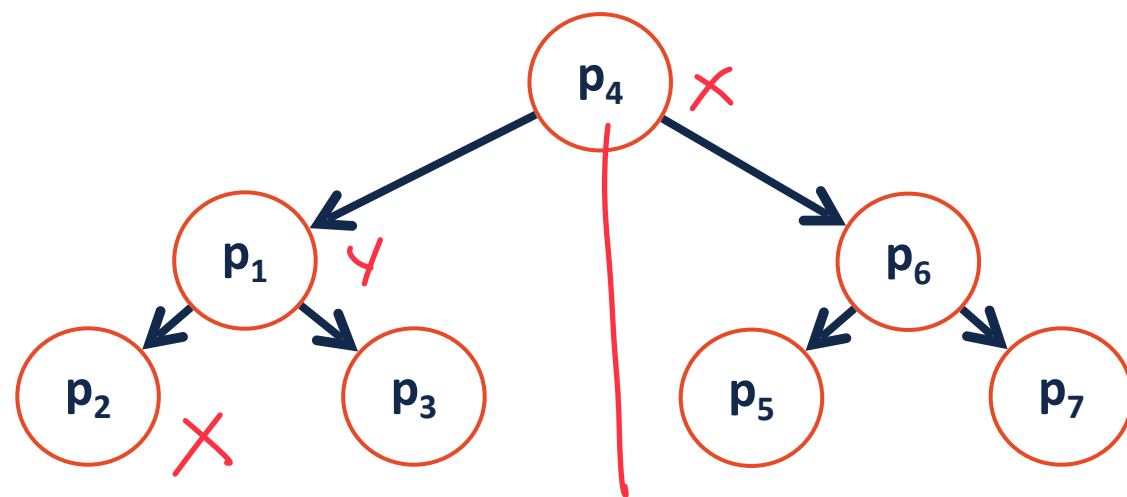
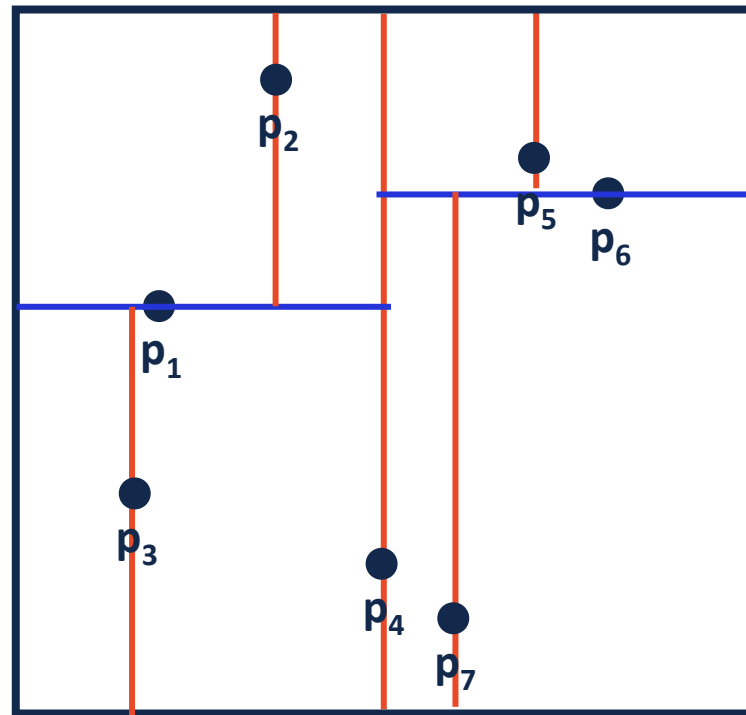


KD Tree Construction

Consider points in 2D: $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$

Tree Construction:

1. Find median point along a particular dim
2. Recurse left and right, **swapping** dimension



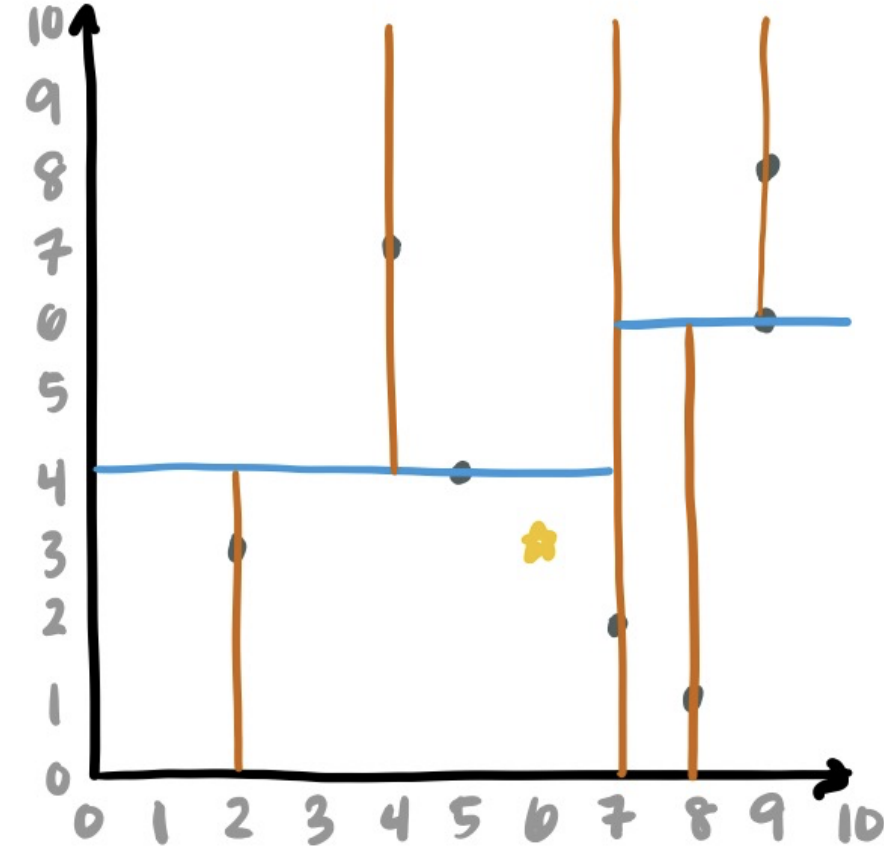
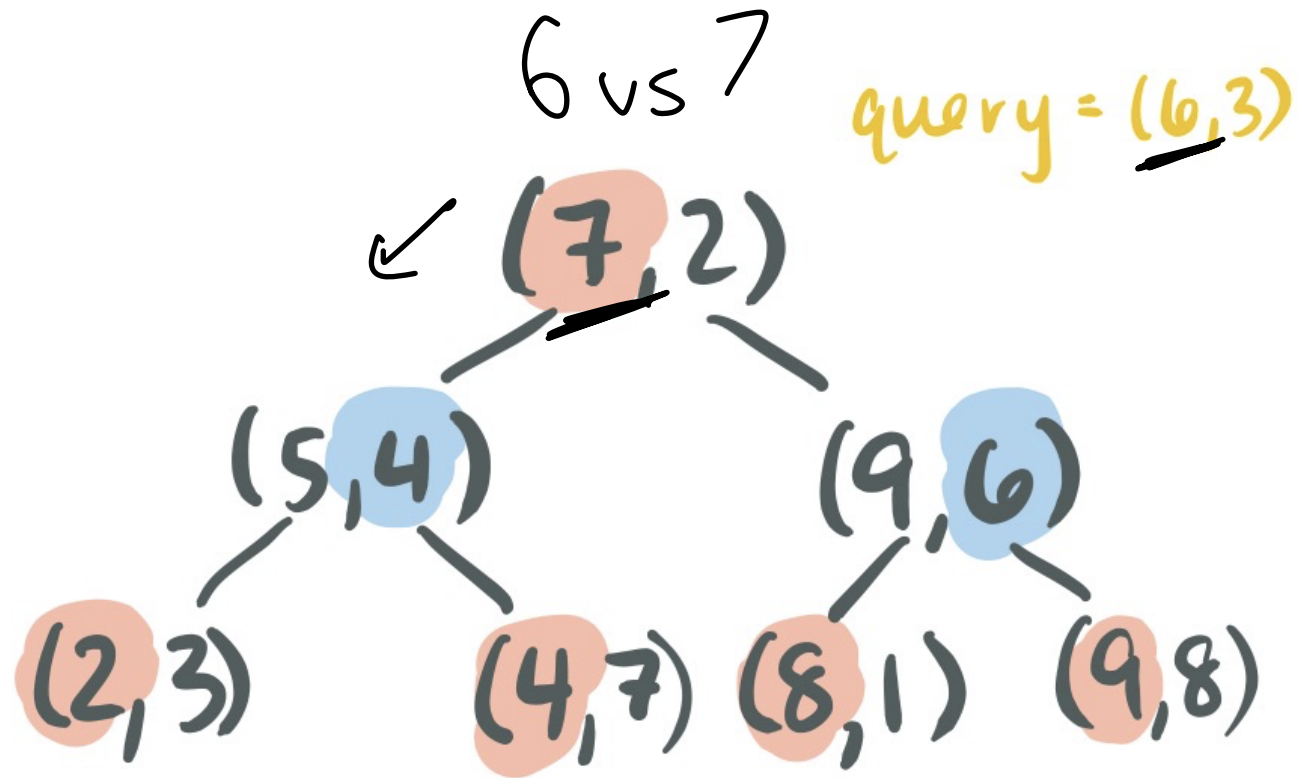
Efficiently identifying median point in range

Last class we assumed we could find median point trivially!

... But how do we actually do this?

