CS2040S Tutorial 5

Tutorial Time

Slides for tutorials are taken and adapted from Christian

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

Trade-off: time complexity, space complexity

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

Time complexity for insert, delete, and find a word with length L to a collection of N words:

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

Time complexity for insert, delete, and find a word with length L to a collection of N words:

- AVL Tree: O(L log N)
- Trie: O(L)

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

Time complexity for insert, delete, and find a word with length L to a collection of N words:

- AVL Tree: O(L log N)
- Trie: O(L)

Space complexity:

Problem 1: AVL vs Trie

Discuss the trade-offs of using AVL and trie to store strings.

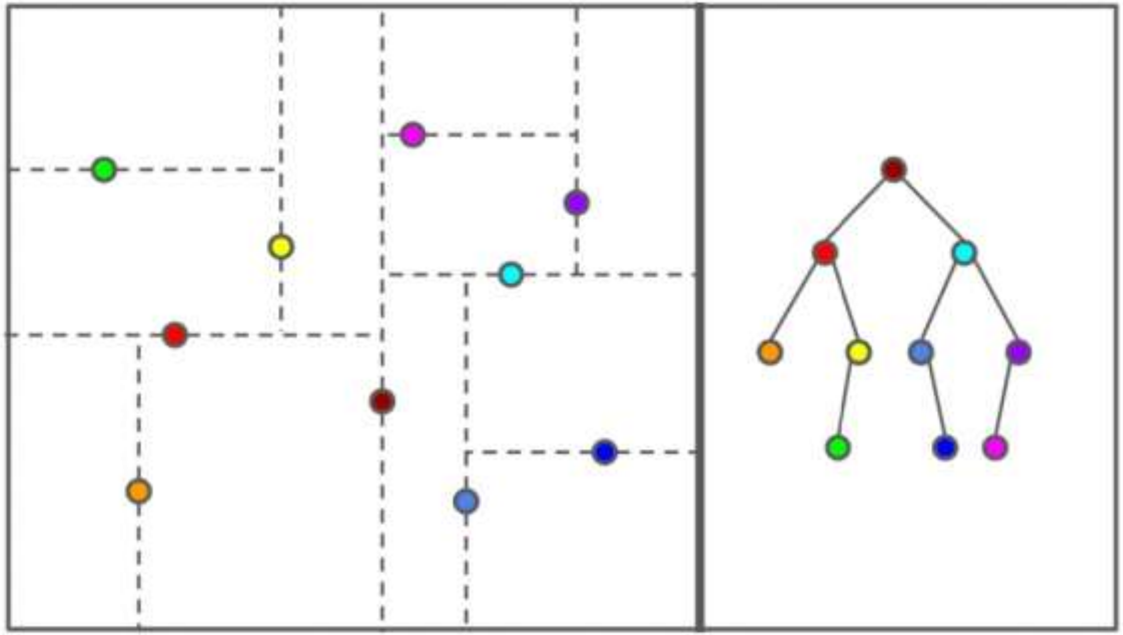
Time complexity for insert, delete, and find a word with length L to a collection of N words:

- AVL Tree: O(L log N)
- Trie: O(L)

Space complexity:

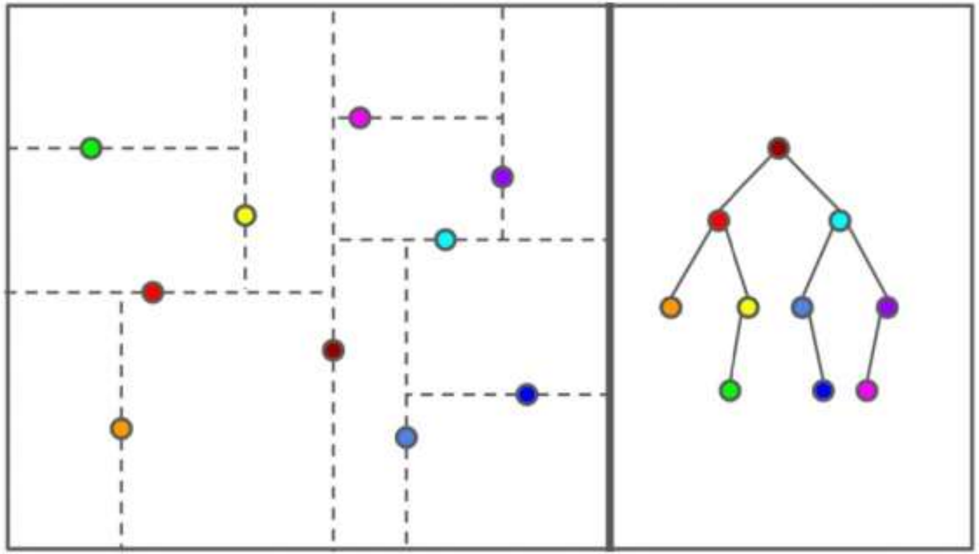
- AVL Tree: O(total_string_length)
- Trie: O(total_string_length)
- Trie tends to have more overhead cost (more nodes and edges)

Problem 2: kd-Trees

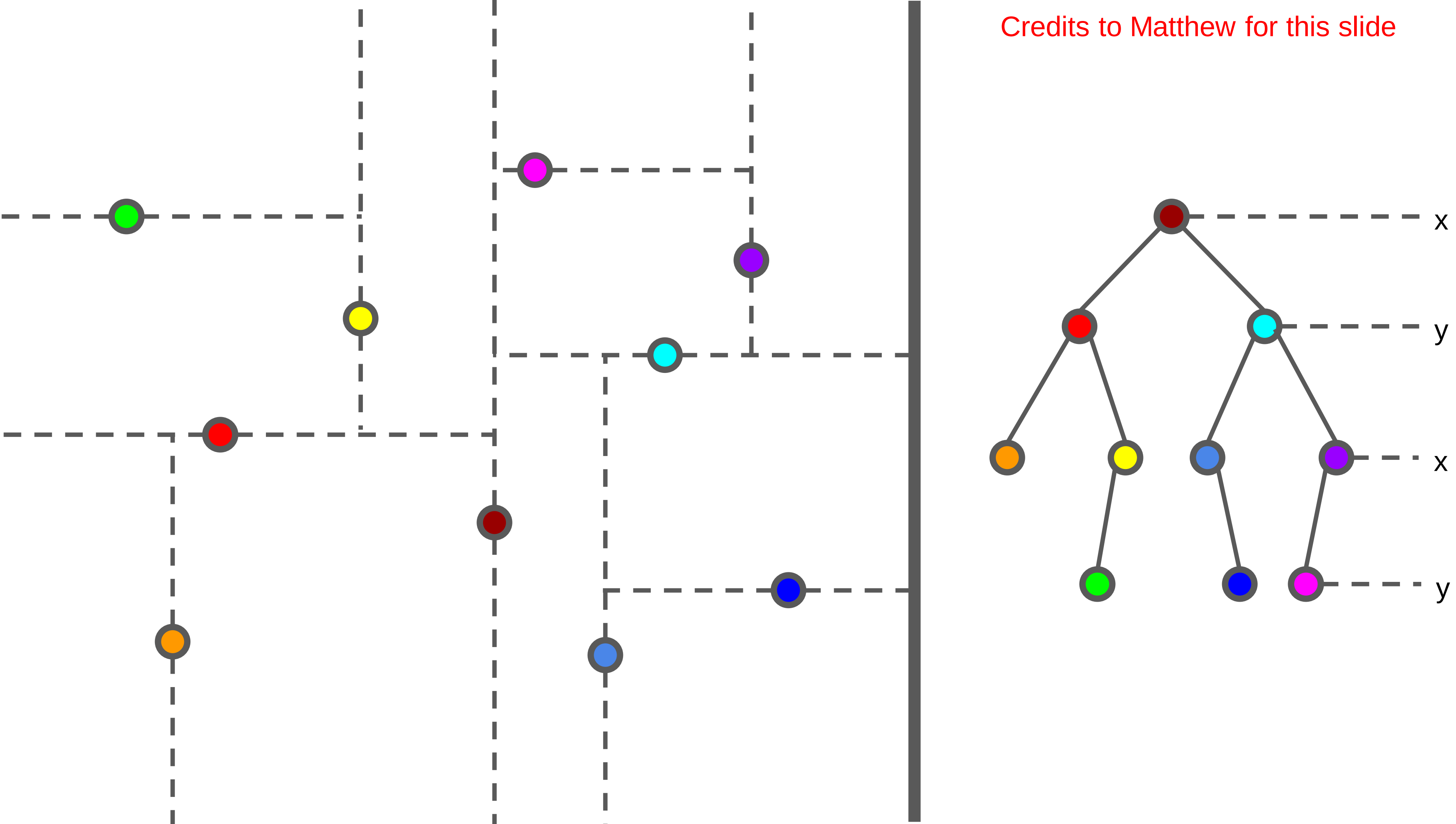


Problem 2a: kd-Trees

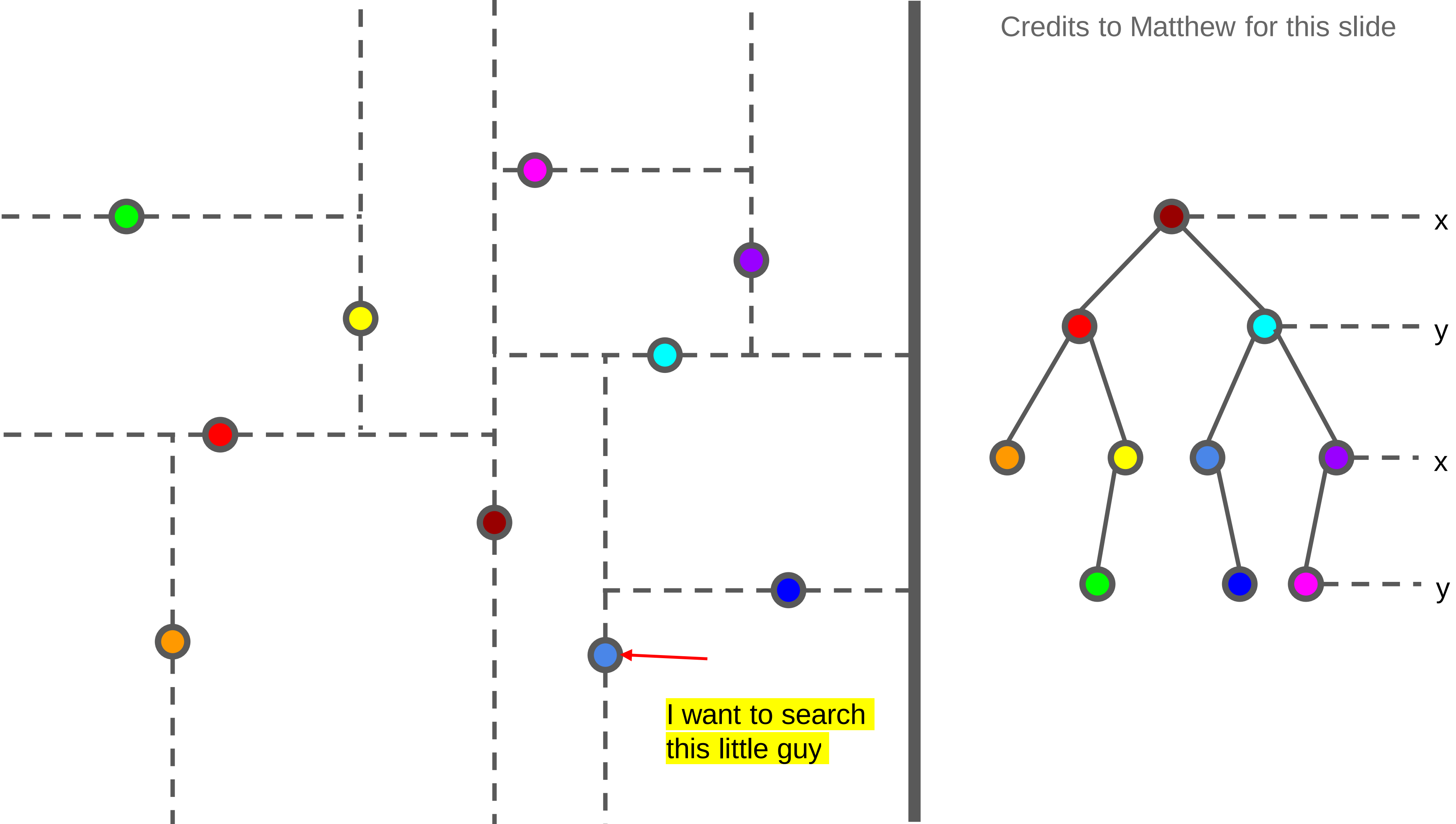
How do you search for a point in a kd-tree? What is the running time?

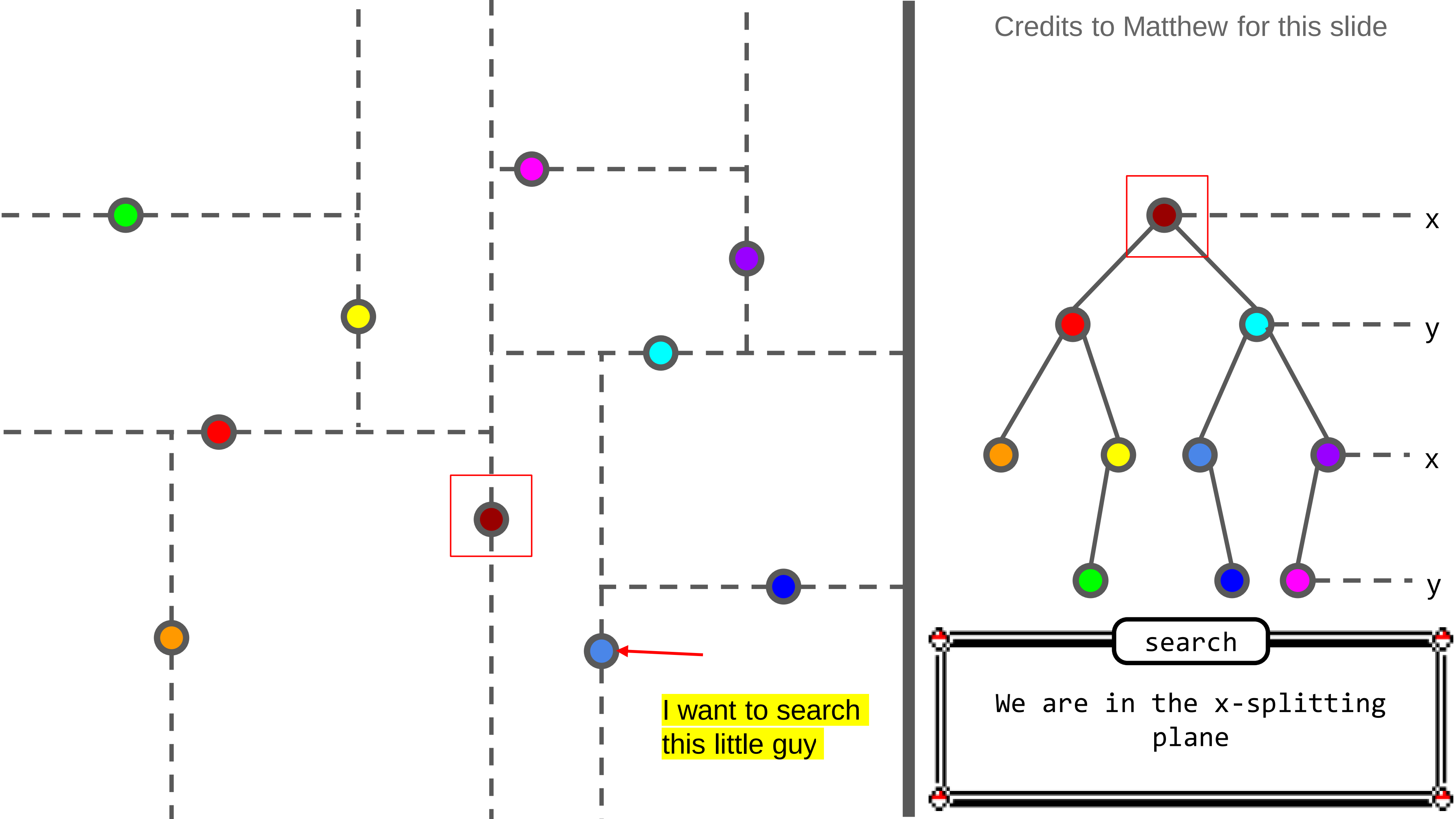


Credits to Matthew for this slide

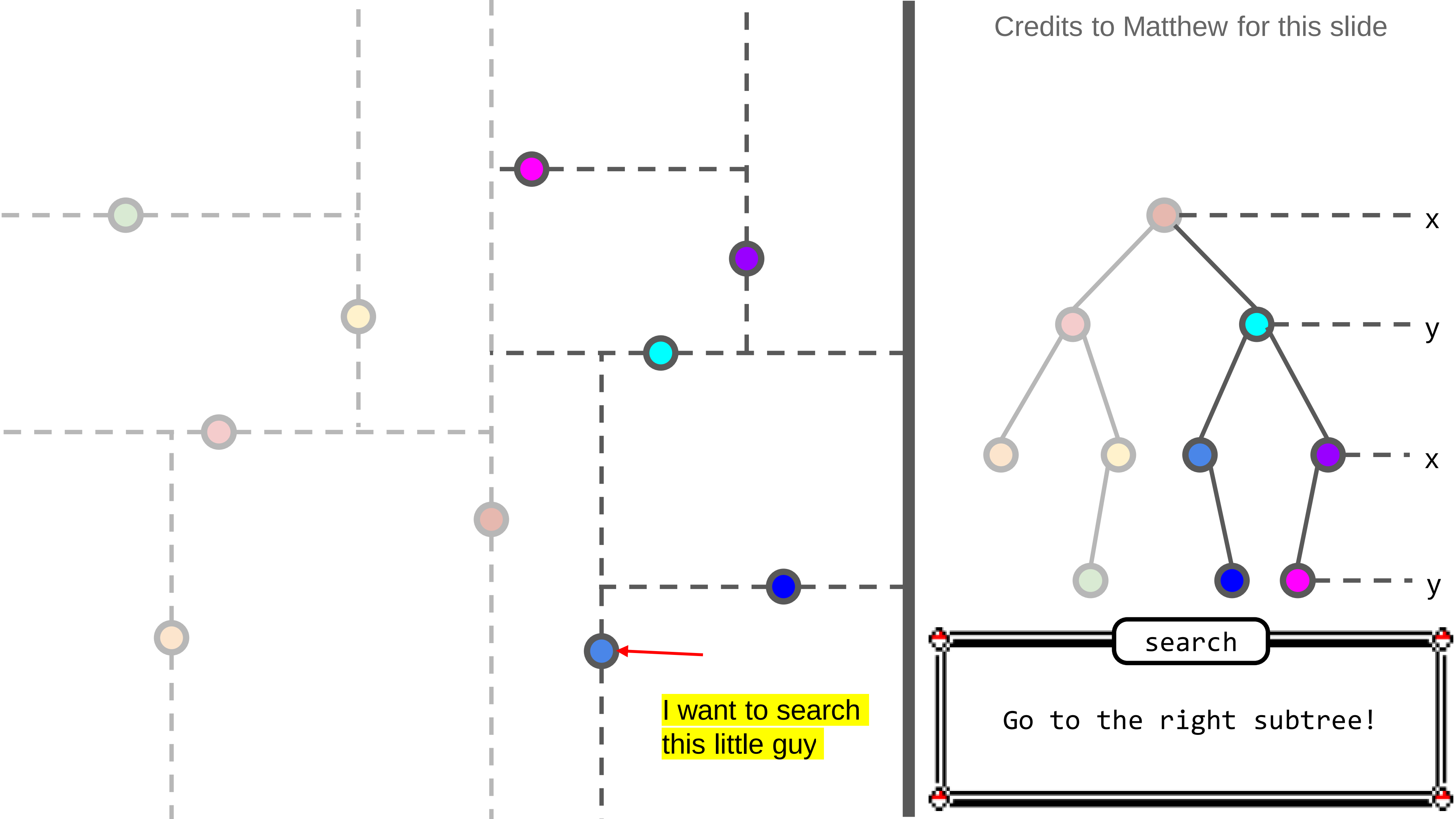


Credits to Matthew for this slide

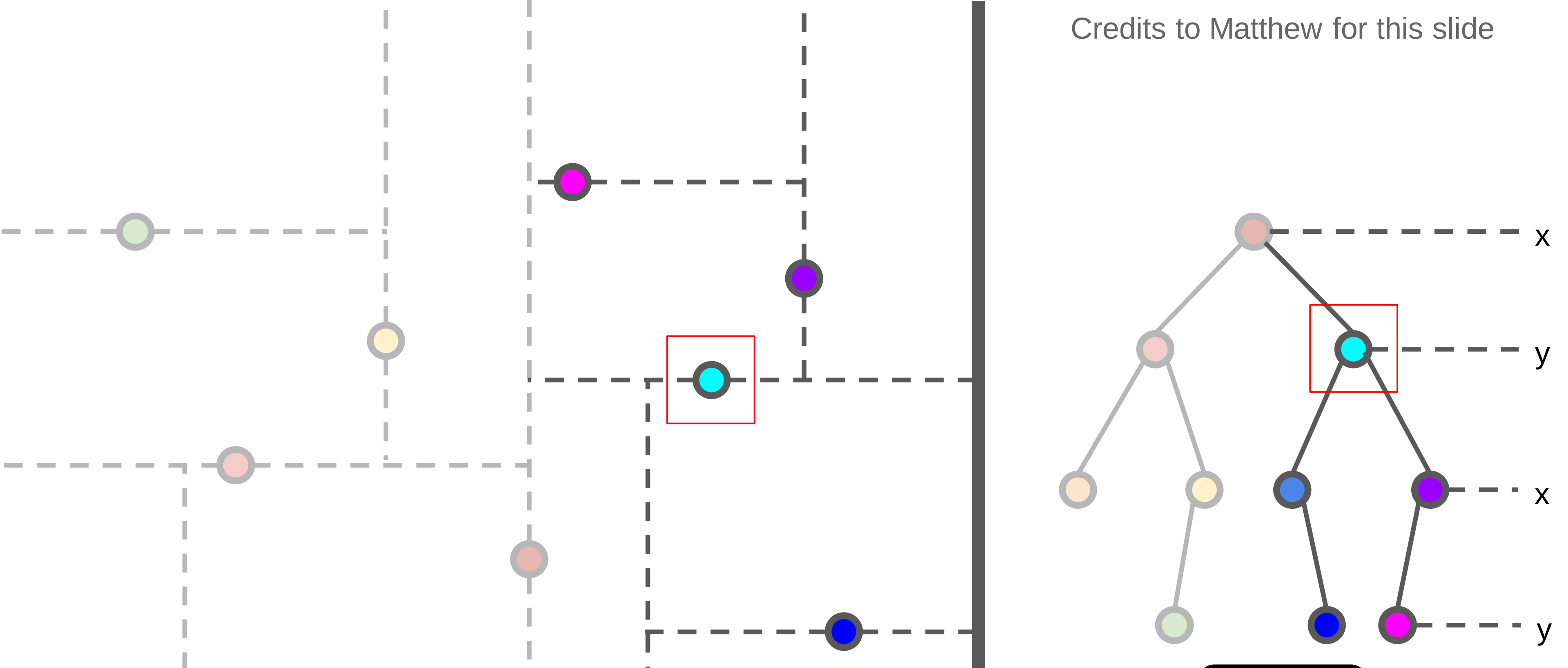


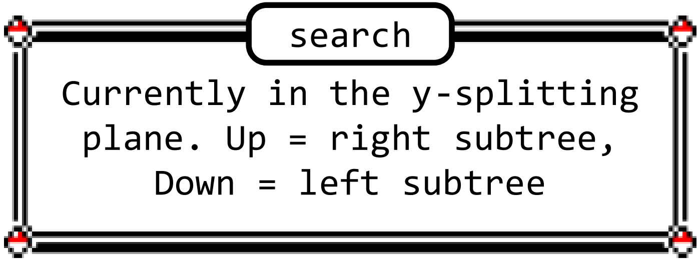

Credits to Matthew for this slide
x
y
x
y
search
We are in the x-splitting plane
I want to search this little guy

Credits to Matthew for this slide

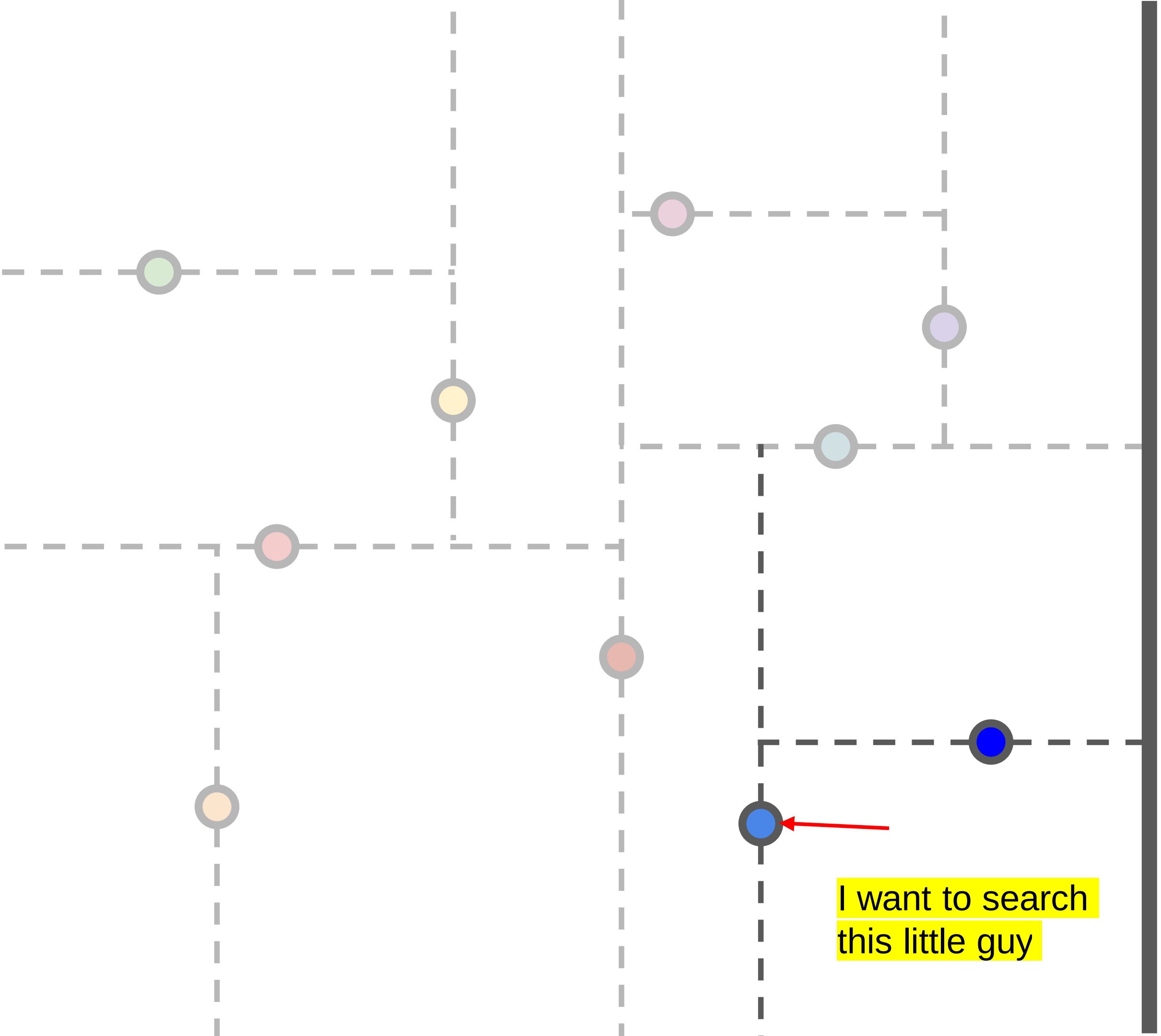


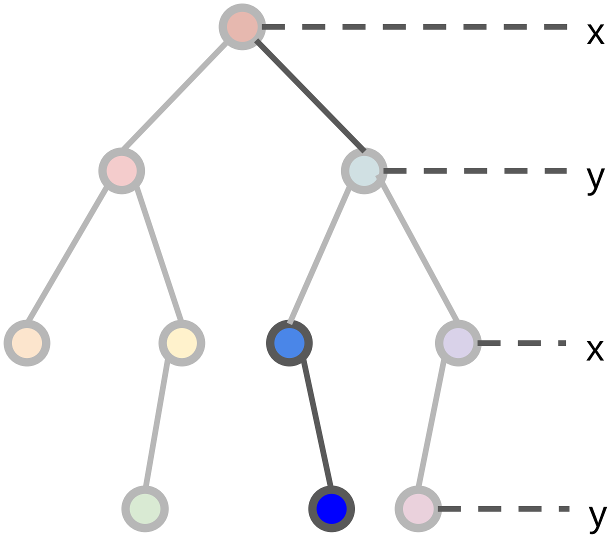
Credits to Matthew for this slide





Credits to Matthew for this slide

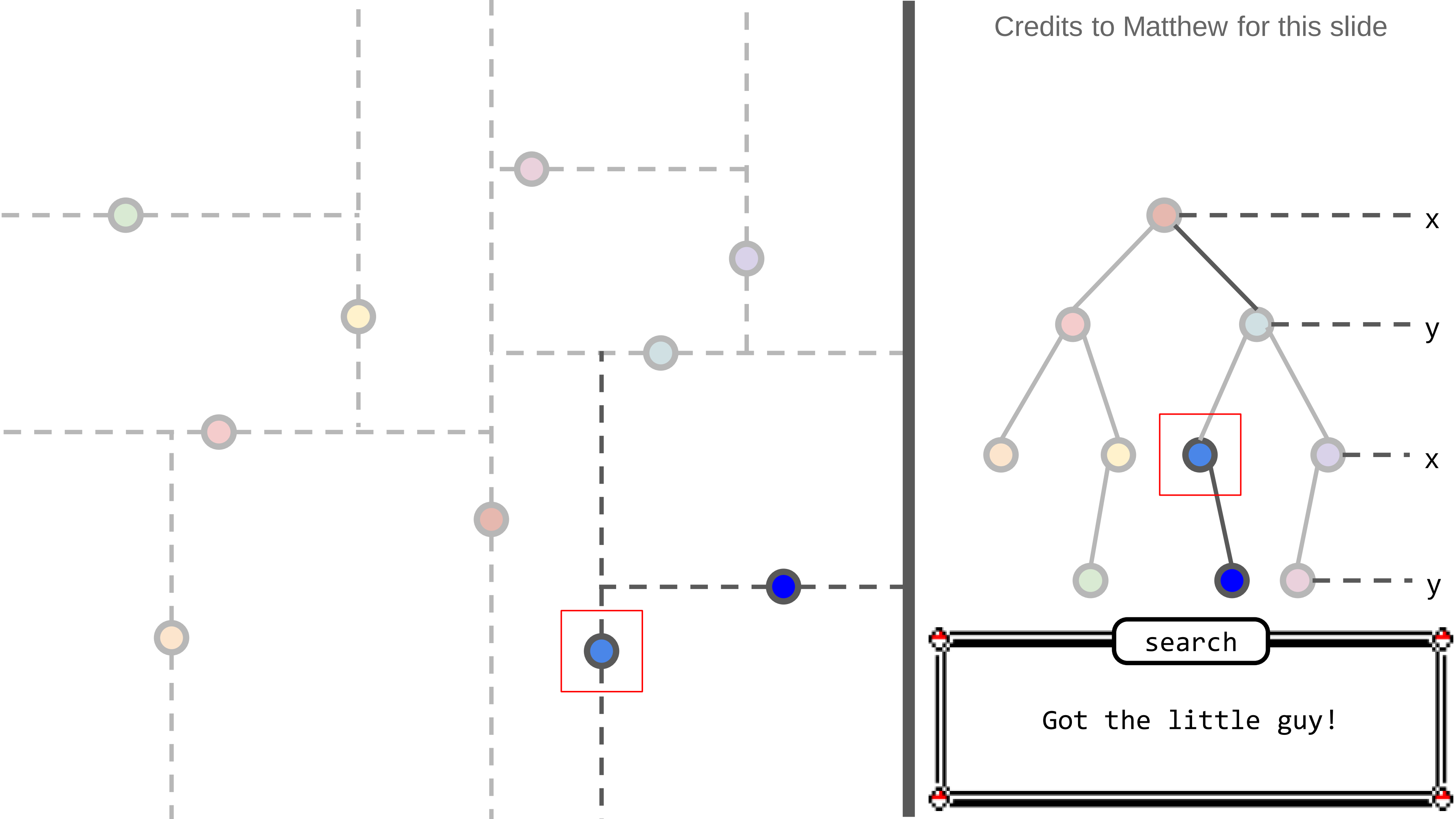




search

Go to the left subtree!