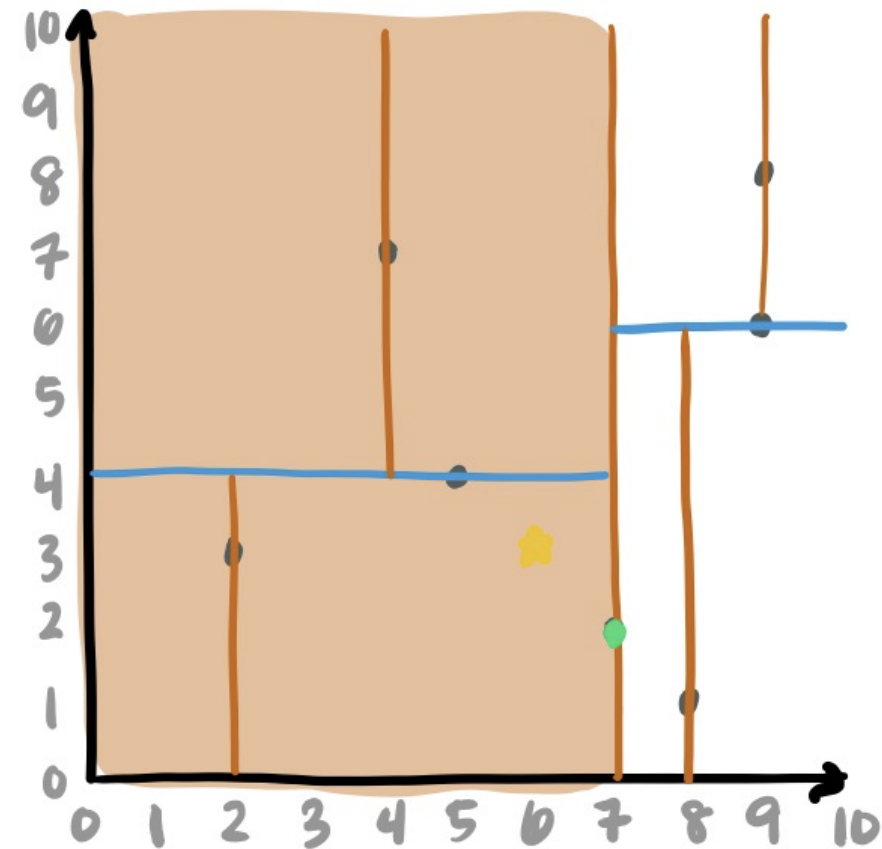
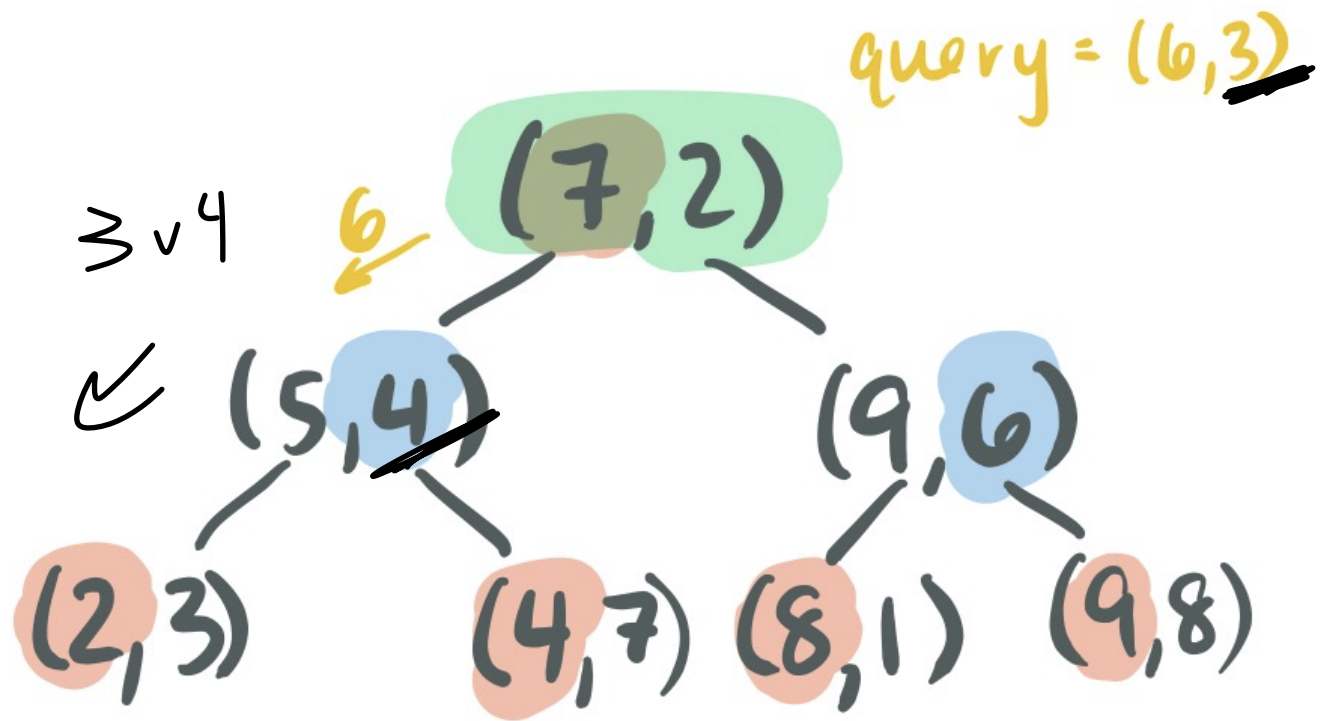
Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first...



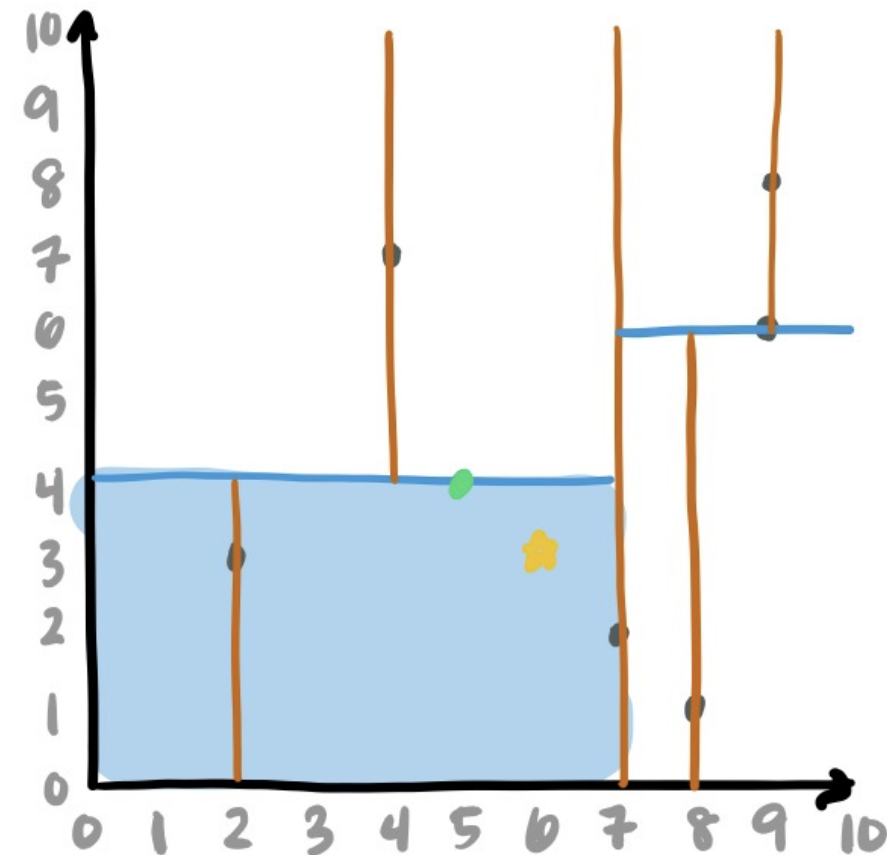
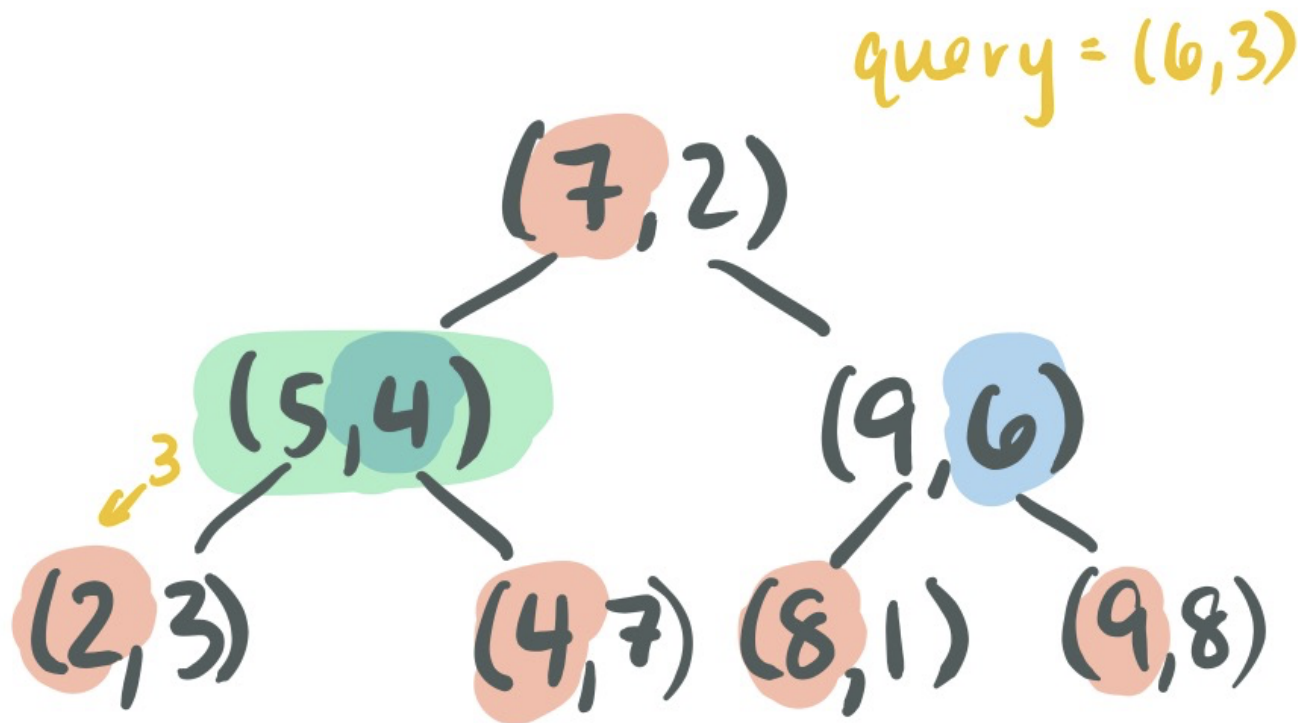
Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first...