Credits to Matthew for this slide



Problem 2a: kd-Trees

What is the running time?

Problem 2a: kd-Trees

What is the running time?

- O(h), height of the tree

Note: kd-tree is not necessarily balanced. That’s the goal of 2b!

Problem 2b: kd-Trees

You are given an (unordered) array of points. What would be a good way to build a kd-tree? Think about what would keep the tree nicely balanced. What is the running time of the construction algorithm?

Problem 2b: kd-Trees

You are given an (unordered) array of points. What would be a good way to build a kd-tree? Think about what would keep the tree nicely balanced. What is the running time of the construction algorithm?

- Find median element in x as root for the first level. Then build left tree and right tree
- Find median element in y as root for the second level. Then build left tree and right tree.

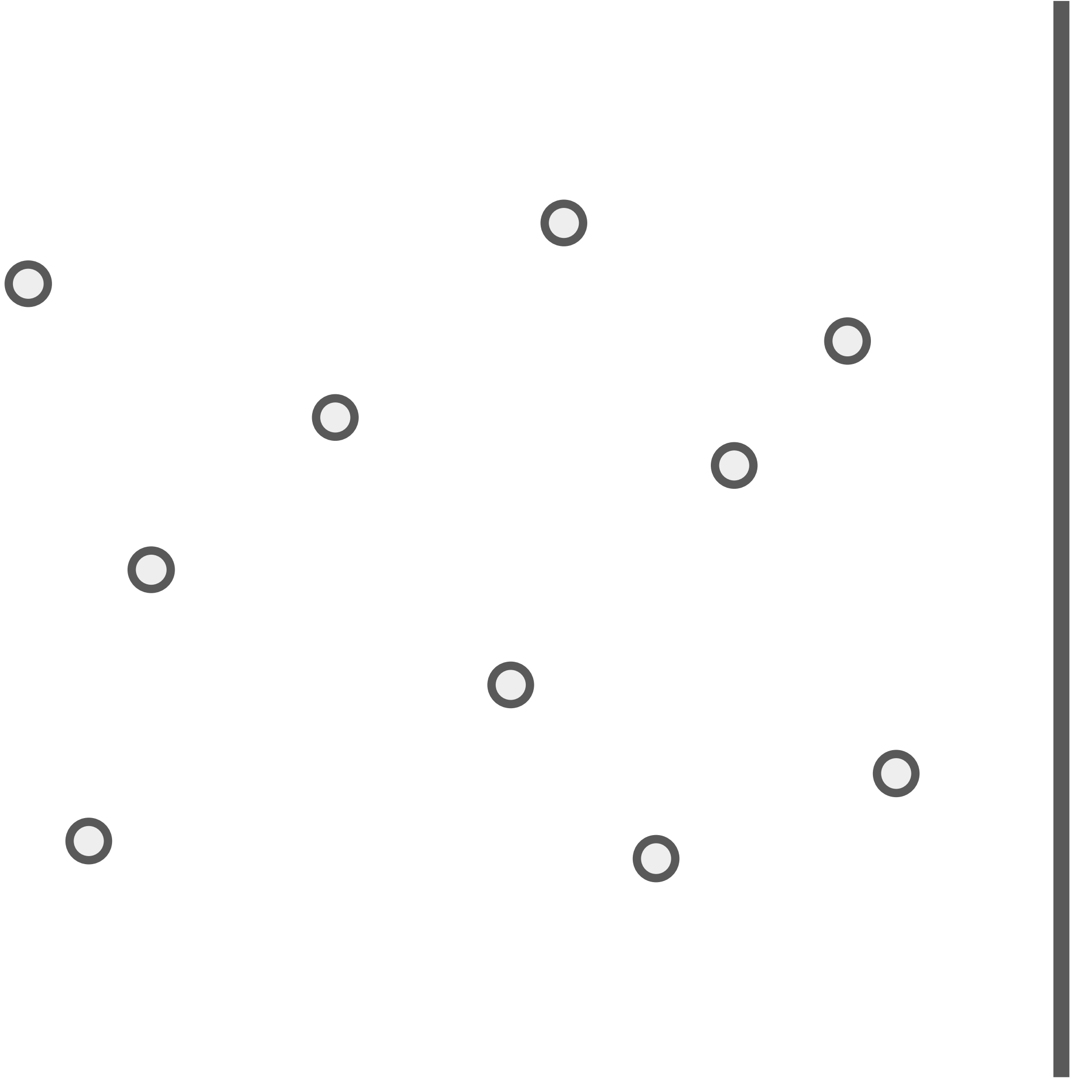
Problem 2b: kd-Trees

You are given an (unordered) array of points. What would be a good way to build a kd-tree? Think about what would keep the tree nicely balanced. What is the running time of the construction algorithm?

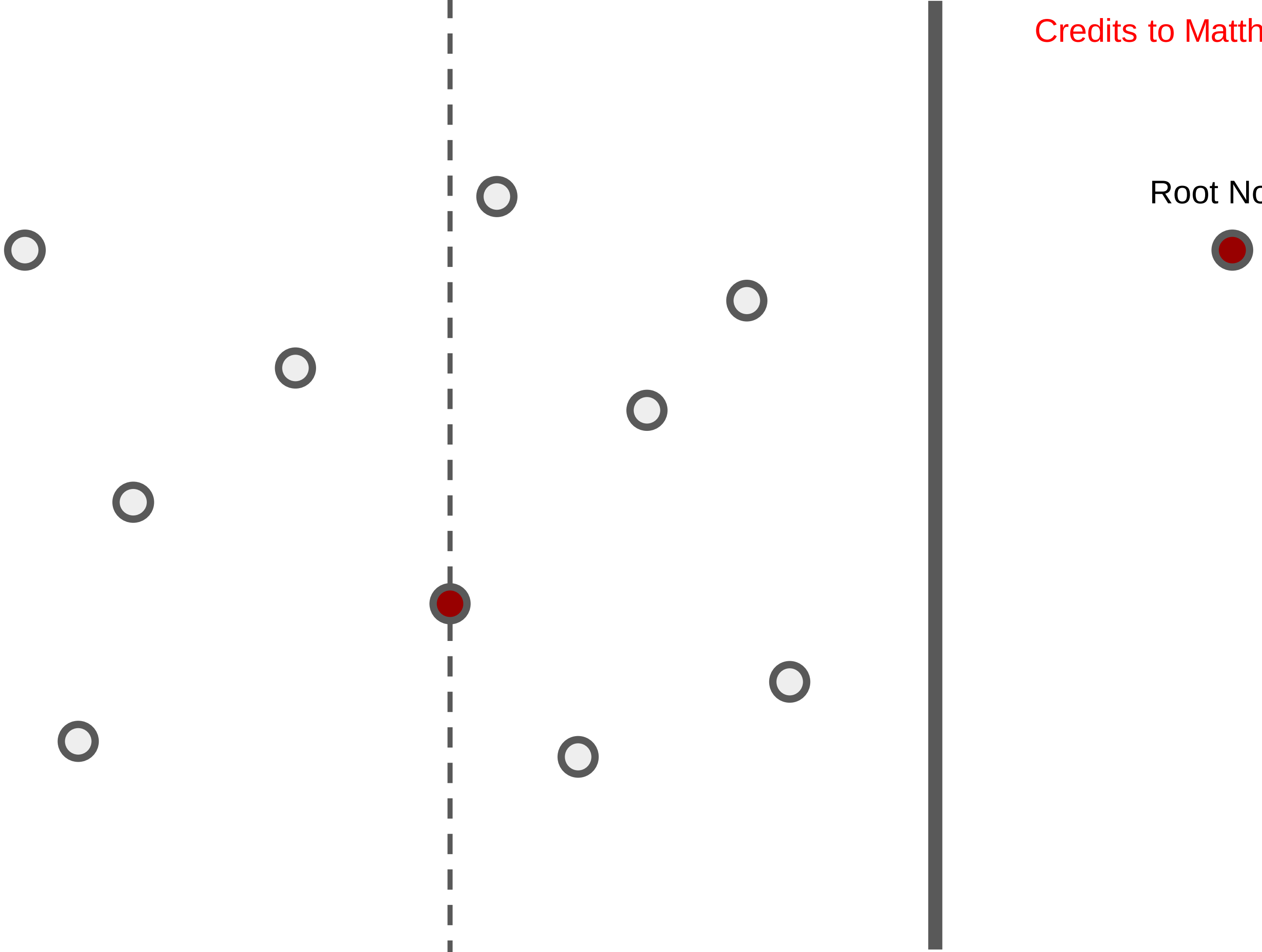
- Find median element in x as root for the first level. Then build left tree and right tree
- Find median element in y as root for the second level. Then build left tree and right tree.

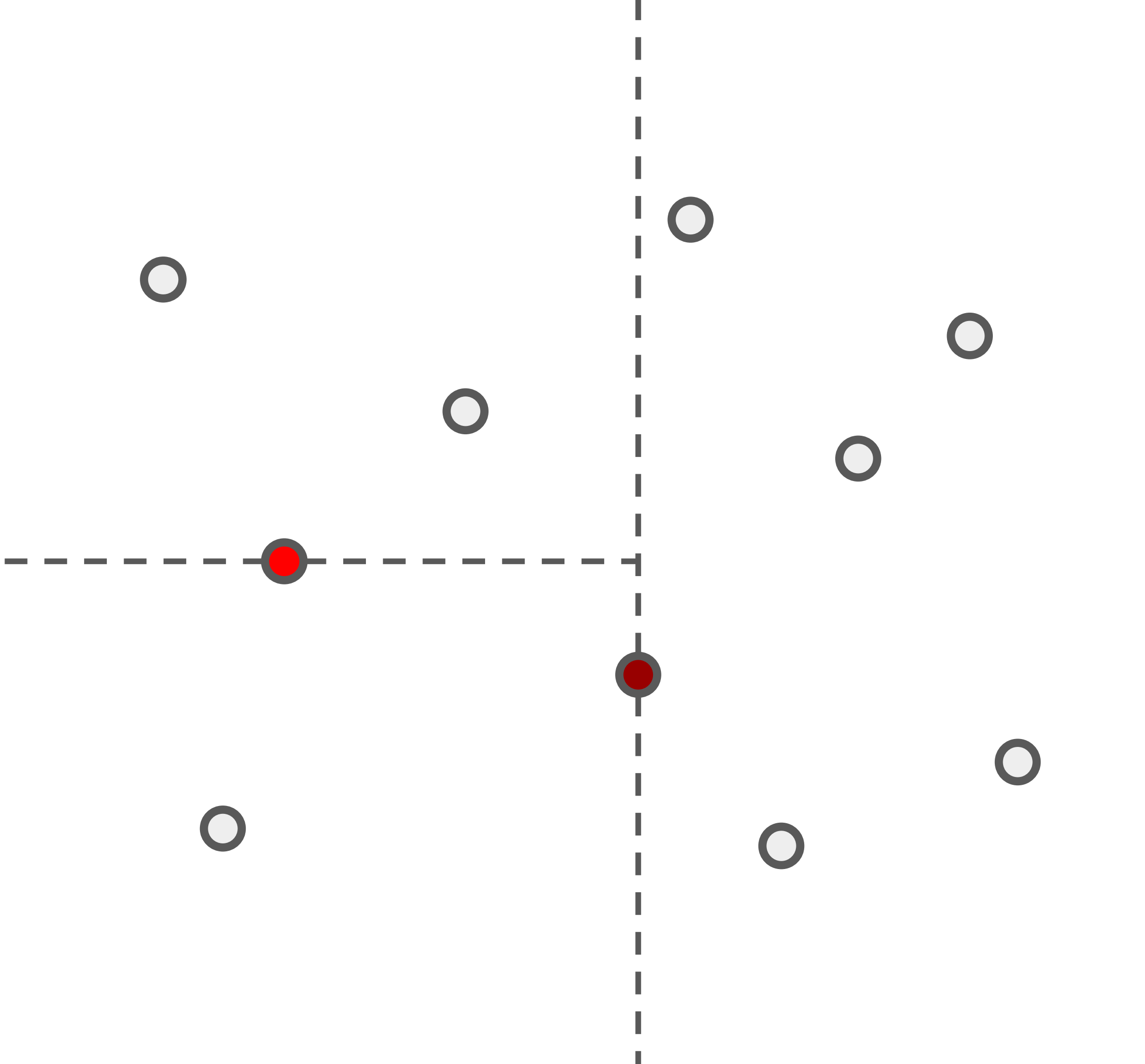
How fast can we find median? What are our options?

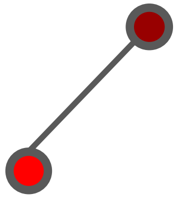
Credits to Matthew for this slide



Credits to Matthew for this slide





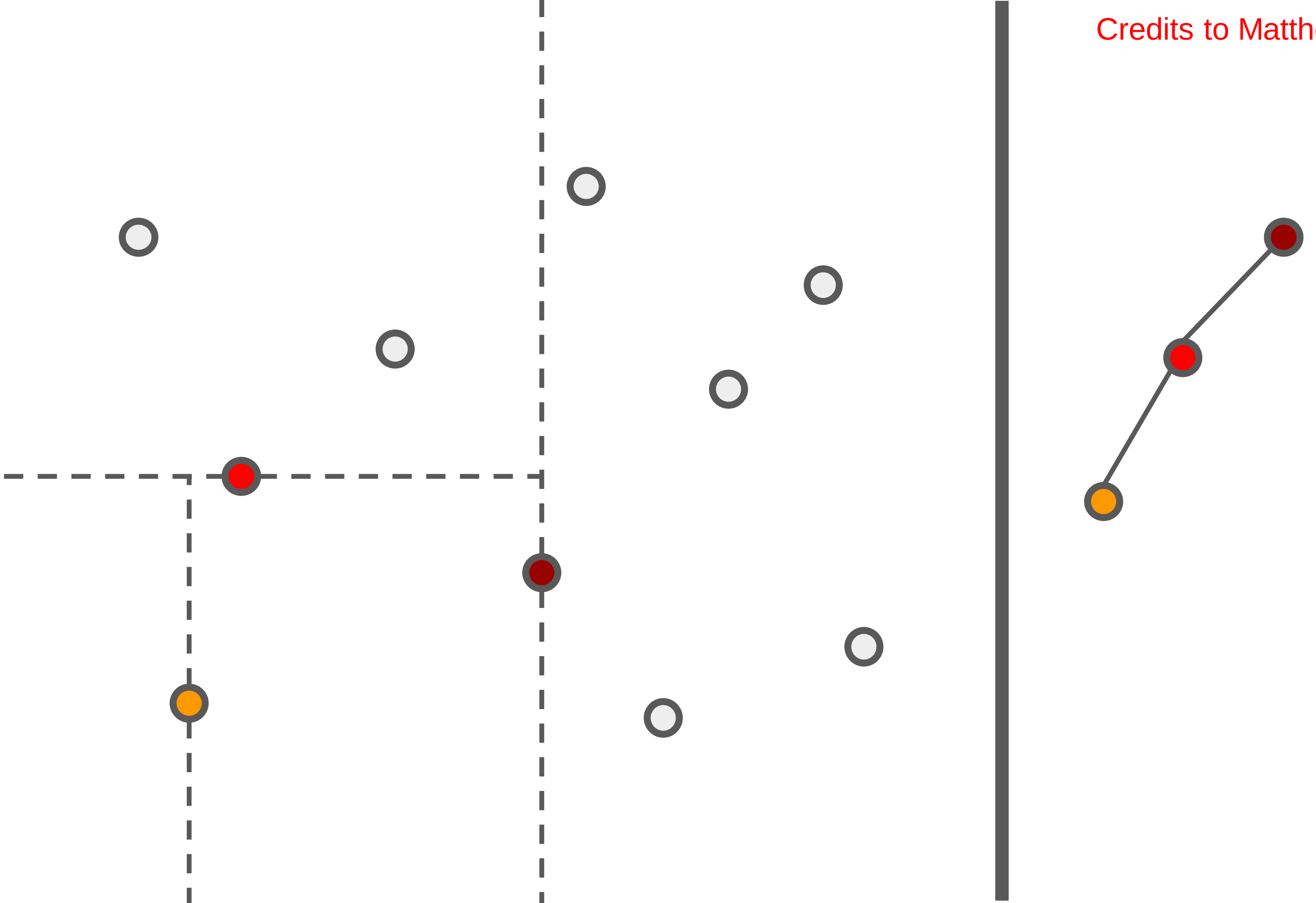


Convention:

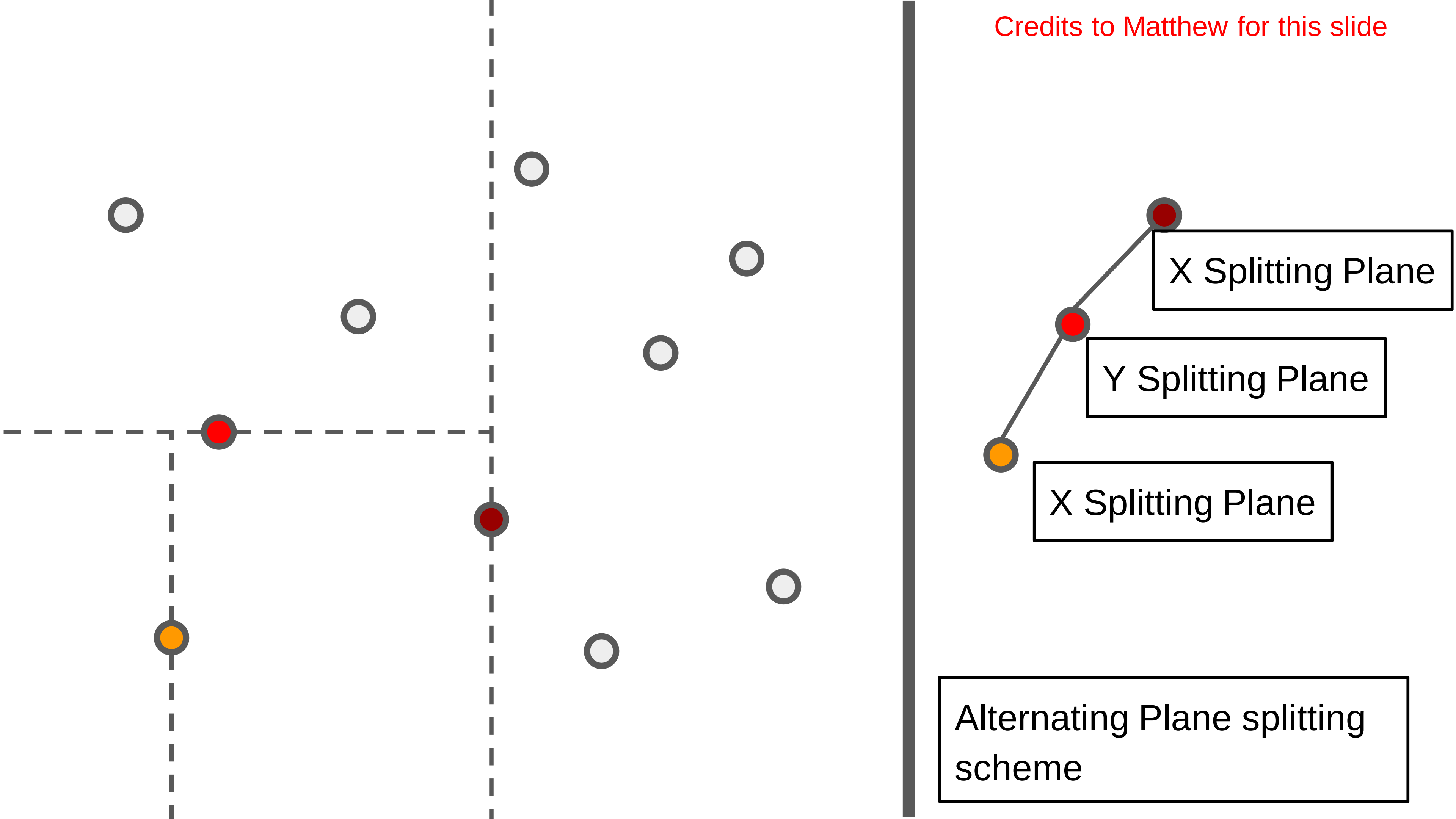
Left child node = smaller x (or y depending on the splitting plane) coordinate

Each level of the tree alternates between X and Y coordinates to split the remaining points