Nearest Neighbor: k-d tree

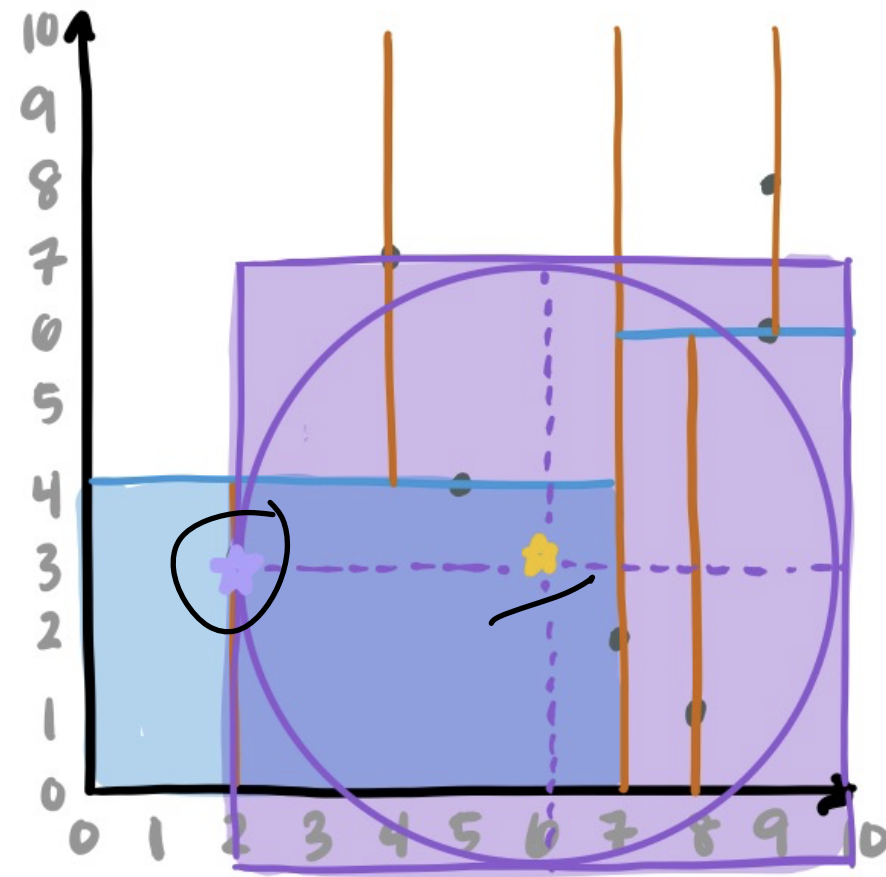
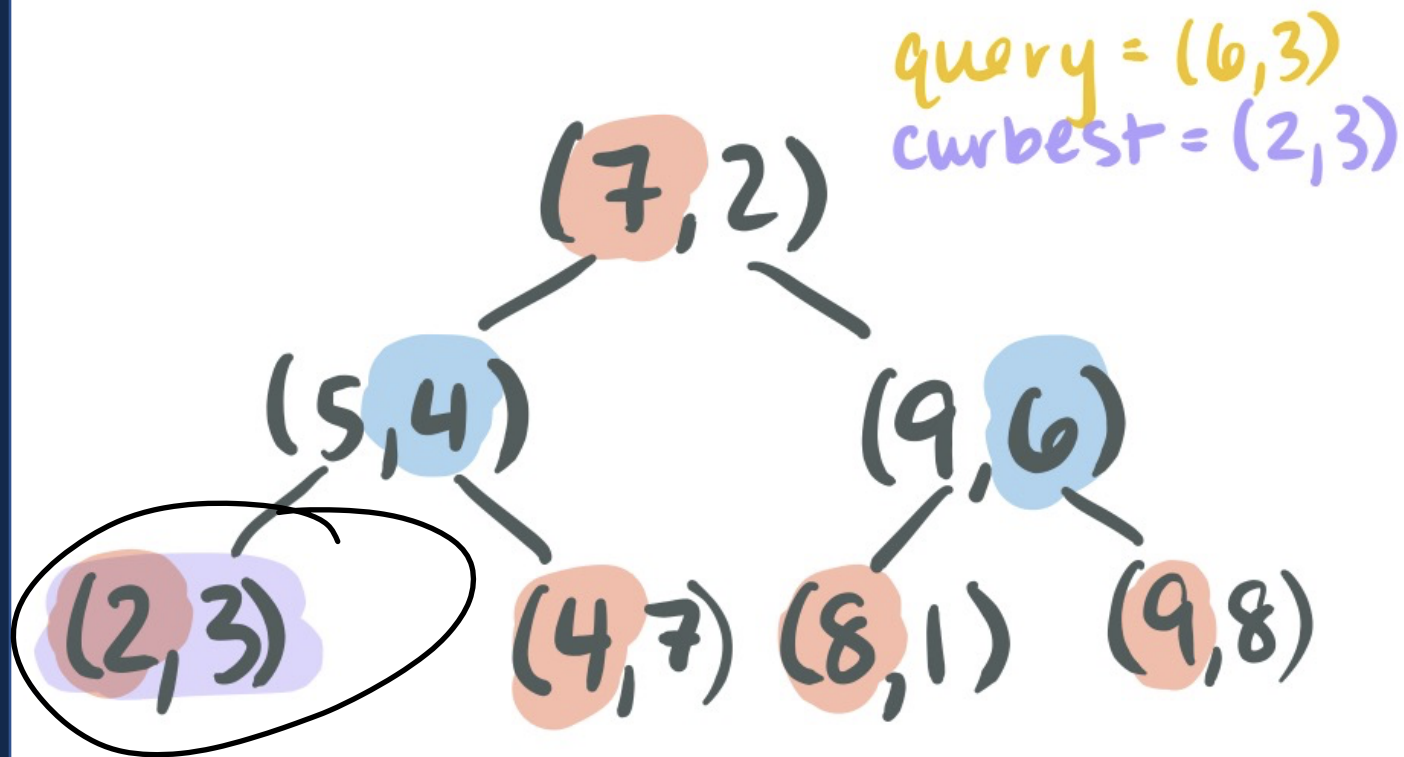
When querying a k-d tree, it acts like a BST* at first...



Nearest Neighbor: k-d tree

When querying a k-d tree, it acts like a BST* at first...

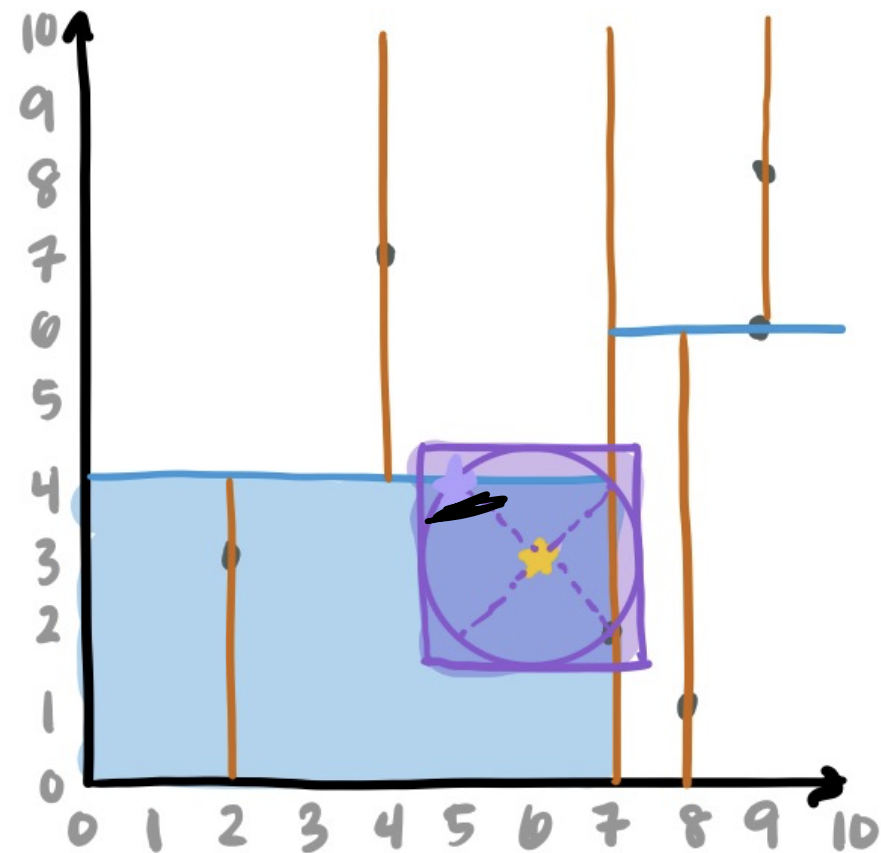
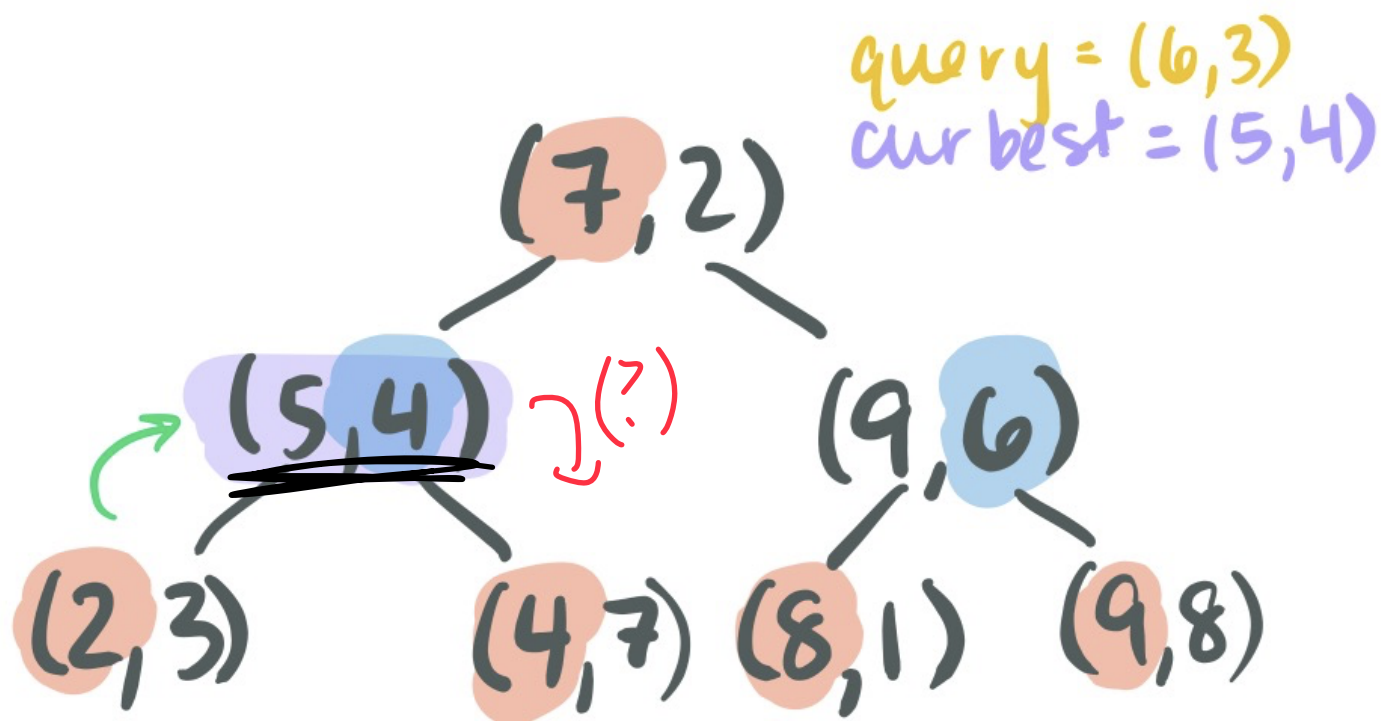
... But if we don't find exact match, have to find nearest neighbor