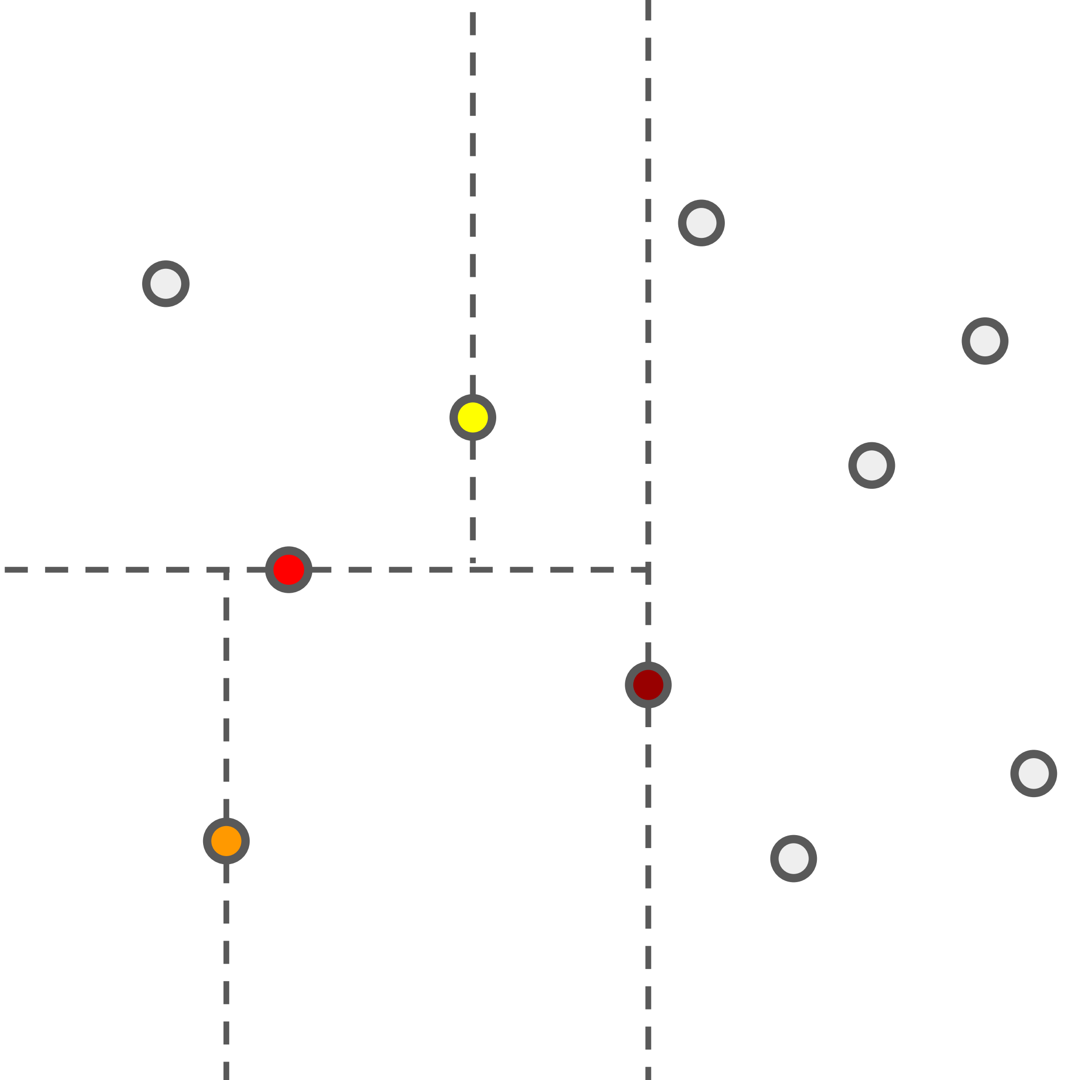
Credits to Matthew for this slide

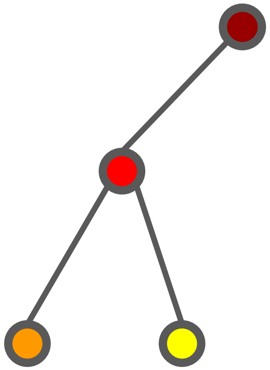


Credits to Matthew for this slide



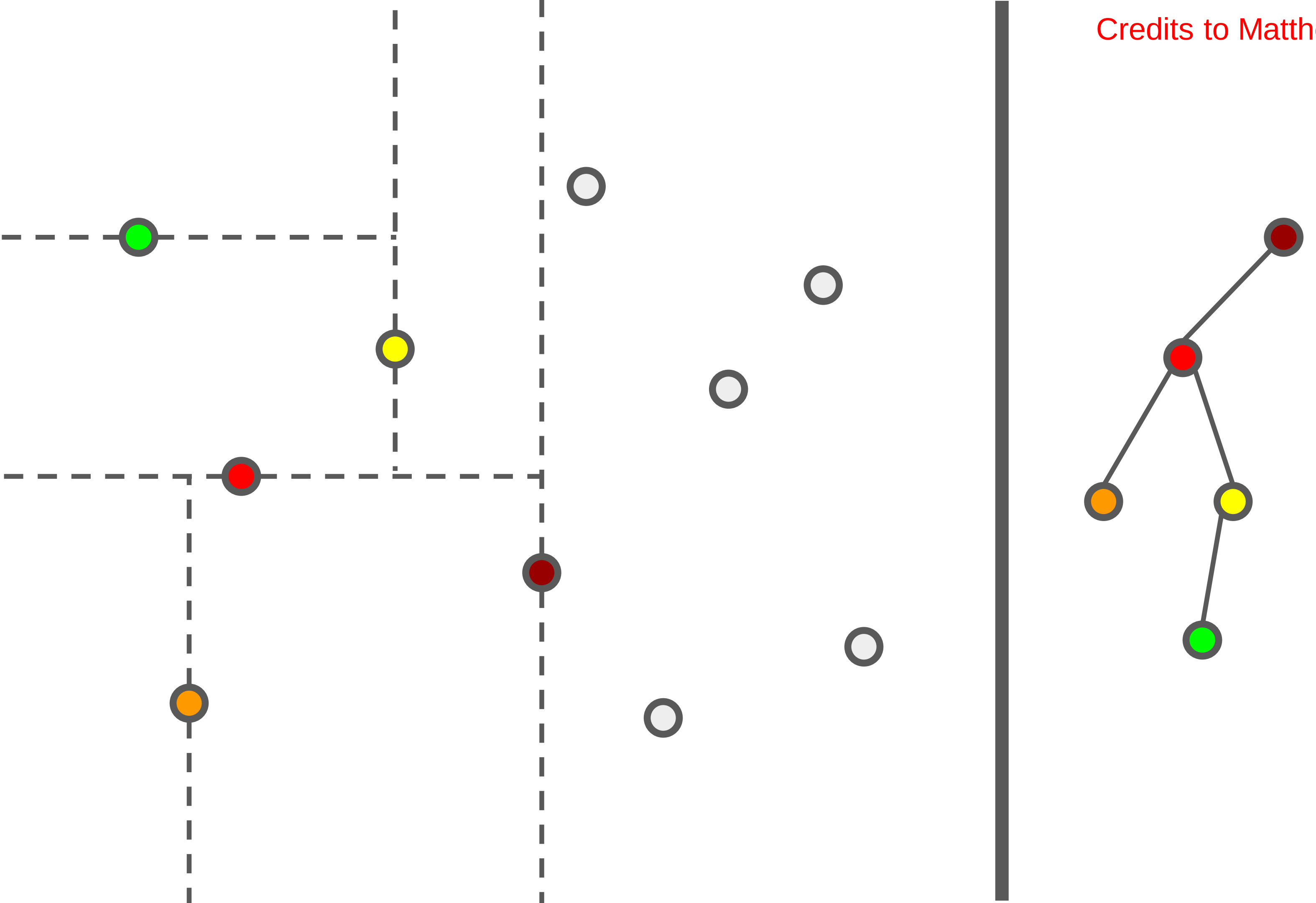
Alternating Plane splitting scheme

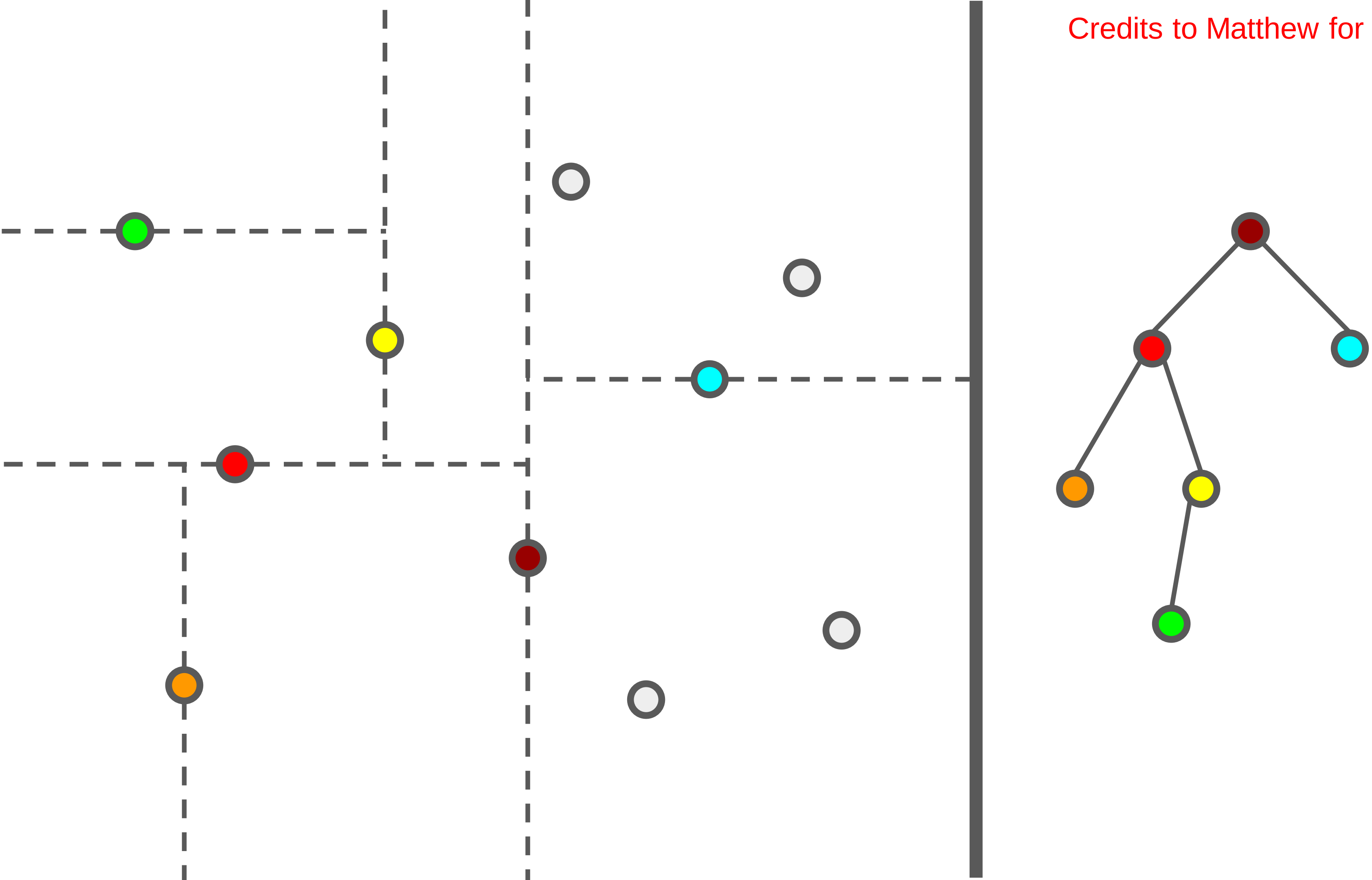




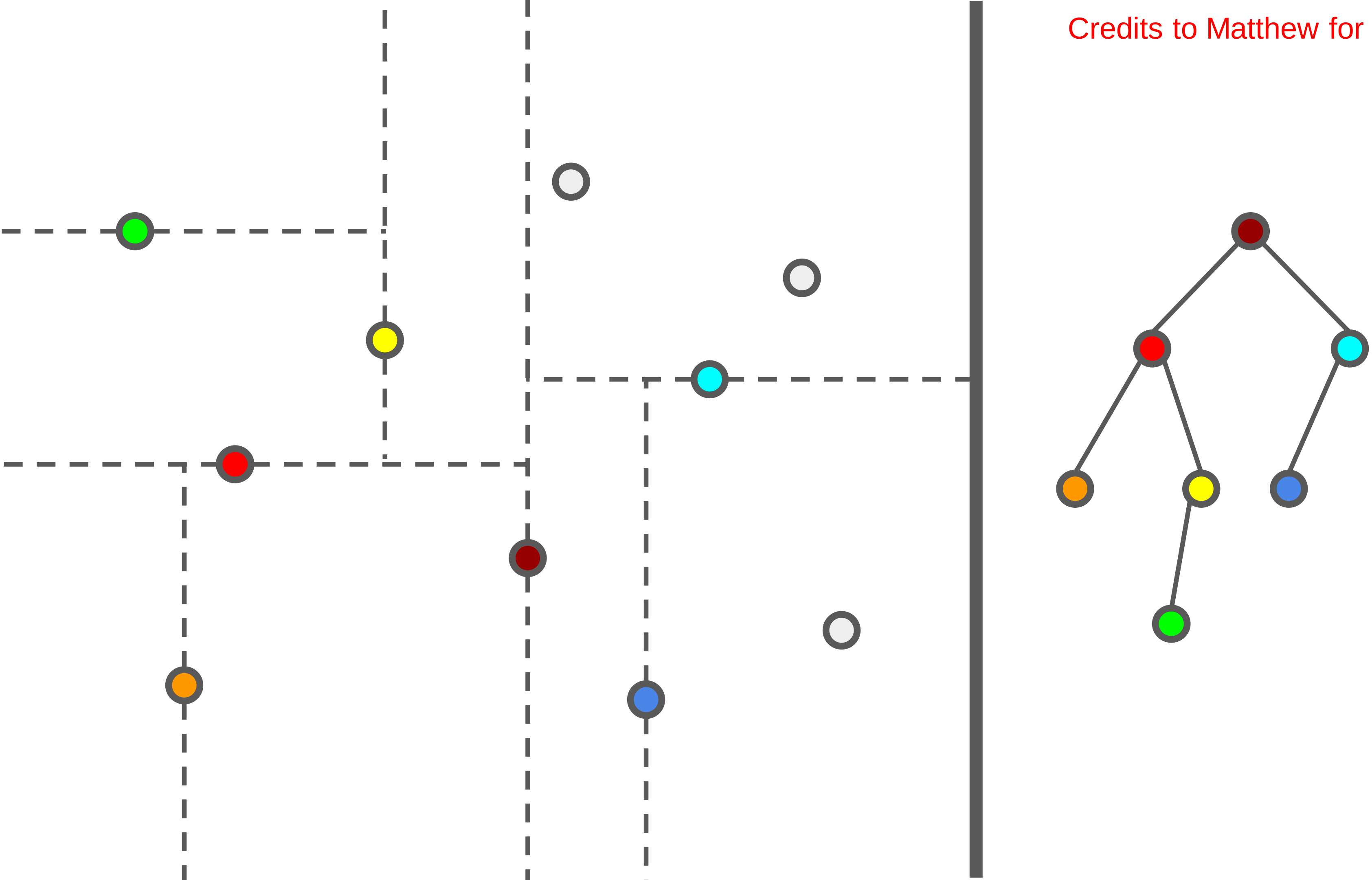
Credits to Matthew for this slide

Credits to Matthew for this slide



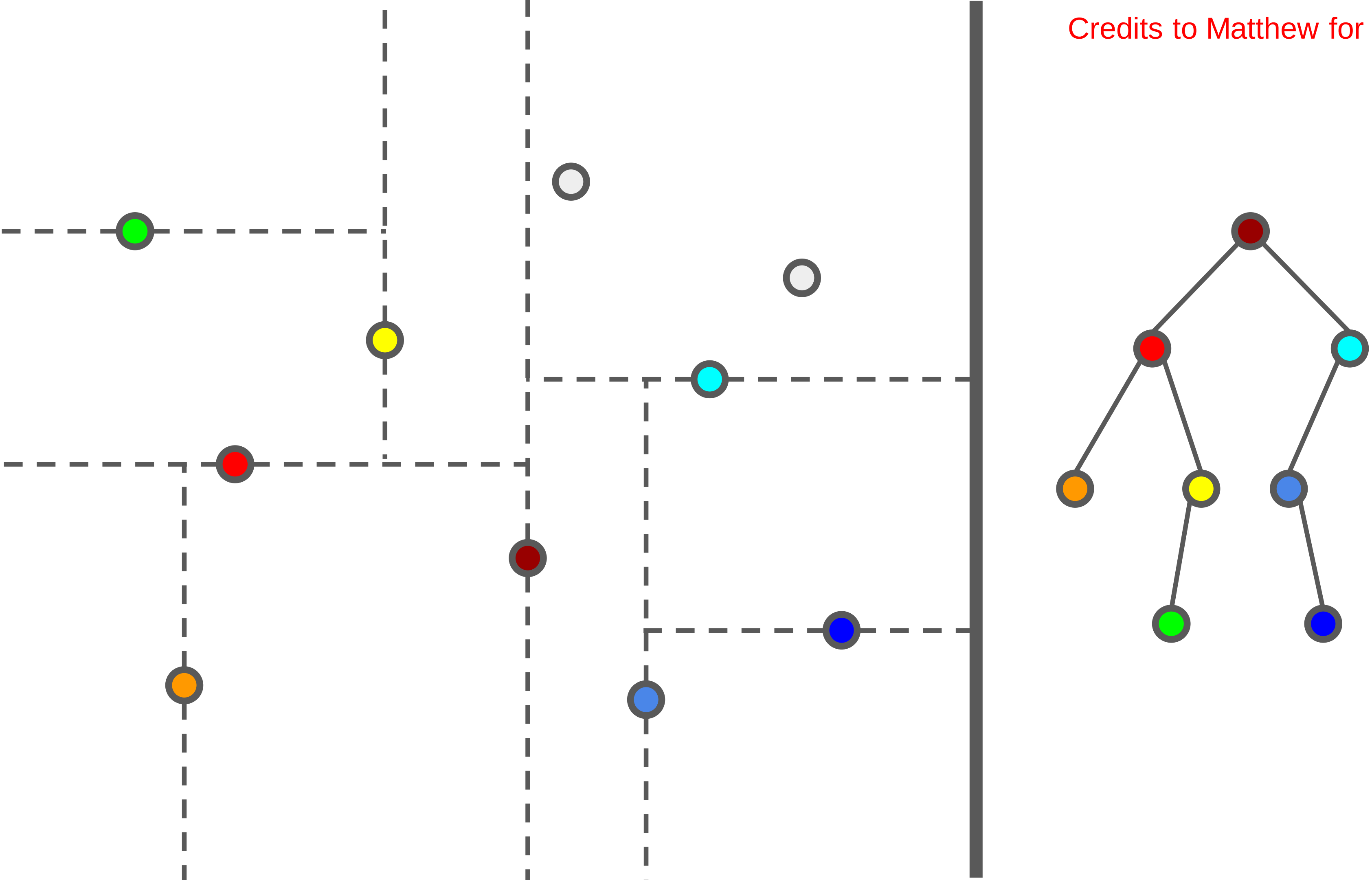


Credits to Matthew for this slide

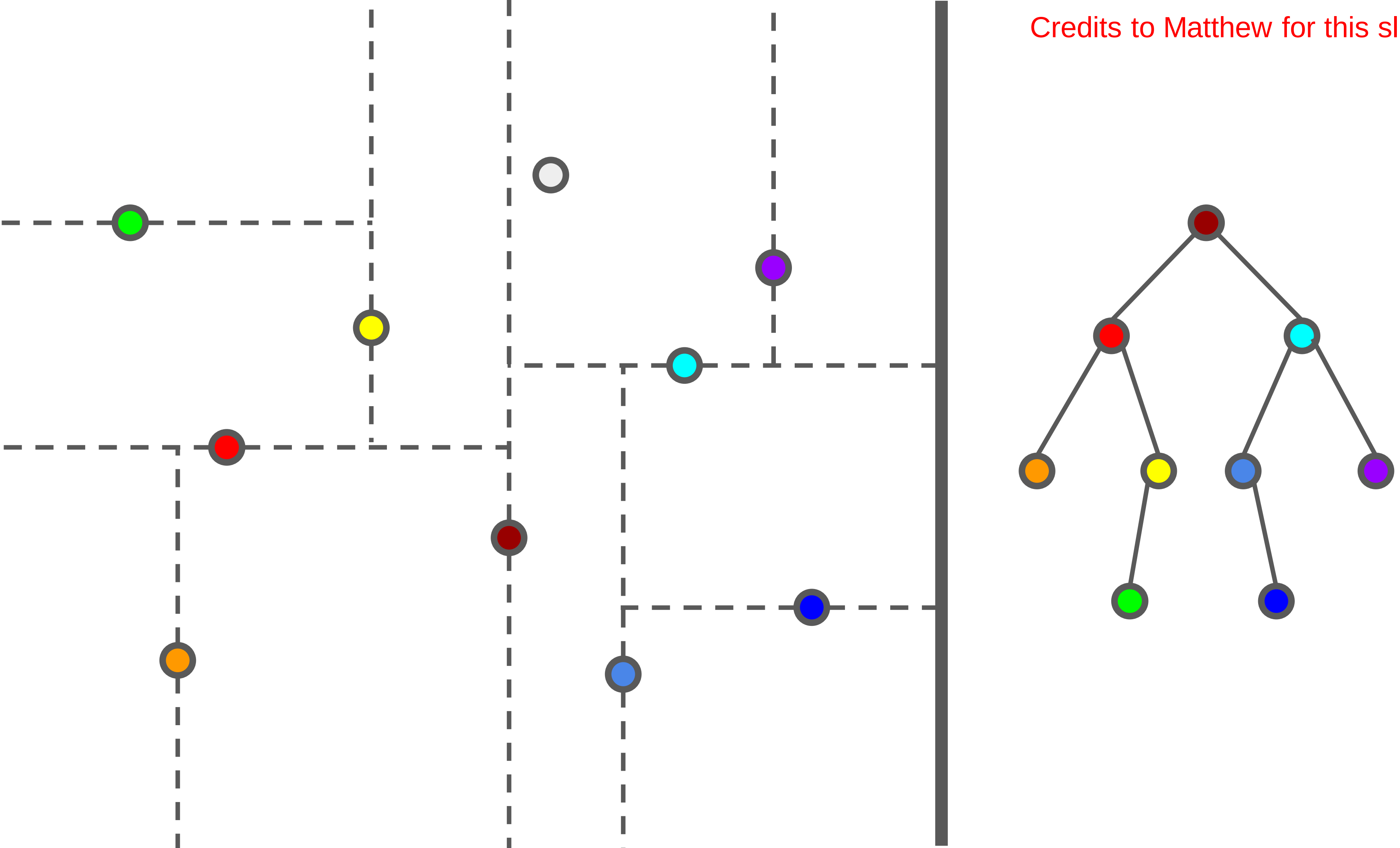


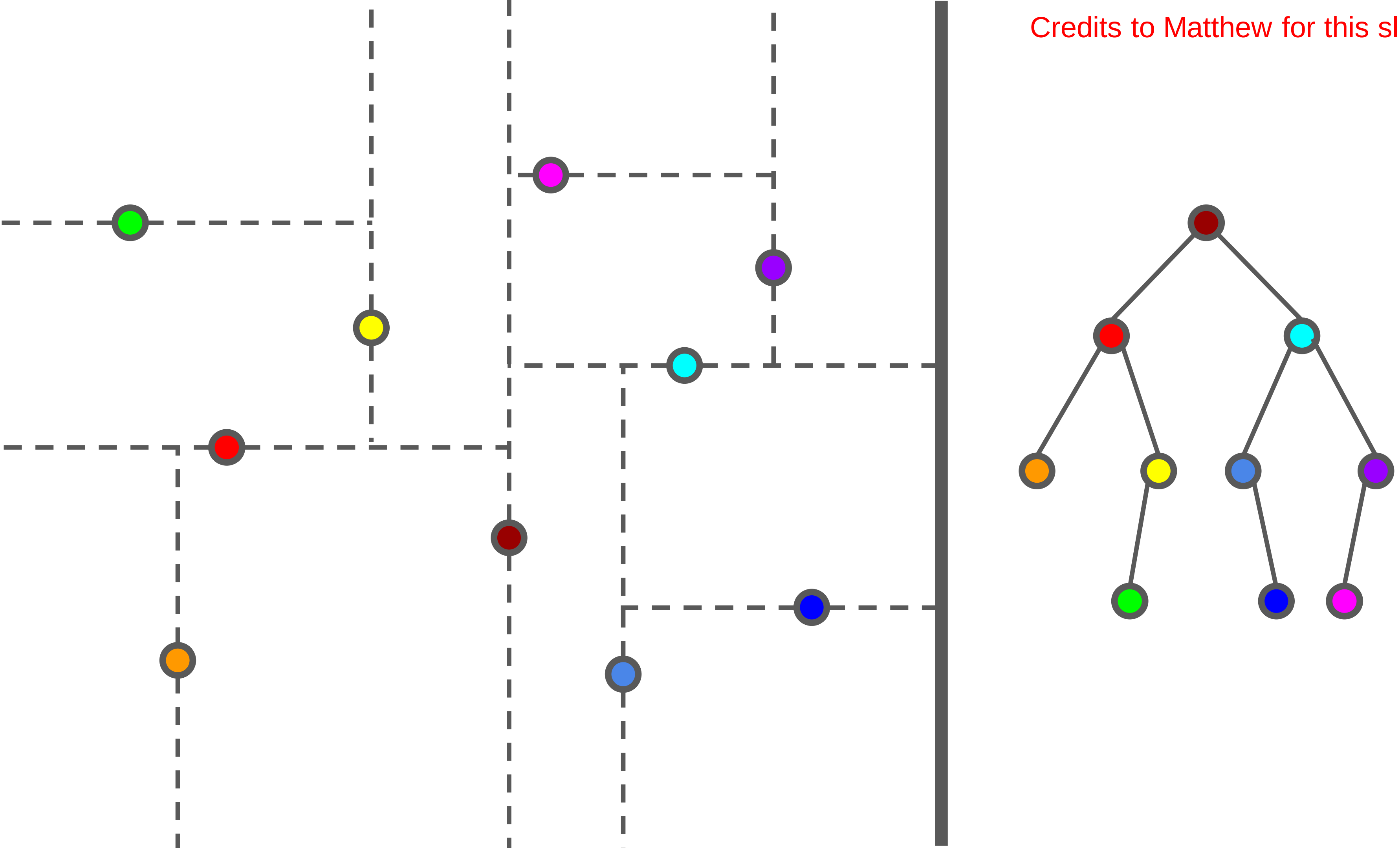
Credits to Matthew for this slide

Credits to Matthew for this slide



Credits to Matthew for this slide





Credits to Matthew for this slide

Problem 2b: kd-Trees

What is the running time of the construction algorithm?

- Find median element in x as root for the first level. Then build left tree and right tree
- Find median element in y as root for the second level. Then build left tree and right tree.

How can we find the median at each level? How fast can we do?

Problem 2b: kd-Trees

How can we find the median at each level? How fast can we do?

Option 1: Sort at every level

- Runtime recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- Runtime: $O(n \log^2 n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n\log n)$$

<u>work done</u>

$T(n)$ $\qquad\qquad$ $n\log n$

$T\left(\frac{n}{2}\right) \quad T\left(\frac{n}{2}\right)$ $\qquad\qquad$ $2\left(\frac{n}{2}\log\frac{n}{2}\right) = n\log\frac{n}{2}$

$T\left(\frac{n}{4}\right) \quad T\left(\frac{n}{4}\right) \quad T\left(\frac{n}{4}\right) \quad T\left(\frac{n}{4}\right)$ $\qquad\qquad$ $4\left(\frac{n}{4}\log\frac{n}{4}\right) = n\log\frac{n}{4}$

$\vdots$ $\qquad\qquad$ $\vdots$

$n\log 1$

$$T(n) = \overbrace{n\log n + n\log\frac{n}{2} + \cdots + n\log 1}^{\log n}$$

$$= n\left[\log\left(n\cdot\frac{n}{2}\cdot\frac{n}{4}\cdot\cdots\cdot 1\right)\right]$$

$$= n\log\left(\frac{n^{\log n}}{2\cdot 4\cdot\cdots}\right) = O(n\log^2 n)$$

Problem 2b: kd-Trees

How can we find the median at each level? How fast can we do?

Option 1: Sort at every level

- Runtime recurrence: $T(n) = 2T(n/2) + O(n \log n)$
- Runtime: $O(n \log^2 n)$

Option 2: Quickselect

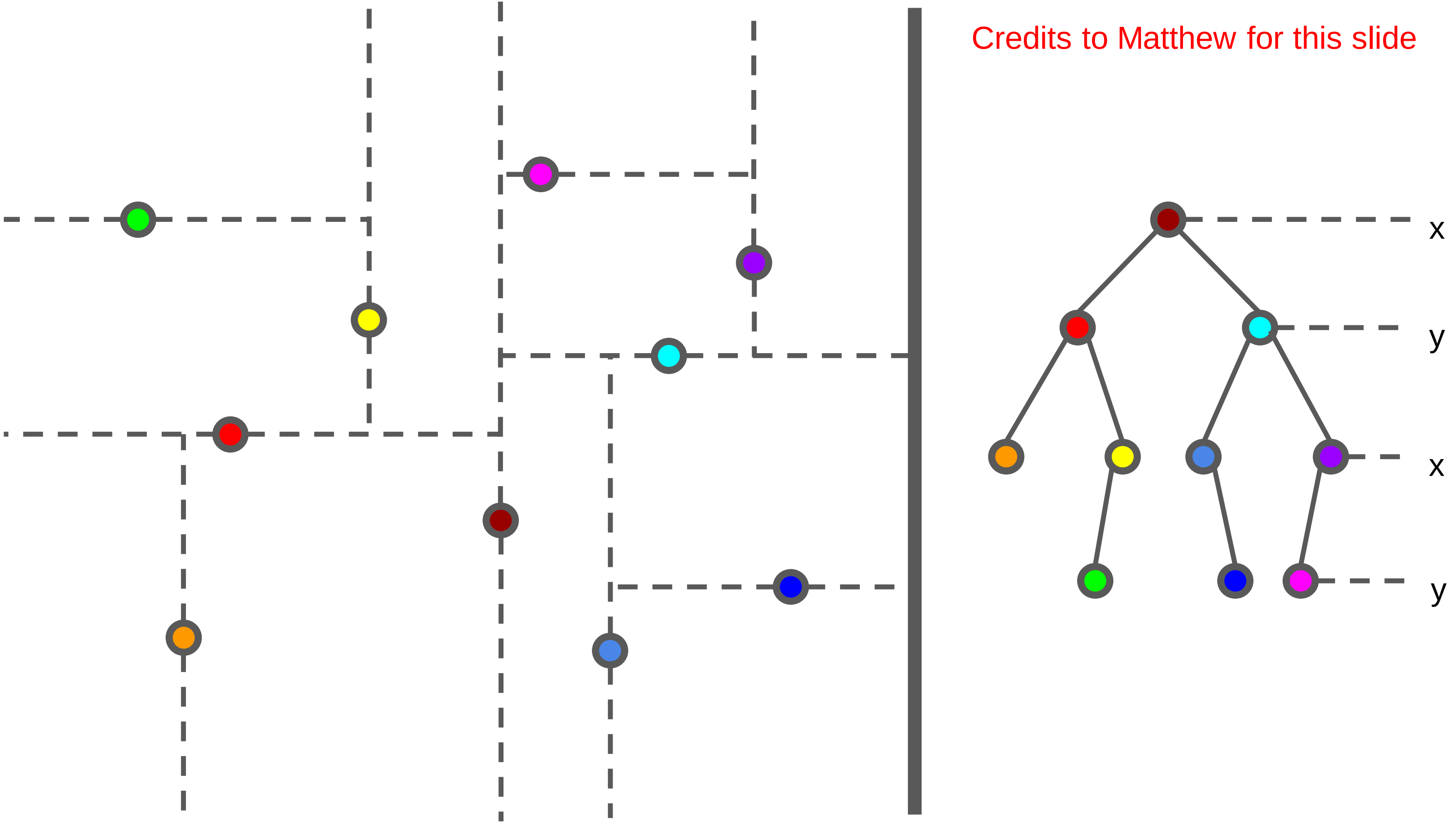
- Runtime recurrence: $T(n) = 2T(n/2) + O(n)$
- Runtime: $O(n \log n)$

Problem 2c: kd-Trees

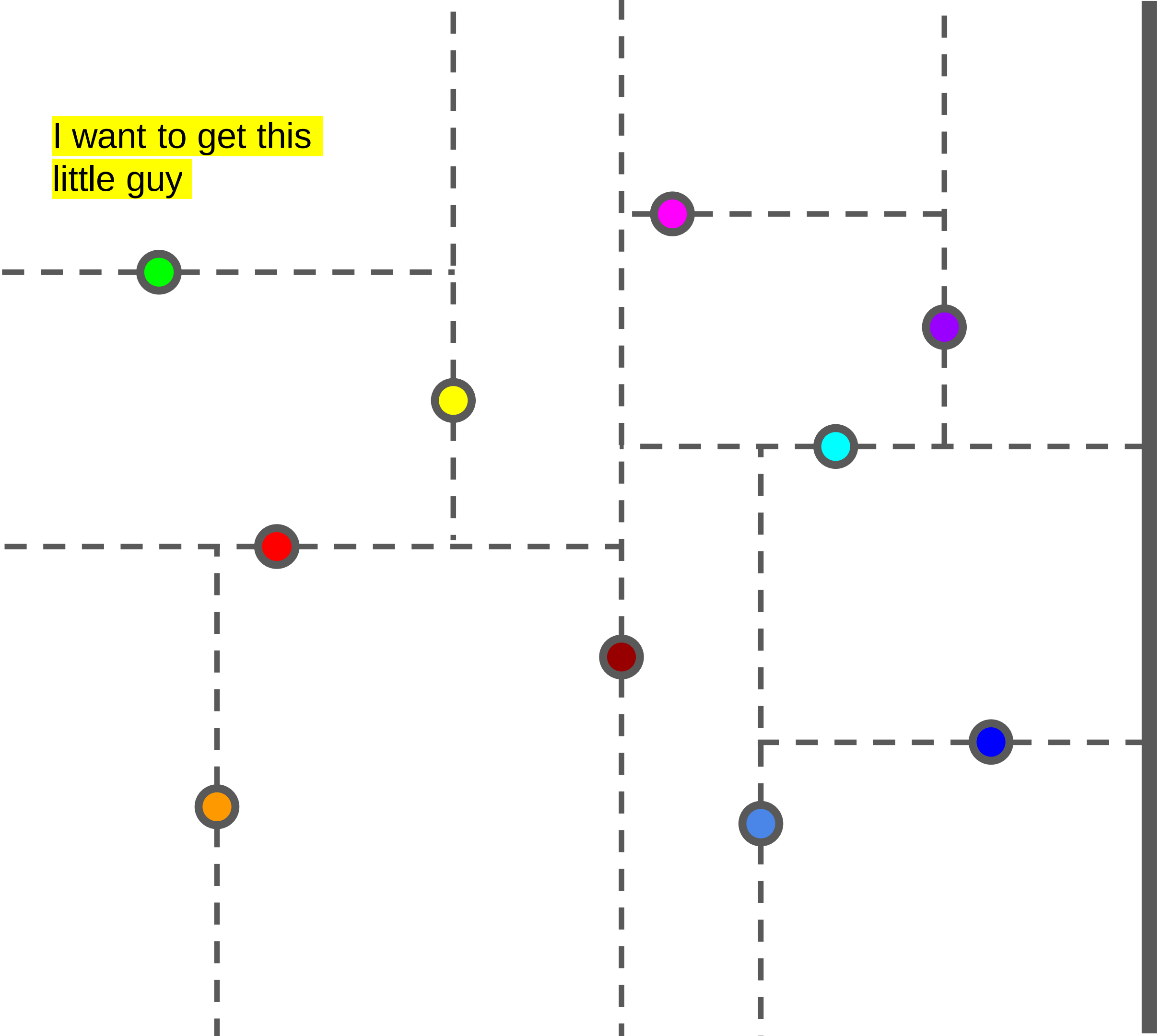
How would you find the element with the minimum (or maximum) x-coordinate in a kd-tree? How expensive can it be, if the tree is perfectly balanced?

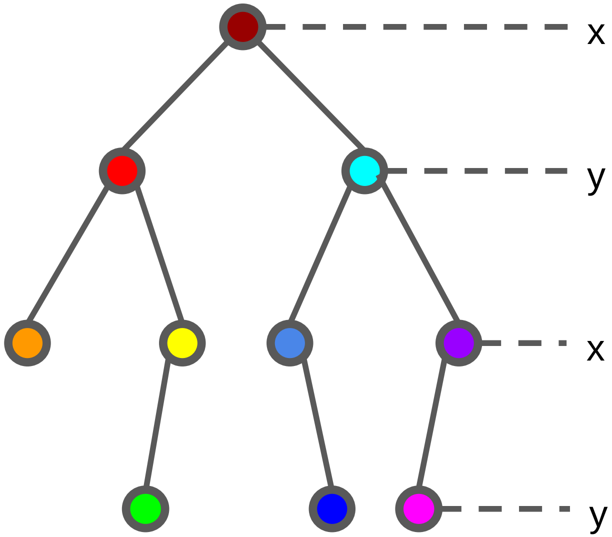
Problem 2c: Min/Max x-coordinate in kd-Tree