Nearest Neighbor: k-d tree

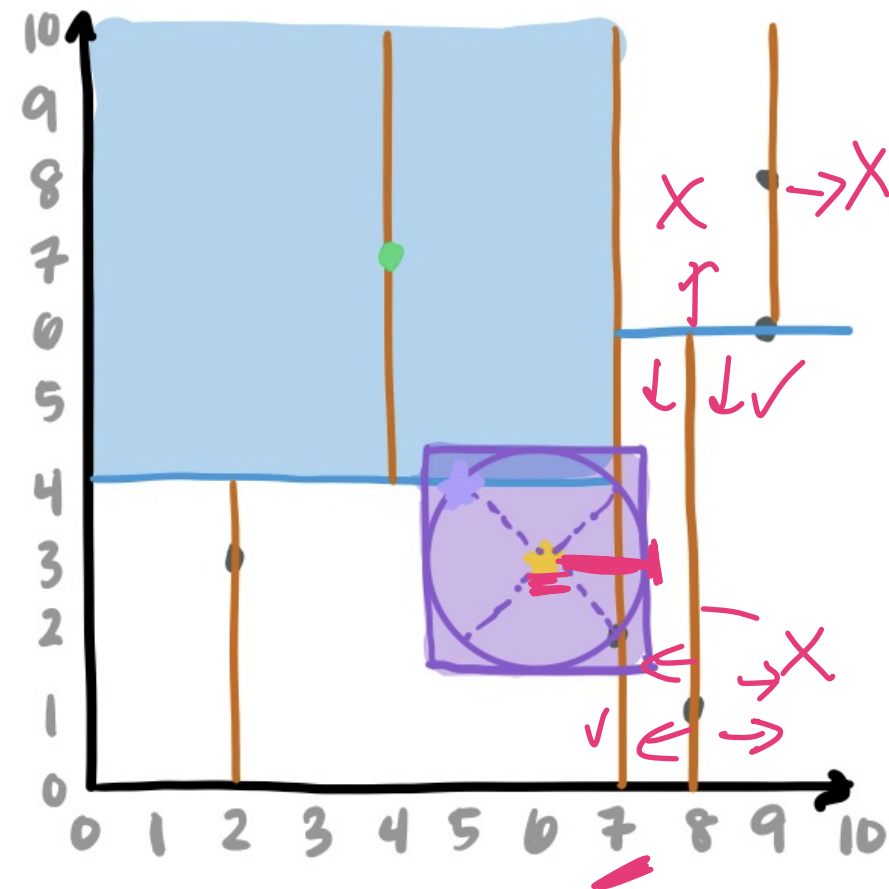
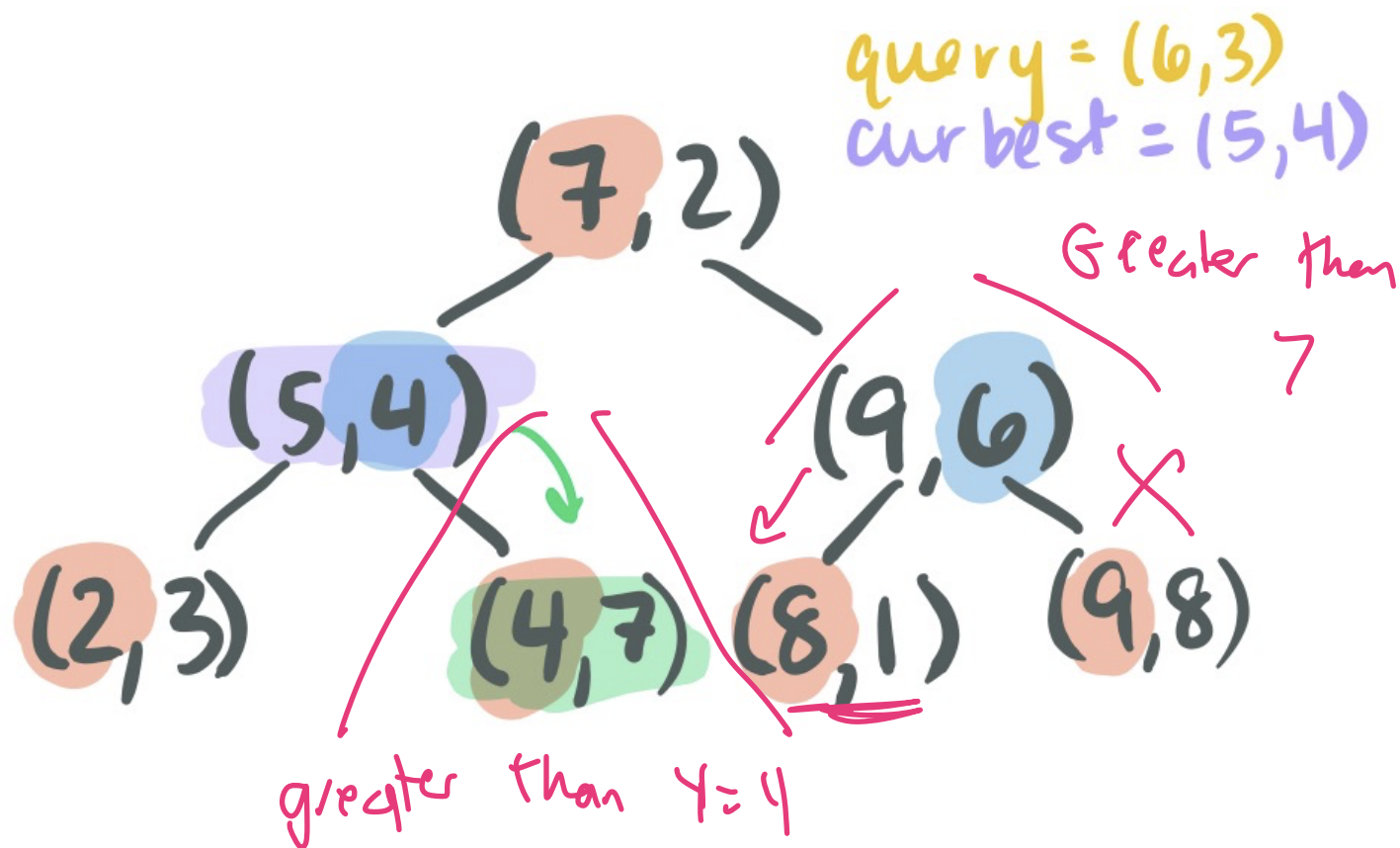
Backtracking: start recursing backwards -- store "best" possibility as you trace back

compute against root on walk back



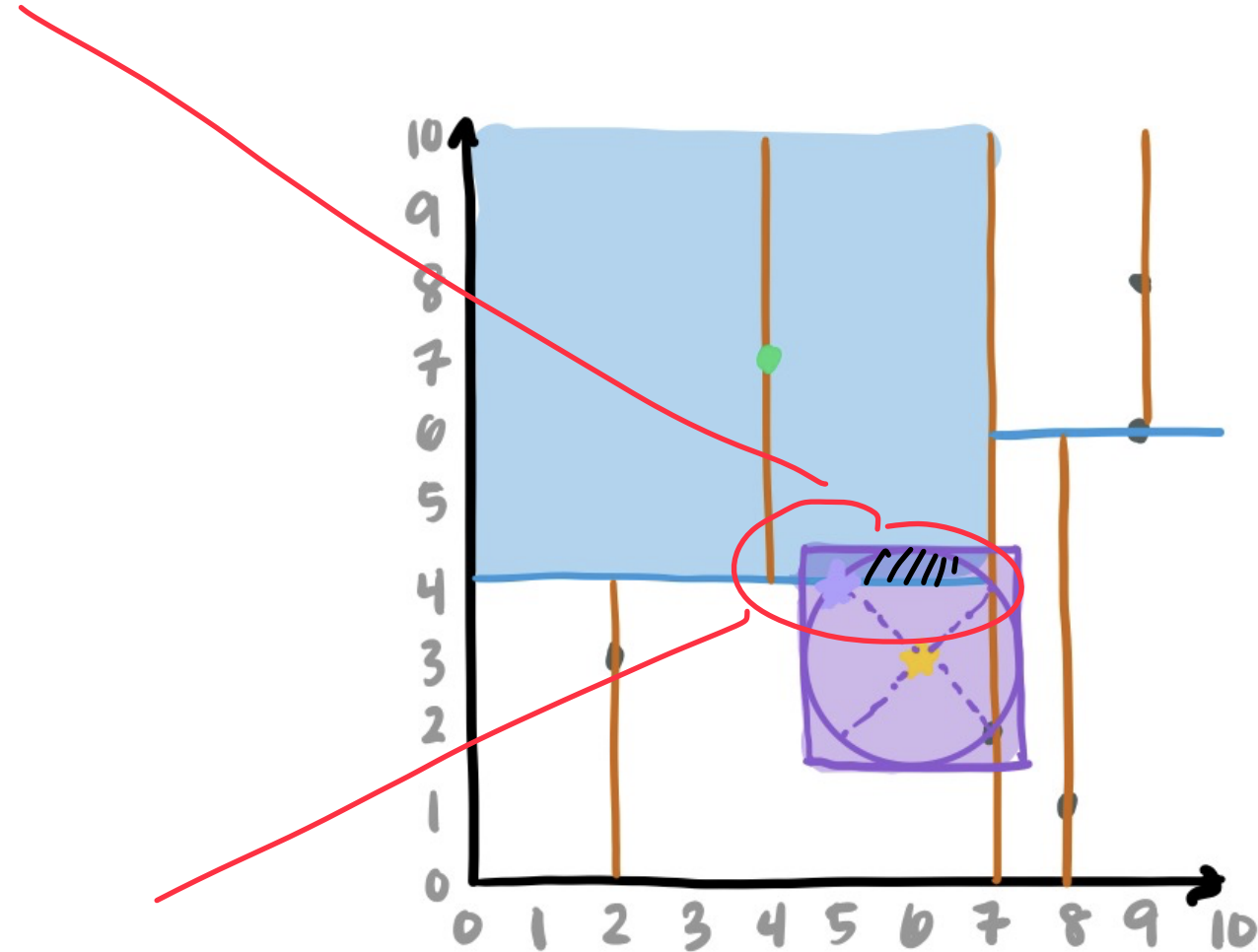
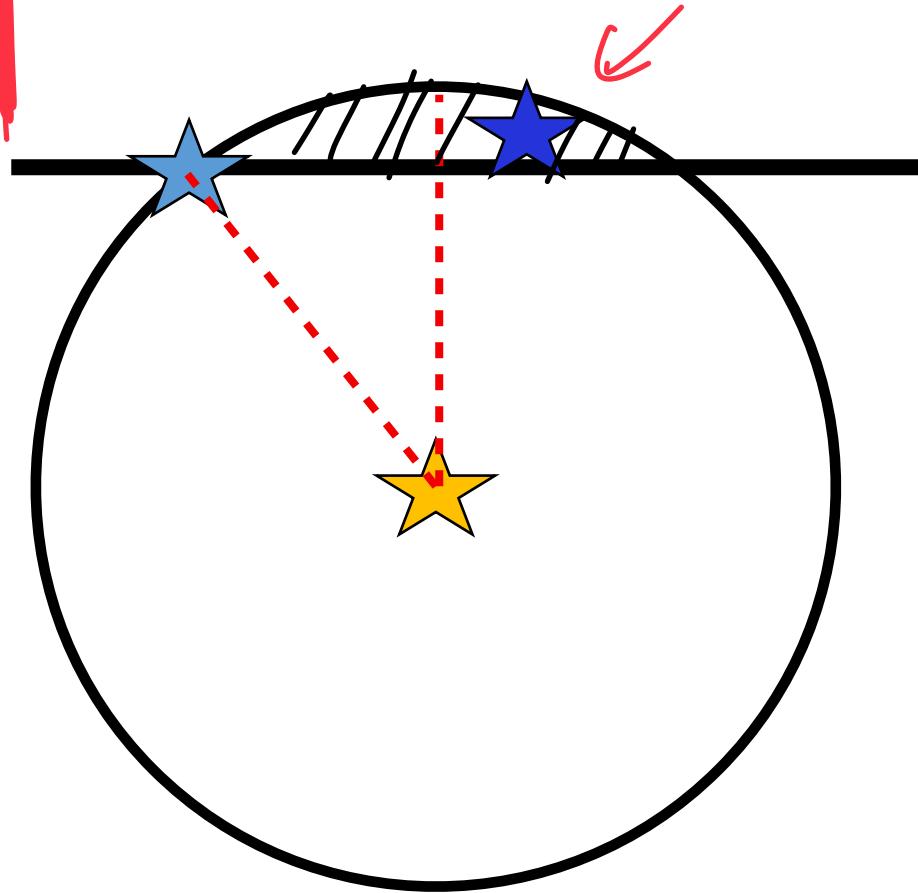
Nearest Neighbor: k-d tree

what branches must $\perp$ search?



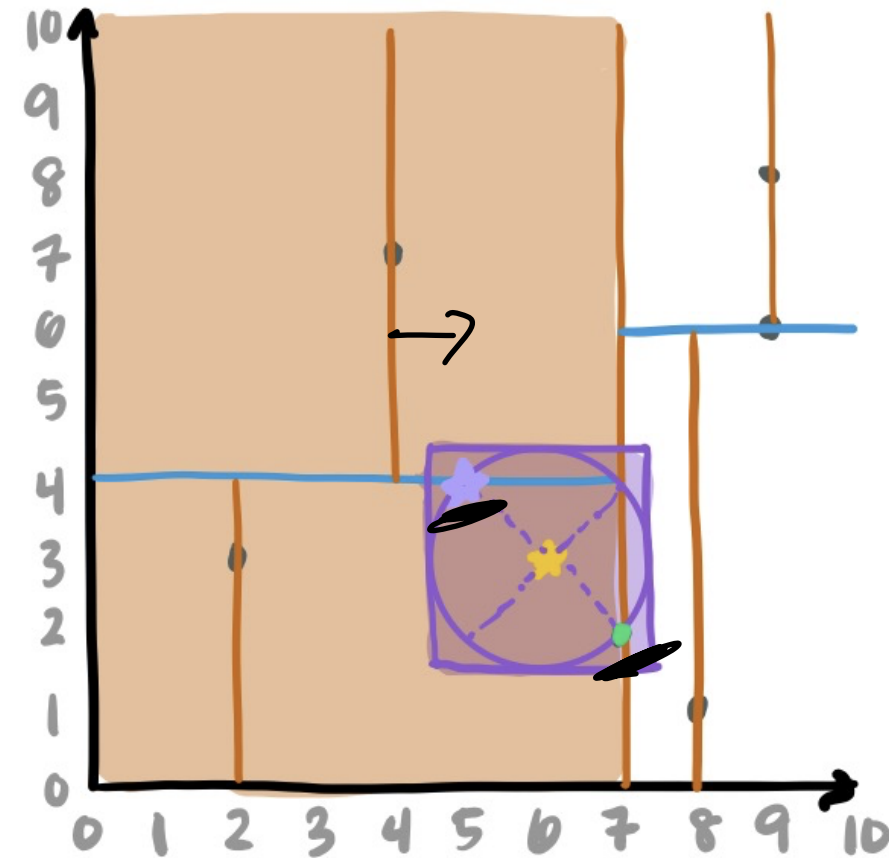
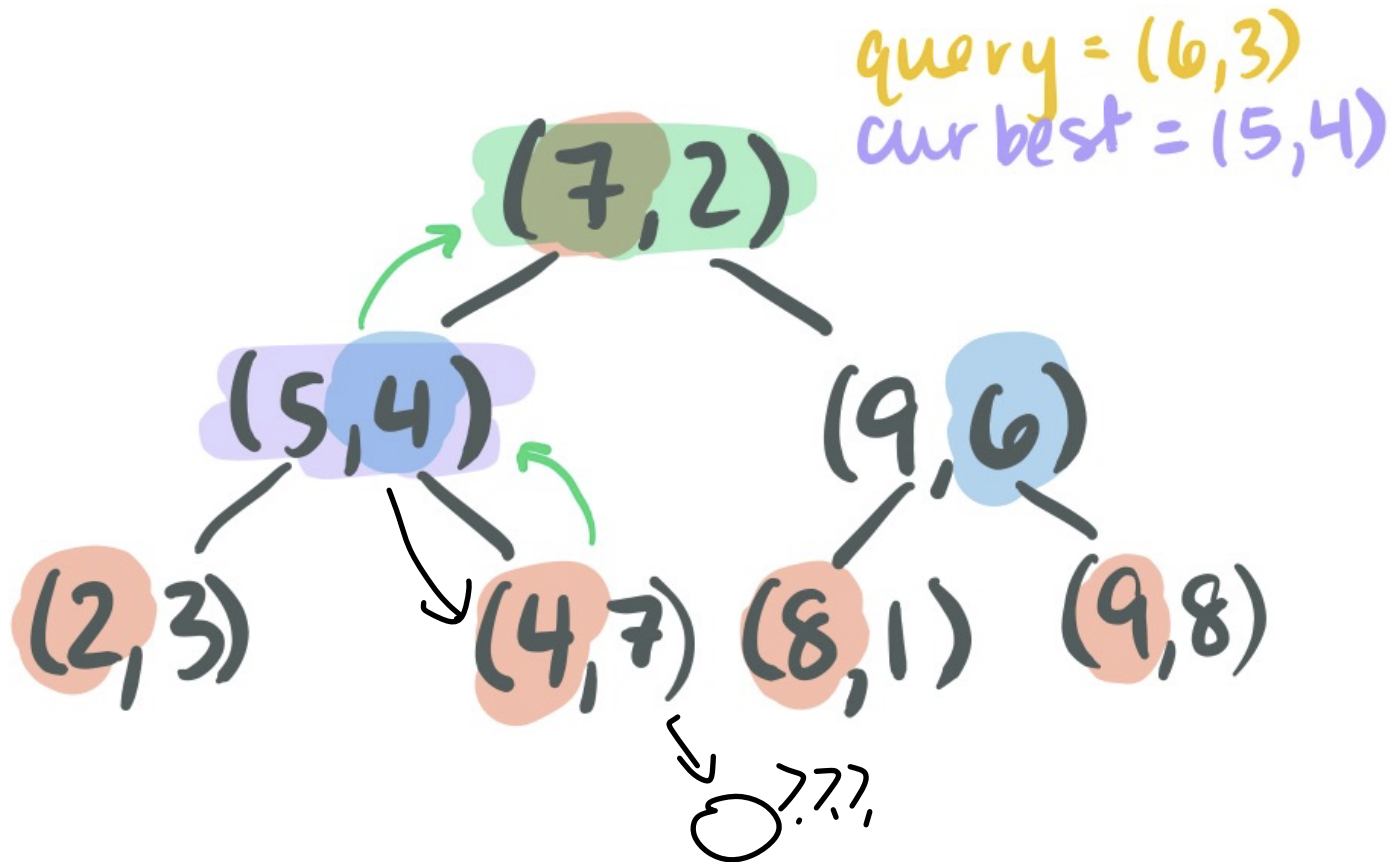
Nearest Neighbor: k-d tree

May have to recursively check other branches of tree — **why?**



Nearest Neighbor: k-d tree

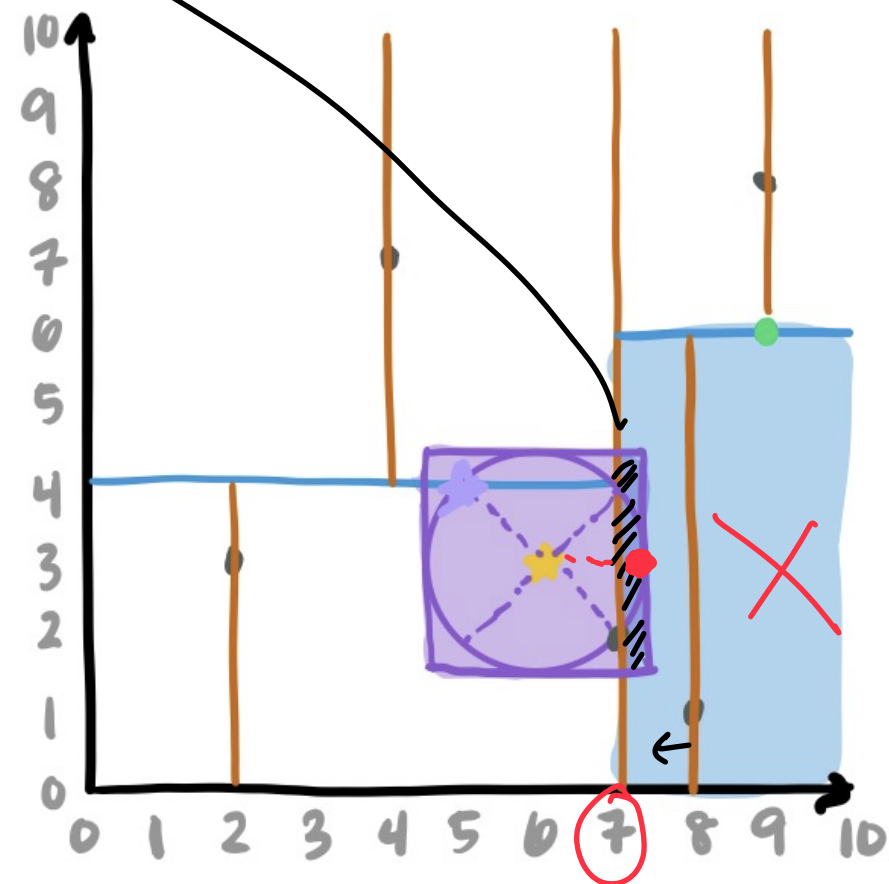
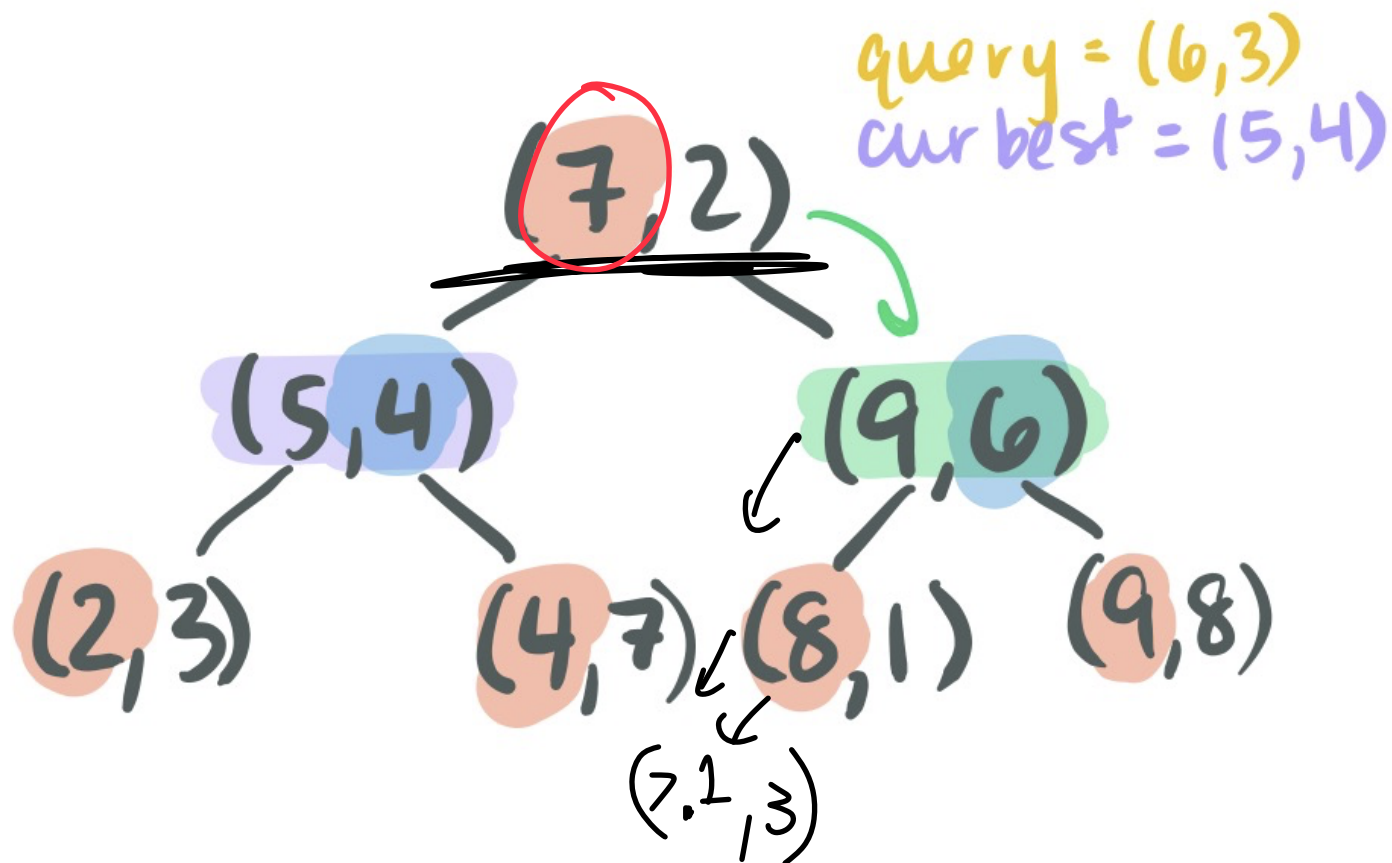
On ties, use smallerDimVal to determine which point remains curBest