Credits to Matthew for this slide



Credits to Matthew for this slide

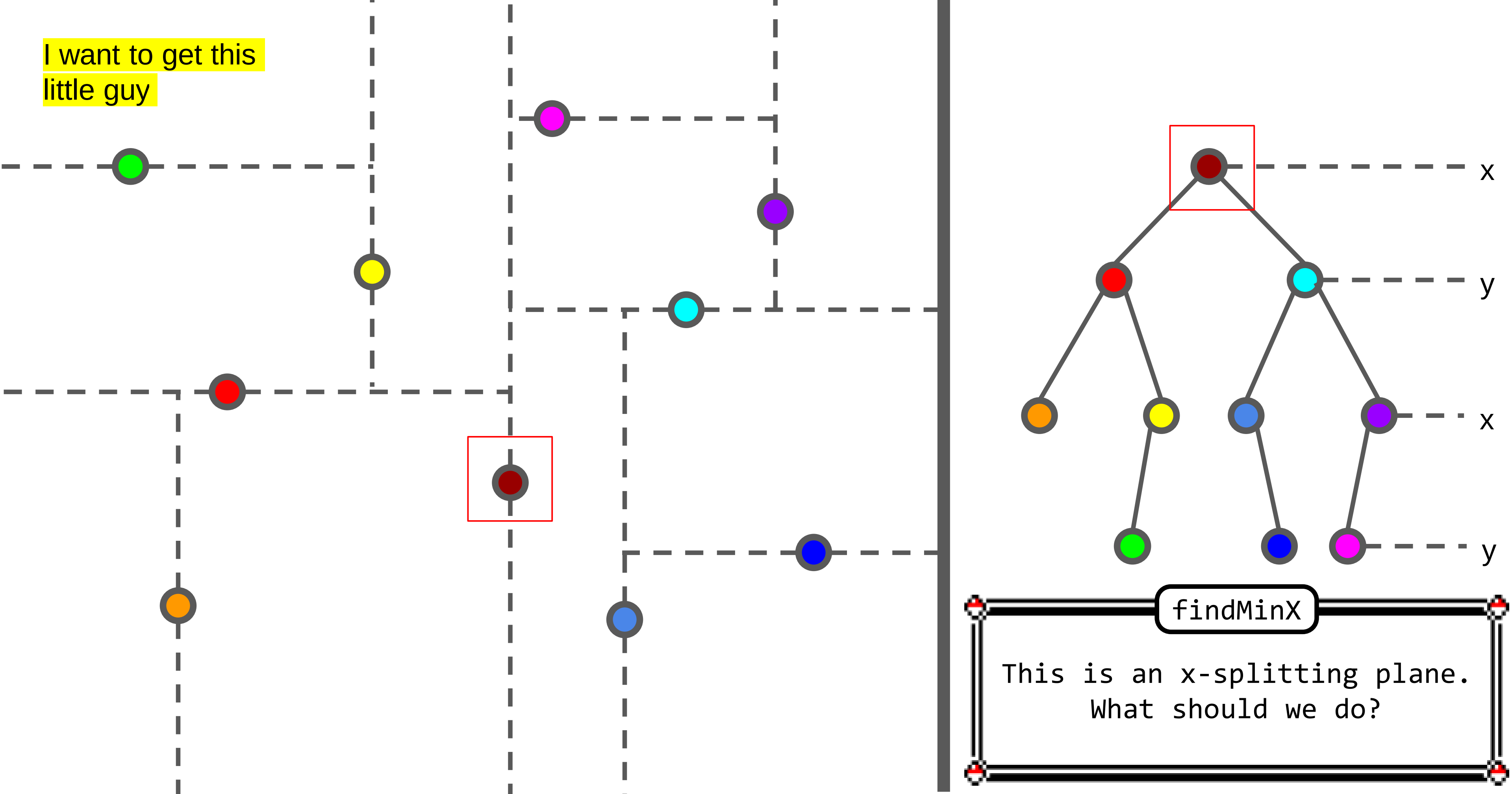




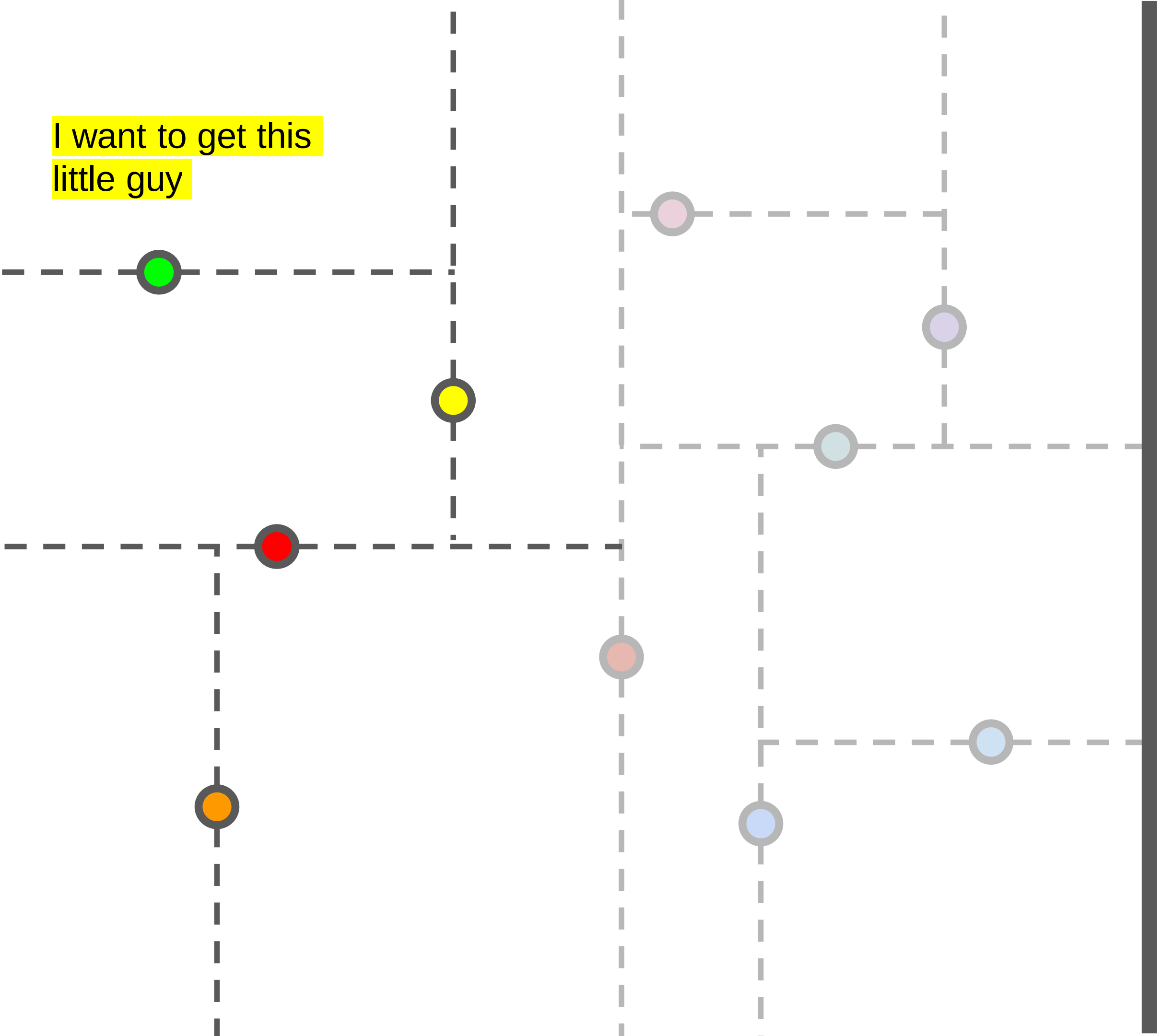
findMinX

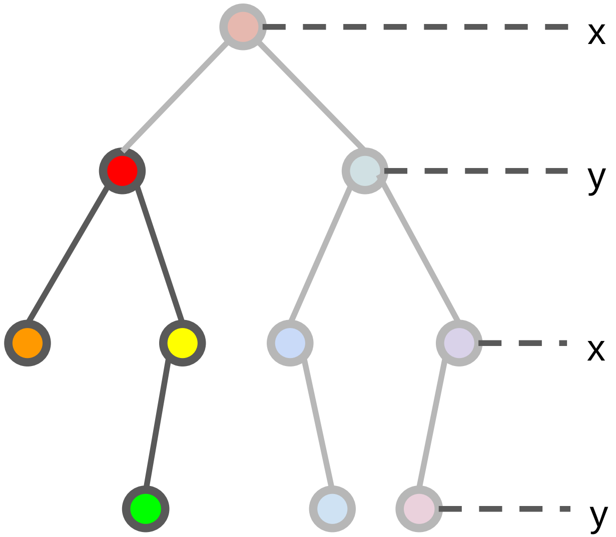
We want to find the node with smallest x-coordinate

Credits to Matthew for this slide



Credits to Matthew for this slide

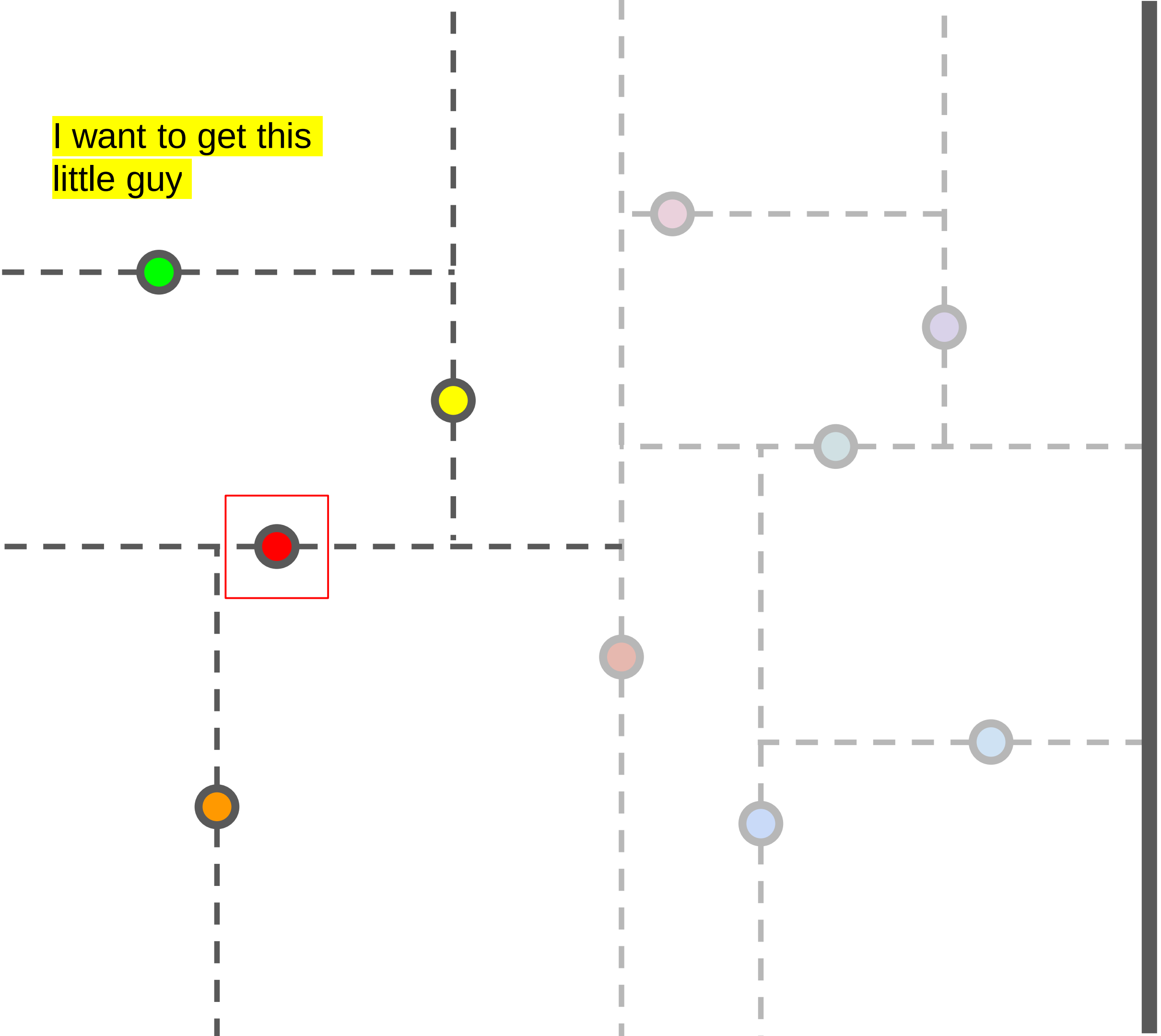


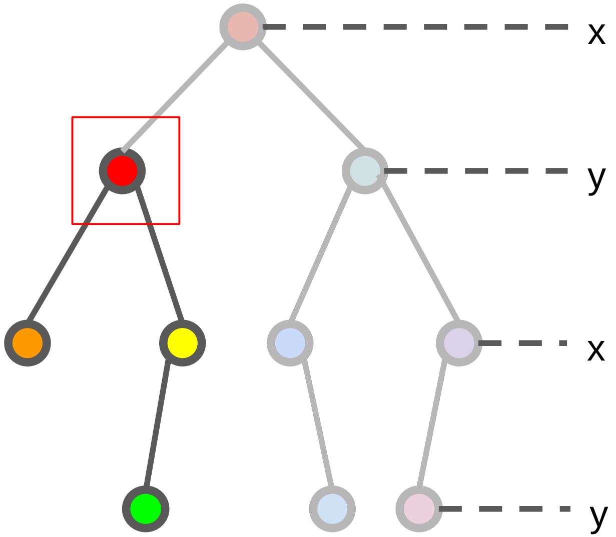


findMinX

Go left!

Credits to Matthew for this slide

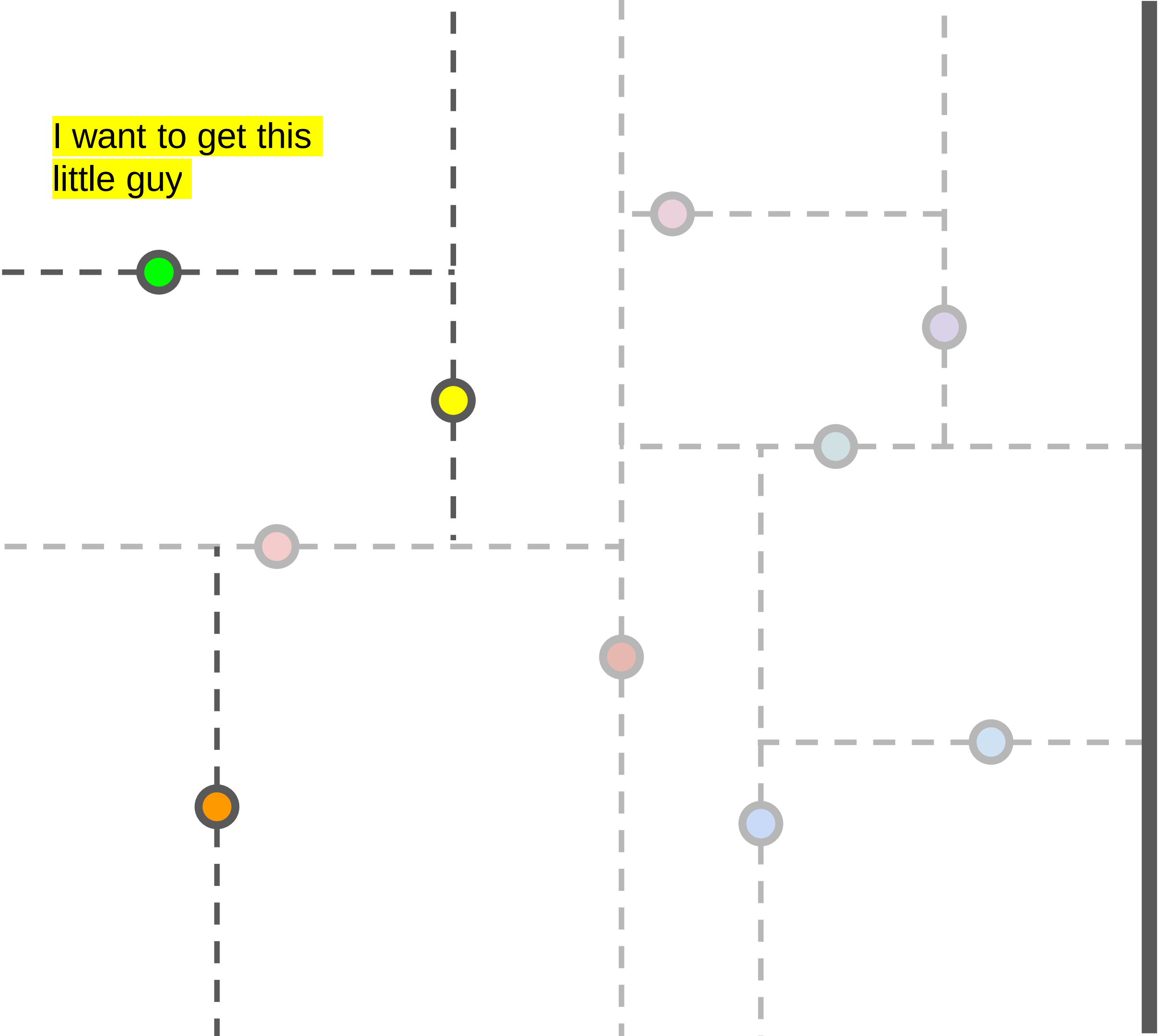


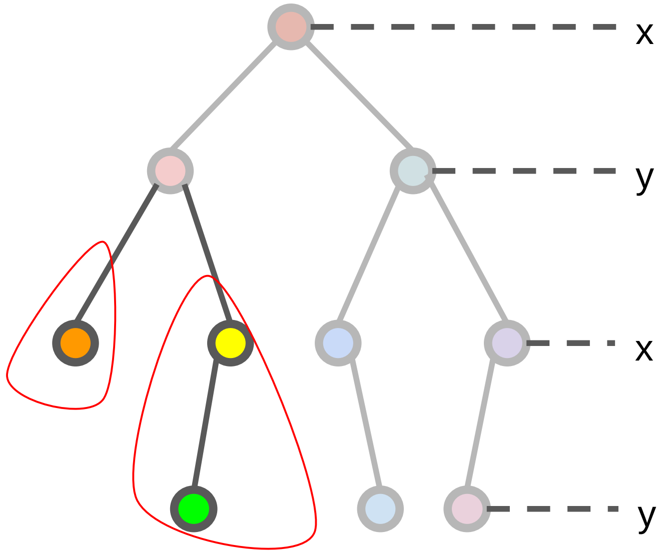


findMinX

We are now in the y-splitting plane. wat do

Credits to Matthew for this slide





findMinX

No choice but to go to both trees! No info about x-coord

Problem 2c: kd-Trees

What’s the recurrence relation?

Problem 2c: kd-Trees

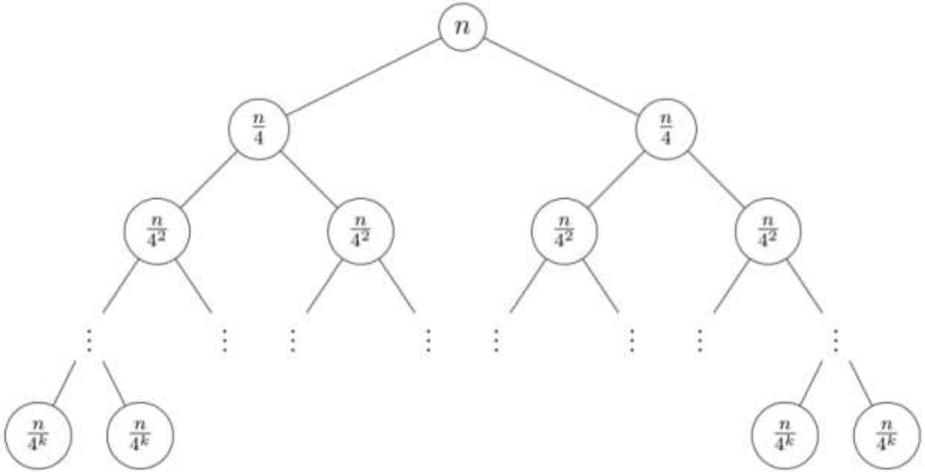
What’s the recurrence relation?

$T(n) = 2T(n / 4) + O(1)$

$n / 4$ because we explore the subtree 2 levels down.

$2T(n / 4)$ because during the y-split, we have to explore both trees.

$$T(n) = 2T\left(\frac{n}{4}\right) + O(1) = O(\sqrt{n})$$



No. of nodes
1
2
4
$\vdots$
$2^k = 2^{\log_4 n}$

$$\begin{aligned}
& 1 + 2 + 4 + \cdots + 2^{\log_4 n} \\
&= 2^{\log_4(n)+1} - 1 \\
&= 2^{\frac{\log n}{2}+1} - 1 \left(\because \log_4 n = \frac{\log_2 n}{\log_2 4}\right) \\
&= 2 \cdot \left(2^{\log n}\right)^{\frac{1}{2}} - 1 \\
&= 2n^{\frac{1}{2}} - 1 \\
&= O(\sqrt{n})
\end{aligned}$$

Problem 2c: kd-Trees

What's the recurrence relation?

$T(n) = 2T(n / 4) + O(1)$

$n / 4$ because we explore the subtree 2 levels down.

$2T(n / 4)$ because during the y-split, we have to explore both trees.