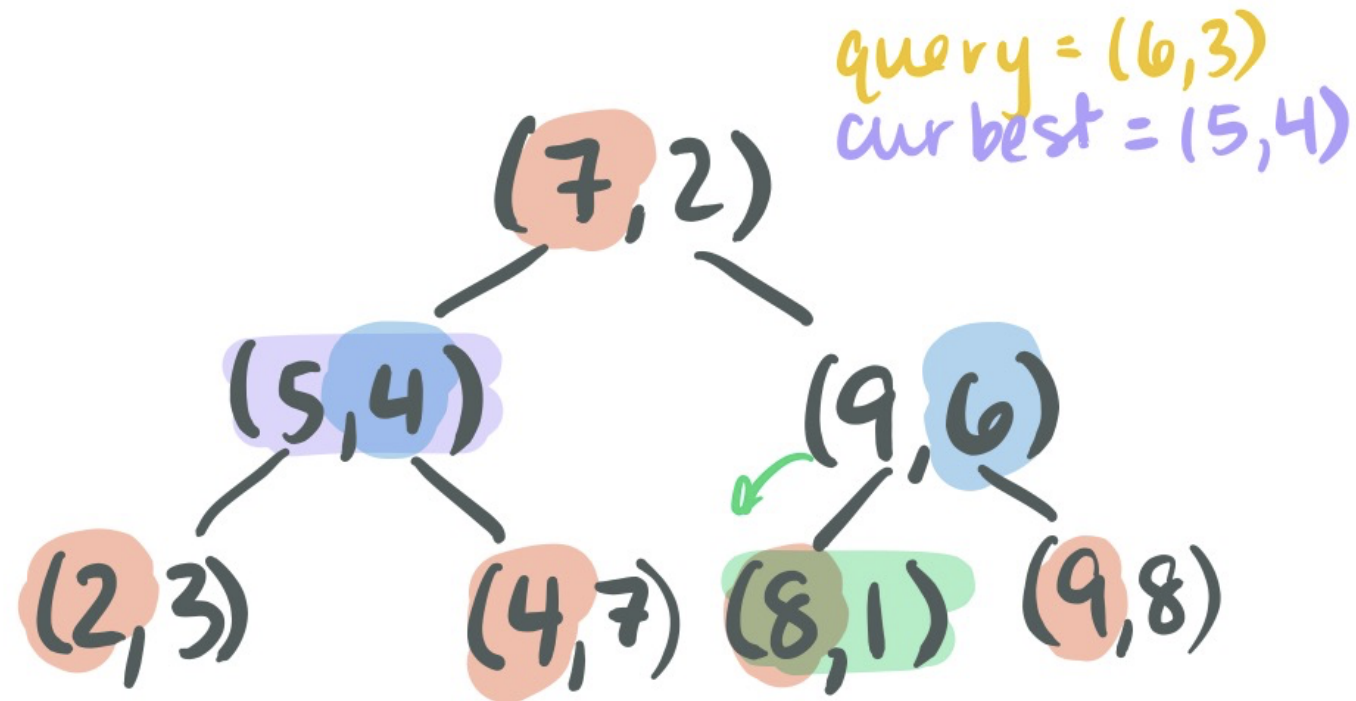
Nearest Neighbor: k-d tree

Why do we need to explore this subtree?

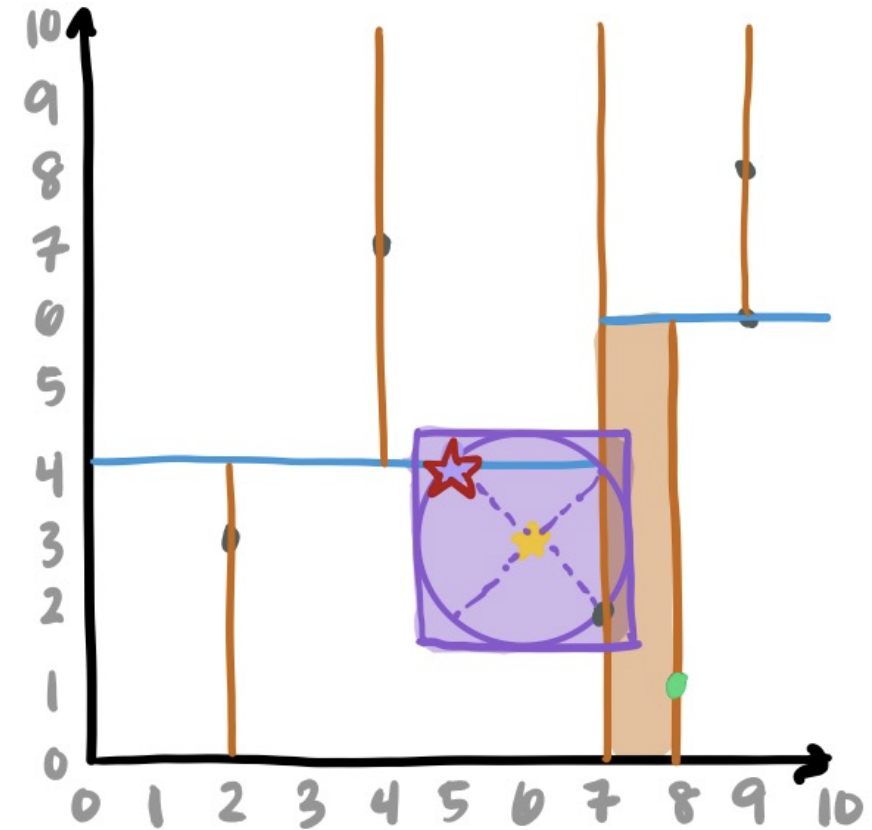
If point here, would be better
Must search!



Nearest Neighbor: k-d tree



BEST: (5,4)



K-D Tree Runtime Efficiency

Big $O_{\text{bad}} n$

Because our search radius can touch every point, search is $O(N)$!

Bad assumption

... But if my data is well-formed (uniform distribution), we can make an argument that the average-case search is actually $O(\log n)$

Intuition: More points == more partitions == smaller search radius

MP_Mosaics shows off this theory vs practice!

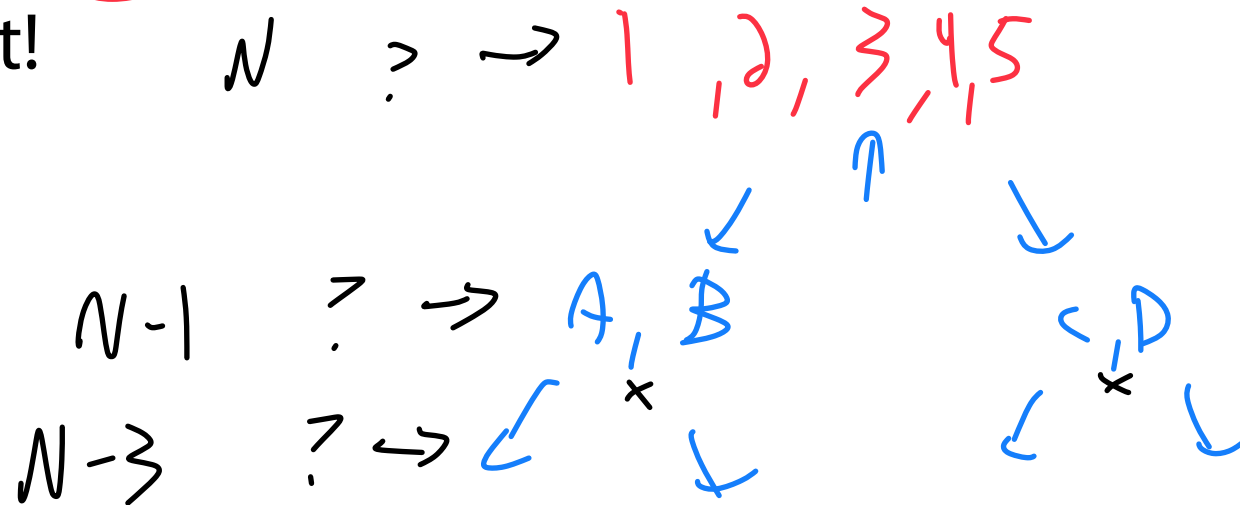
Efficiently identifying median point in range

Last class we assumed we could find median point trivially!

... But how do we actually do this?

Naive: Sort the list!

Problem:



$(1, A), (2, B), \dots$

$\text{perfect} \downarrow$

$\log_2 n \leq \log_2 \frac{n+1}{2} - 1$

$O(\log n)$

Efficiently identifying median point in range

Last class we assumed we could find median point trivially!

... But how do we actually do this?

Naive: Sort the list!

Problem: We have to do this for each node!

Given median partition, at level k there are 2^k nodes

Each node has to sort $\frac{n}{2^k}$ points to find its own median.

Runtime: $\sim O(n \log n)$ per level * $O(\log n)$ levels = $O(n \log^2 n)$



Efficiently identifying median point in range

Last class we assumed we could find median point trivially!

... But how do we actually do this?

Better: We don't need to sort all items, we just need to find median!

2) We want to partition around the median value

... without fully sorting the list

How can this be useful for K-D Tree Construction?

N items give me $K = N/2$ smallest item

median

QuickSelect Algorithm

Join Code: 225



Goal: Find the k-th smallest value without sorting

1. Pick a pivot (randomly)

placed the kth smallest value in correct spot

2. **Partition** into smaller / larger around the pivot value

3. Recurse into the partition that contains k-th smallest value

When do we stop this recursion? What is our base case?

1) when kth smallest value is partition
~10%

2) when entire array sorted
~



3) when partition is single item
50%

4) when we have k partitions
~10%

QuickSelect Algorithm

Goal: Find the k-th smallest value without sorting

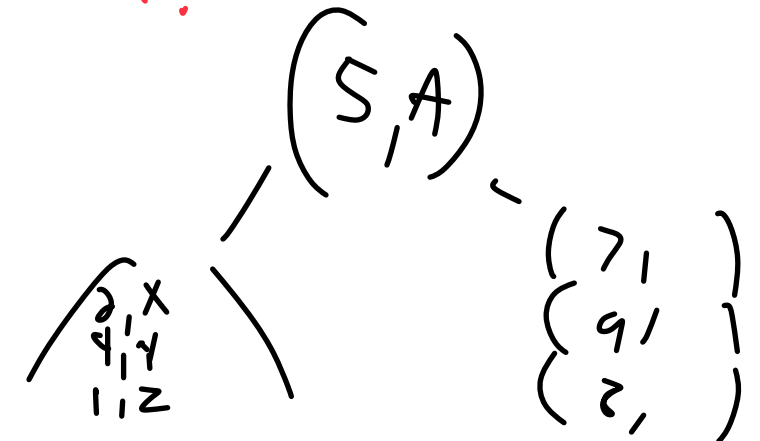
Ex: Find the ~~third smallest~~ value in
4th smallest (median)

Partition 1 randomly selects 5 as pivot

7	2	9	4	1	8	5
---	---	---	---	---	---	---

2	4	1	5	7	9	8
---	---	---	---	---	---	---

↓
In this case we are done!



QuickSelect Algorithm

Goal: Find the k-th smallest value without sorting

Ex: Find the third smallest value in

7	2	9	4	1	8	5
---	---	---	---	---	---	---

Partition 1 randomly selects 5 as pivot

2	4	1	5	7	9	8
---	---	---	---	---	---	---

Partition 2 randomly selects 2 as pivot

1	2	4
---	---	---

QuickSelect Algorithm

Goal: Find the k-th smallest value without sorting

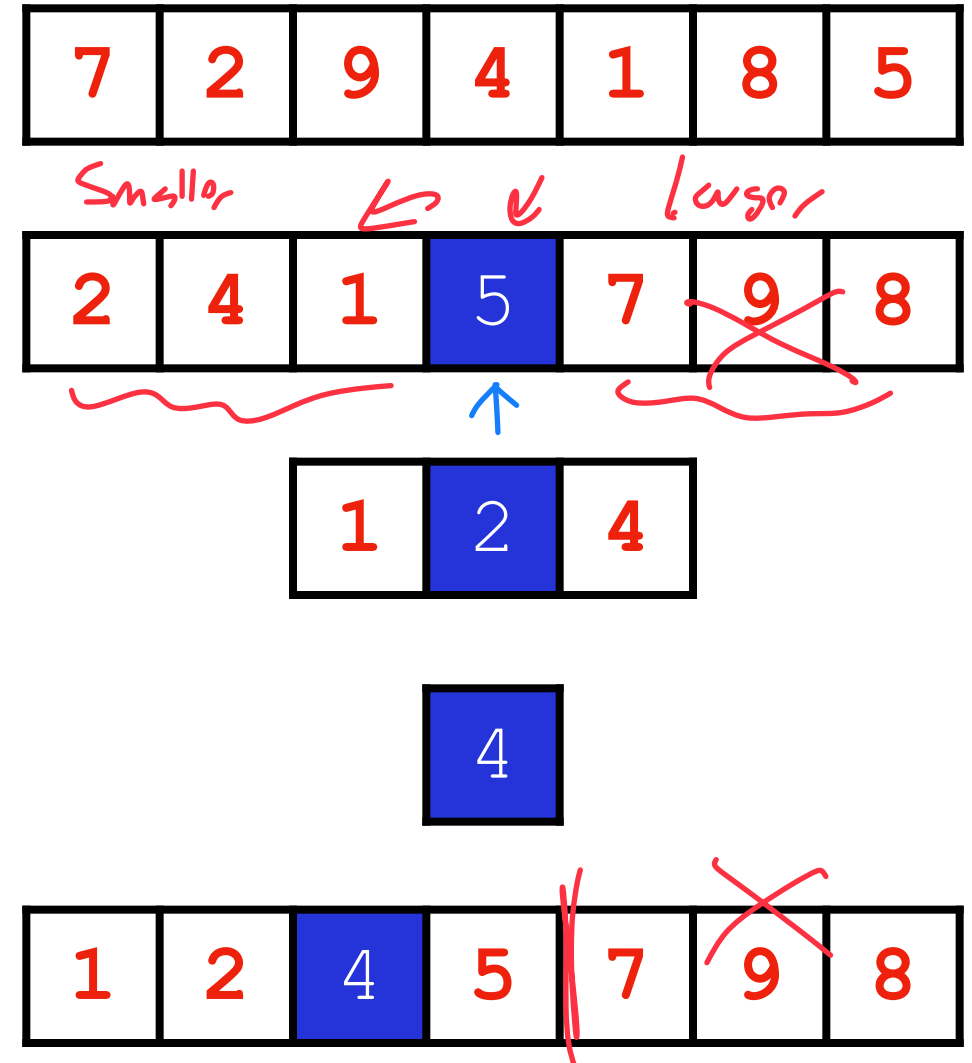
Ex: Find the third smallest value in

Partition 1 randomly selects 5 as pivot

Partition 2 randomly selects 2 as pivot

Partition 3 randomly "selects" 4 as pivot

Final Output: 3rd smallest value is 4!



QuickSelect Partition Algorithm

Goal: Partition items around pivot value as “smaller” or “larger”

1. Select a random pivot

Ex: $k = 3$

↑ “last item”

6	1	0	4	7	9	2	3
---	---	---	---	---	---	---	---

QuickSelect Partition Algorithm

Goal: Partition items around pivot value as “smaller” or “larger”

1. Select a random pivot

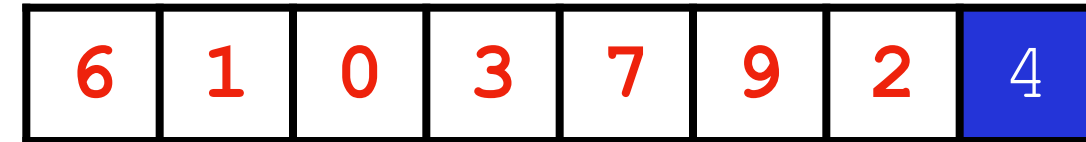
Ex: $k = 3$



A horizontal array of 8 cells containing the numbers 6, 1, 0, 4, 7, 9, 2, and 3. The number 4 is circled in red, indicating it is the selected pivot.

6	1	0	4	7	9	2	3
---	---	---	---	---	---	---	---

2. Swap pivot value to end of array



A horizontal array of 8 cells containing the numbers 6, 1, 0, 3, 7, 9, 2, and 4. The number 4 is now in the last cell and is highlighted with a blue background, indicating it has been swapped to the end of the array.

6	1	0	3	7	9	2	4
---	---	---	---	---	---	---	---

QuickSelect Partition Algorithm

Goal: Partition items around pivot value as “smaller” or “larger”

1. Select a random pivot

Ex: $k = 3$

6	1	0	4	7	9	2	3
---	---	---	---	---	---	---	---

2. Swap pivot value to end of array

6	1	0	3	7	9	2	4
---	---	---	---	---	---	---	---

3. Swap remaining items around pivot

1	0	3	2	7	9	6	4
---	---	---	---	---	---	---	---

Small = 4 larger

Track number smaller (small = 4)

↪ In detail in a slide or two

QuickSelect Partition Algorithm

Goal: Partition items around pivot value as “smaller” or “larger”

1. Select a random pivot

Ex: $k = 3$

6	1	0	4	7	9	2	3
---	---	---	---	---	---	---	---

2. Swap pivot value to end of array

6	1	0	3	7	9	2	4
---	---	---	---	---	---	---	---

3. Swap remaining items around pivot

Track number smaller (small = 4)

tricky step

1	0	3	2	7	9	6	4
---	---	---	---	---	---	---	---

4. Swap pivot value to its proper place

1	0	3	2	4	9	6	7
---	---	---	---	---	---	---	---

```

1 swap(arr[k], arr[length-1]);
2
3 for (int i = 0; i < length-1; i++){
4     if (arr[i] < arr[length-1]){
5         swap(arr[i], arr[small]);
6         small++;
7     }
8 }
9 swap(arr[small], arr[length-1]);

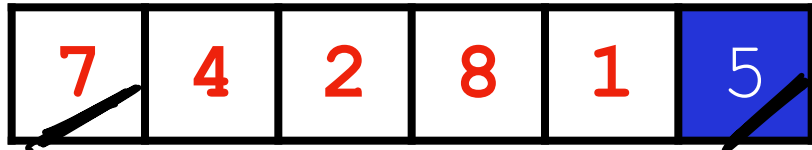
```

Swap pivot w/ end

(pivot)

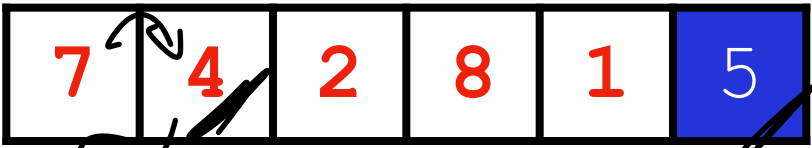
Swap pivot w/ first larger item

1D Partition Example with pivot $k = 2$ on $[7, 4, 5, 8, 1, 2]$



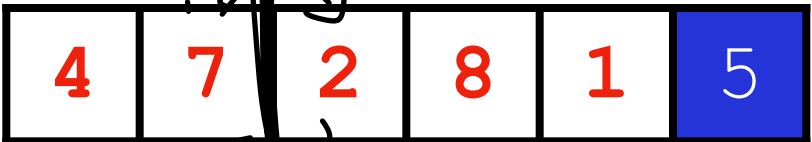
Index = 0, Small = 0 $7 > 5$

Swap it w/ $small = 0$

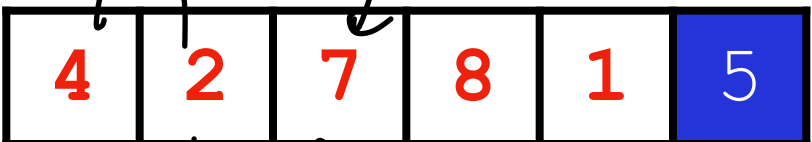


Index = 1, Small = 0 $4 < 5$

Small++



Index = 2, Small = 1 $2 < 5$



Index = 3, Small = 2

Join Code: 225



```
1 swap(arr[k], arr[length-1]);
2
3 for (int i = 0; i < length-1; i++){
4     if (arr[i] < arr[length-1]){
5         swap(arr[i], arr[small]);
6         small++;
7     }
8 }
9 swap(arr[small], arr[length-1]);
```

1D Partition Example with pivot $k = 2$ on $[7, 4, 5, 8, 1, 2]$

Input Array	Input Index	Input Small
7, 4, 2, 8, 1, 5	0	0
7, 4, 2, 8, 1, 5	1	0
4, 7, 2, 8, 1, 5	2	1
4, 2, 7, 8, 1, 5	3	2
	4	
	5 (Not run)	



```
1 swap(arr[k], arr[length-1]);
2
3 for (int i = 0; i < length-1; i++){
4     if (arr[i] < arr[length-1]){
5         swap(arr[i], arr[small]);
6         small++;
7     }
8 }
9 swap(arr[small], arr[length-1]);
```

1D Partition Example with pivot $k = 2$ on $[7, 4, 5, 8, 1, 2]$

Input Array	Input Index	Input Small
7, 4, 2, 8, 1, 5	0	0
7, 4, 2, 8, 1, 5	1	0
4, 7, 2, 8, 1, 5	2	1
4, 2, 7, 8, 1, 5	3	2
4, 2, 7, 8, 1, 5	4	2
4, 2, 1, 8, 7, 5	5 (Not run)	3 (Final)

Final Output:

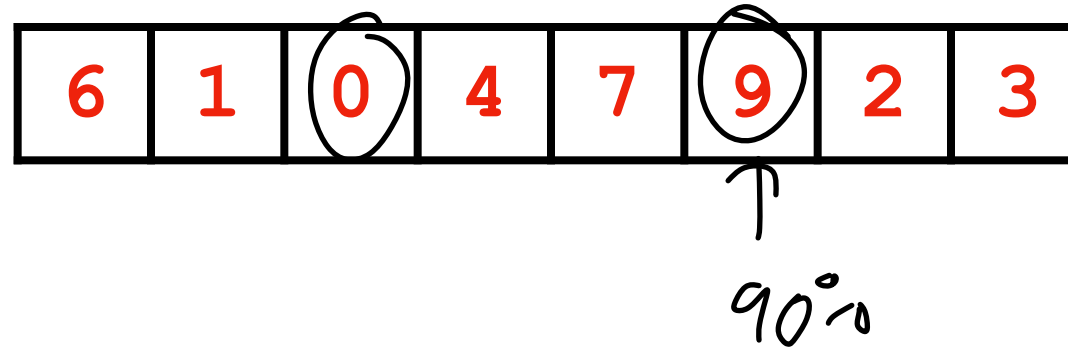
4	2	1	5	7	8
---	---	---	---	---	---

QuickSelect Big O

Join Code: 225



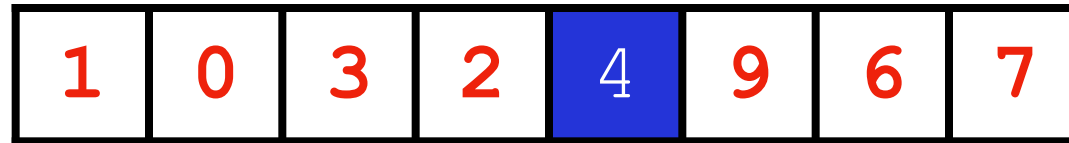
What is the worst case partition **value** for the following array?



Smallest or largest value partitions space uselessly

QuickSelect Big O

Why is the actual median value the best case partition value?



QuickSelect Big O

We are not sorting the array so it should be faster than $O(n \log n)$!

Average Case:

Big O:



QuickSelect Big O

We are not sorting the array so it should be faster than $O(n \log n)$!

Average Case: $O(N)$

First partition always $O(N)$

Random pivot **expectation:**

Given k as pivot index

$$\frac{1}{n} \sum_{k=0}^{n-1} \mathbb{E}[\max(k, n - k - 1)]$$

Unproven: This is also $O(N)$

Big O: $O(N^2)$

Ex: Find 1st smallest value

1	2	3	4	5
---	---	---	---	---

N

1	2	3	4
---	---	---	---

$N-1$

1	2	3
---	---	---

$N-2$

1	2
---	---

$N-3$

1

$\dots$

$N + (N-1) + \dots$

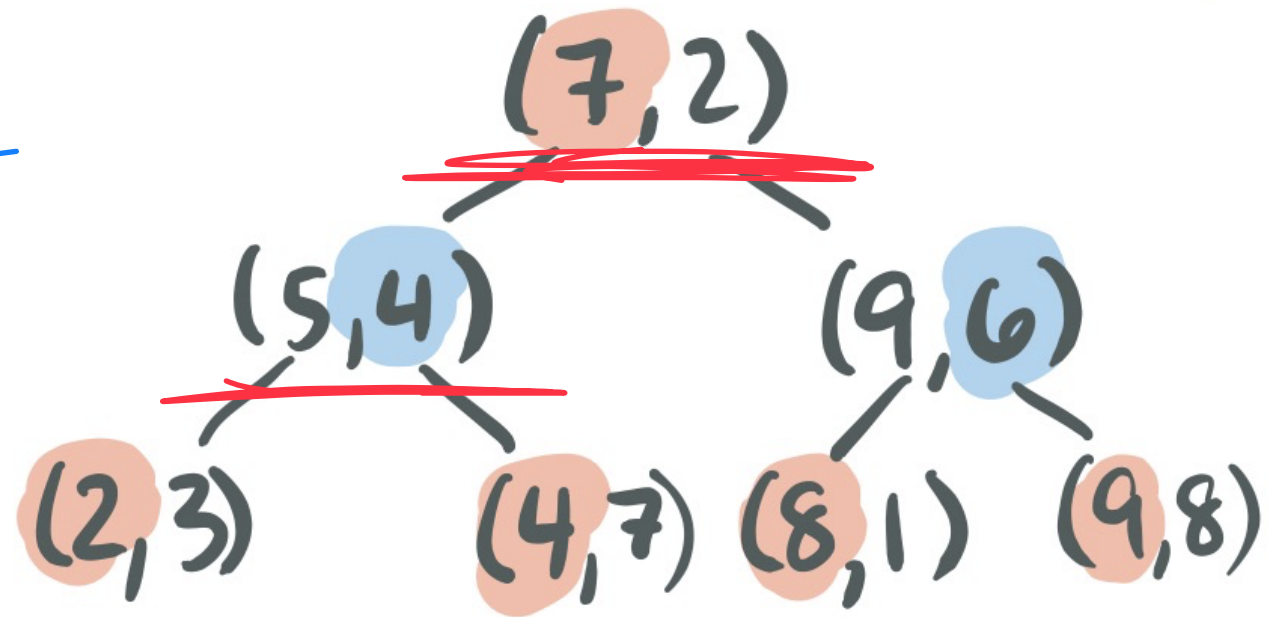
$= \frac{N(N+1)}{2}$

$O(N^2)$

QuickSelect in a KD Tree

How do we use a 1D algorithm in a k-dimension tree constructor?

Due to lost time
all of this on Monday
or bonus video



Hint: **Lambda functions!**

Functions as arguments

Consider the function from Excel
`COUNTIF(range, criteria)`

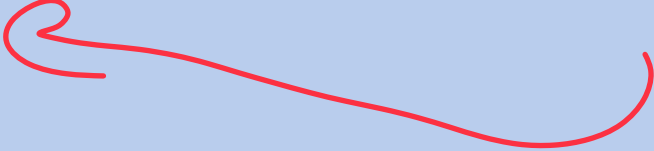
A10 ✕ ✓ fx =COUNTIF(A1:A9,"<0")			
	A	B	C
1	1		
2	102		
3	105		
4	4		
5	5		
6	27		
7	41		
8	-7		
9	999		
10	1		
11			

Functions as arguments

Countif.hpp

```
10 template <typename Iter, typename Pred>
11 int Countif(Iter begin, Iter end, Pred pred) {
12     int count = 0;
13     auto cur = begin;
14
15     while(cur != end) {
16         if(pred(*cur))
17             ++count;
18         ++cur;
19     }
20
21     return count;
22 }
```

Args which are functions



Lambda Functions in C++

Here are several ways to write a function as an object

main.cpp

```
1 bool isNegative(int num) { return (num < 0); }
2
3 class IsNegative {
4 public:
5     bool operator() (int num) { return (num < 0); }
6 };
7
8 int main() {
9     std::vector<int> numbers = {1, 102, 105, 4, 5, 27, 41, -7, 999};
10
11     auto isnegl = [](int num) { return (num < 0); };
12     auto isnegfp = isNegative;
13     auto isnegfunctor = IsNegative();
14
15     cout << "There are " << Countif(numbers.begin(), numbers.end(), _____)
16         << " negative numbers" << std::endl;
17 }
```

multiple ways to pass a function

Lambda Functions in C++

An inline function without a name

[Capture](Arg List){ Function Body }

Lambda Functions in C++

An inline function without a name

[Capture](Arg List){ Function Body}

Capture: Takes the value of object based on when the lambda was defined, NOT the current value of the object!

Arg List: Standard way of inputting into a function

Function Body: Code can use both capture vars and arg vars

Lambda Functions in C++



main.cpp

```
29  int big;  
30  std::cout << "How big is big? ";  
31  std::cin >> big;  
32  
33  auto isbig = [big](int num) { return (num >= big); };  
34  
35  
36  
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)  
38  << " big numbers" << std::endl;  
}
```

Lambda Functions in C++



main.cpp

```
29  int big;  
30  std::cout << "How big is big? ";  
31  std::cin >> big;  
32  
33  auto isbig = [big](int num) { return (num >= big); };  
34  
35  
36  
37  std::cout << "There are " << Countif(numbers.begin(), numbers.end(), isbig)  
38  << " big numbers" << std::endl;  
}
```

Useful for mp_mosaics!

KD-Tree will split points in one dimension

When comparing, we need to remember what dimension we are in!

Final Tips

The mp_mosaic writeup / documentation is long. **READ IT**

The suggestions in the writeup should be followed carefully

```
1  template <typename RandIter, typename Comparator>
2  void select(RandIter start, RandIter end, RandIter k, Comparator cmp)
3  {
4      /**
5       * @todo Implement this function!
6       */
7
8  }
```

Understanding 'randIter'

An iterator is a container giving access in different ways:

Forward

Bidirectional

Random Access

Implementing quickselect with RandIter

Random Access Iterator lets you:

Swap items using `std::swap()`

```
1  template <typename RandIter, typename Comparator>
2  void BlackBox(RandIter A, RandIter B)
3  {
4
5      std::swap(*A, *B) ;
6
7
8  }
9
```

Hint: We just covered when you might want to swap in quick select!

Implementing quickselect with RandIter

Random Access Iterator lets you:

Access container indices using math operations

```
randIter A;
```

```
auto nth = *(A + n);
```

Get distance between two iterators

```
randIter A, B;
```

```
A < B;           // True if A is earlier in container than B
```

```
A - B;           // The distance between A and B
```

Implementing quickselect with RandIter

Random Access Iterator lets you:

Do most things you'd expect an array to be able to do!

The power of the **Interface!**

https://en.cppreference.com/w/cpp/iterator/random_access_iterator