Runtime: $O(\sqrt{n})$

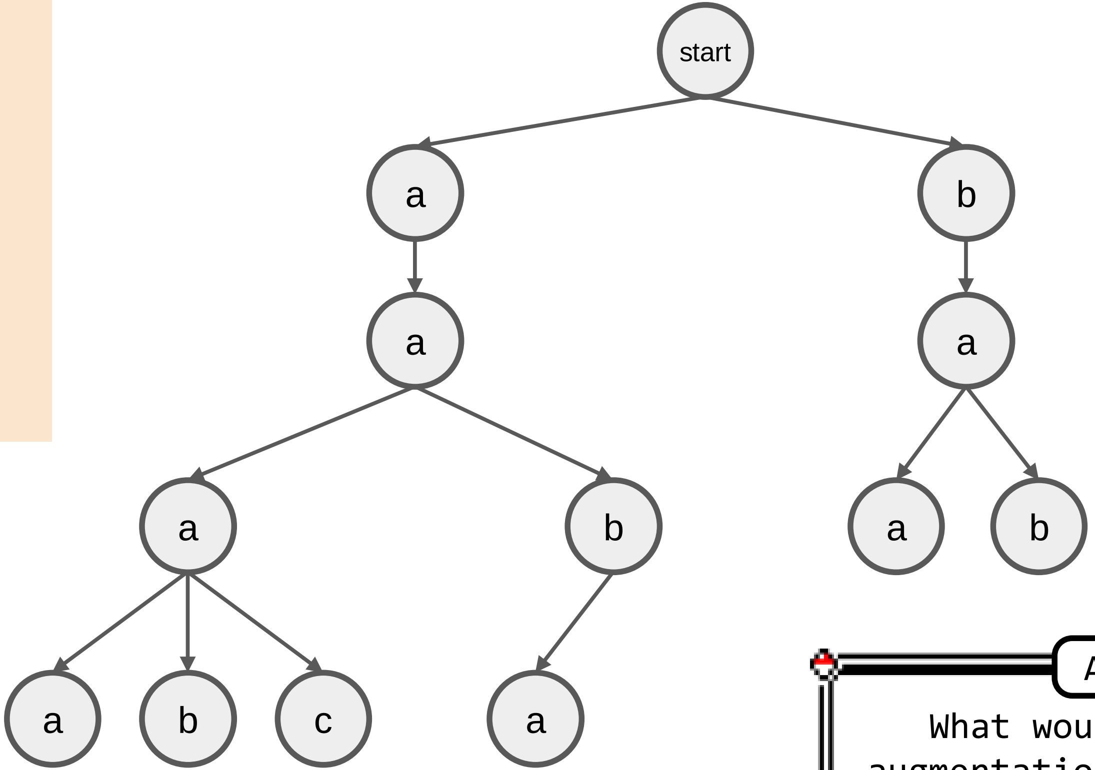
Problem 3: Tries

Coming up with a good name for your baby is hard. Imagine you want to build a data structure to help answer these types of questions. Your data structure should support the following operations:

- `insert(name, gender, count)`: adds a name of a given gender, with a count of how many babies have that name.
- `countName(name, gender)`: returns the number of babies with that name and gender.
- `countPrefix(prefix, gender)`: returns the number of babies with that prefix of their name and gender.
- `countBetween(begin, end, gender)`: returns the number of babies with names that are lexicographically after begin and before end that have the proper gender.

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

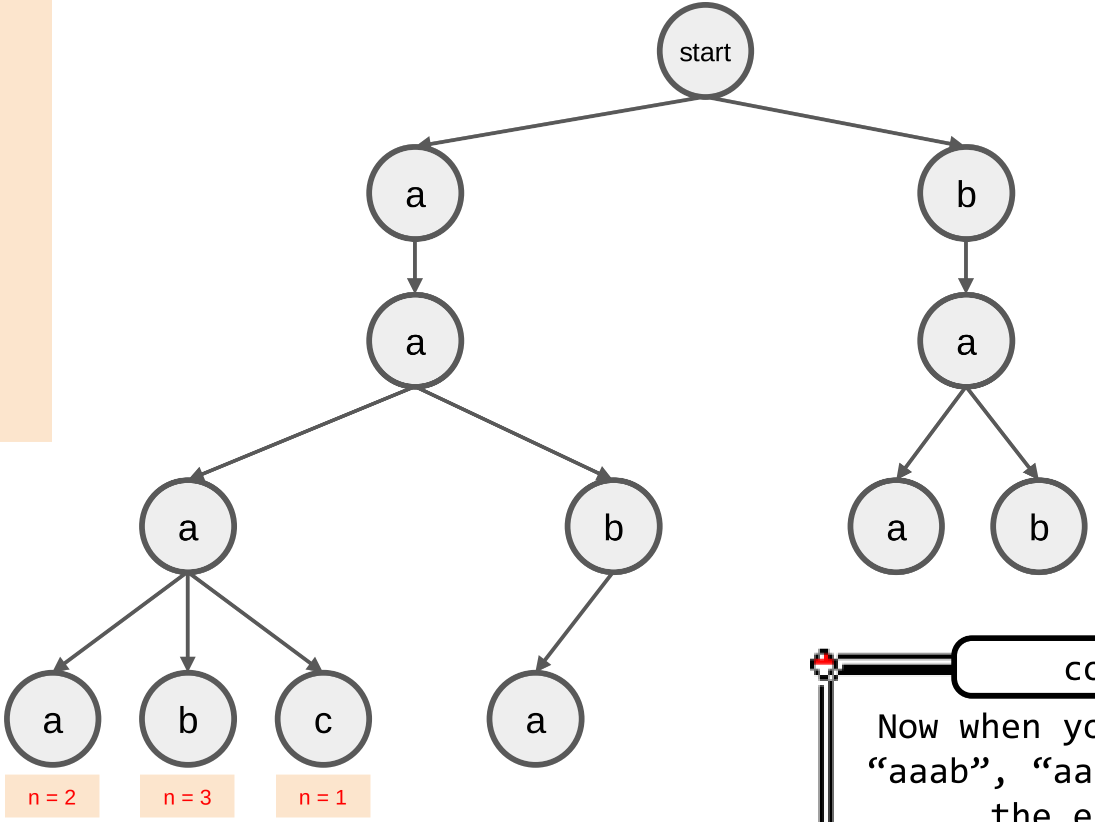


Augment

What would be a useful augmentation? Maybe we start with the one at the leaves

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

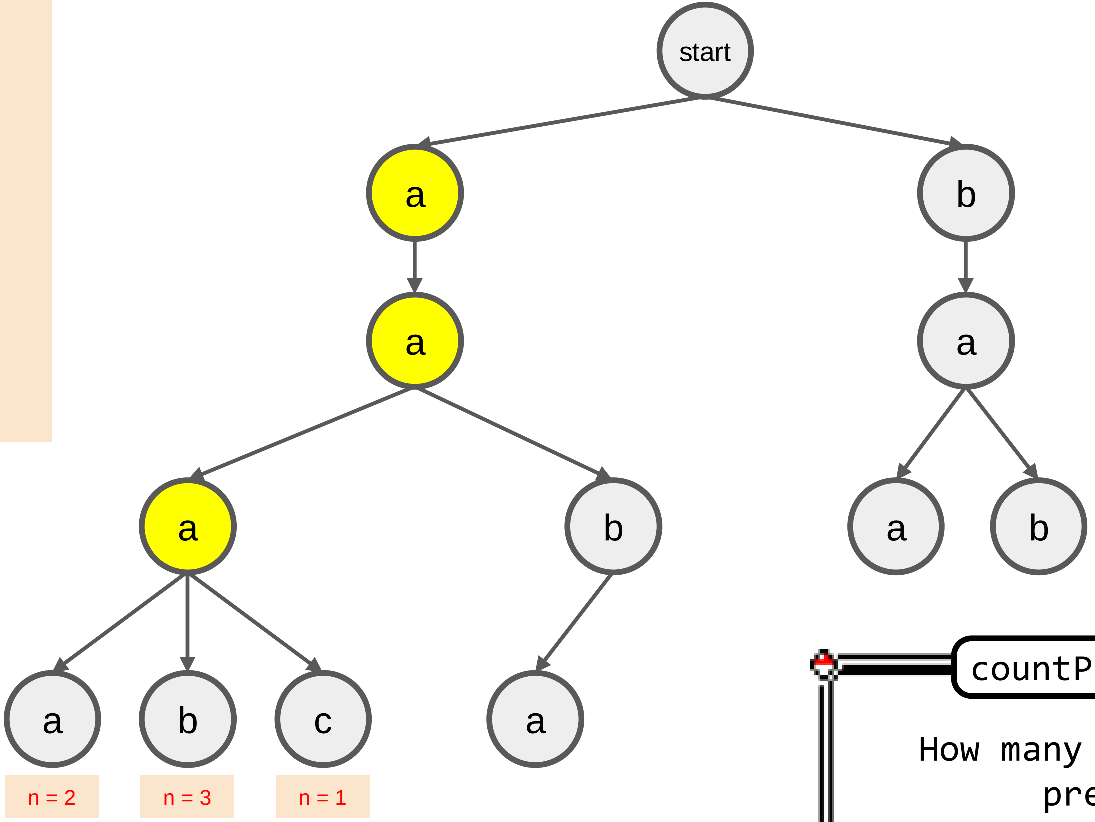


countName

Now when you search “aaaa”, “aaab”, “aaac”, you will get the exact count!

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

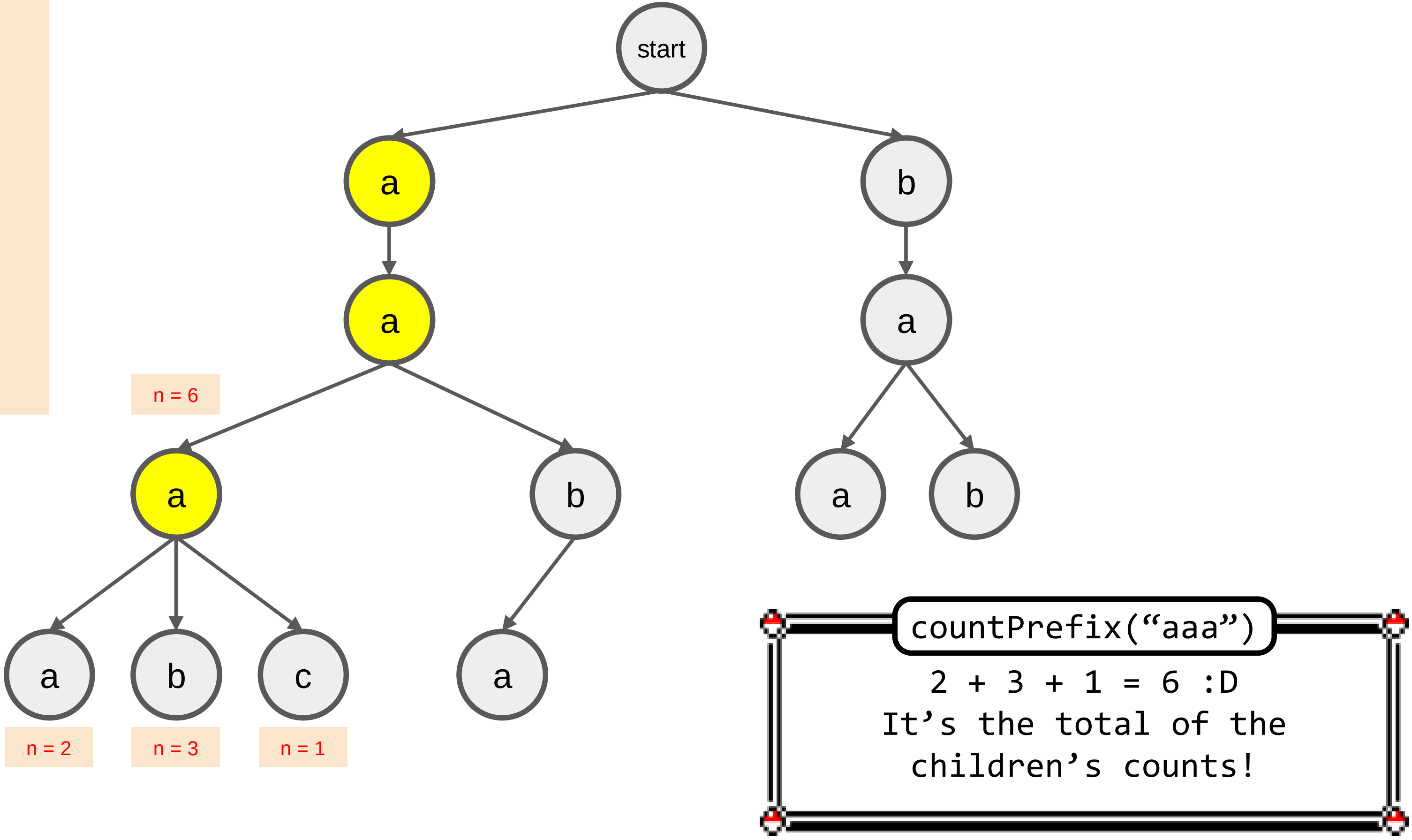


`countPrefix(“aaa”)`

How many names have the prefix aaa?

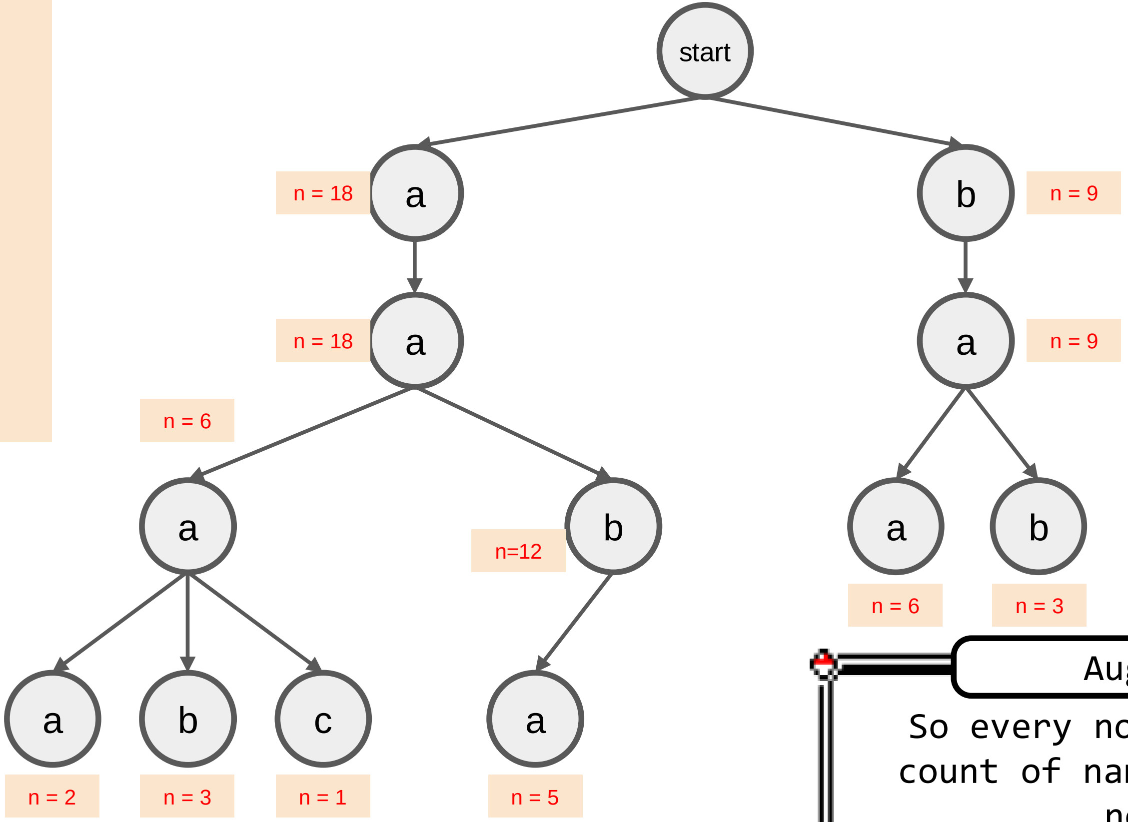
Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)



Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

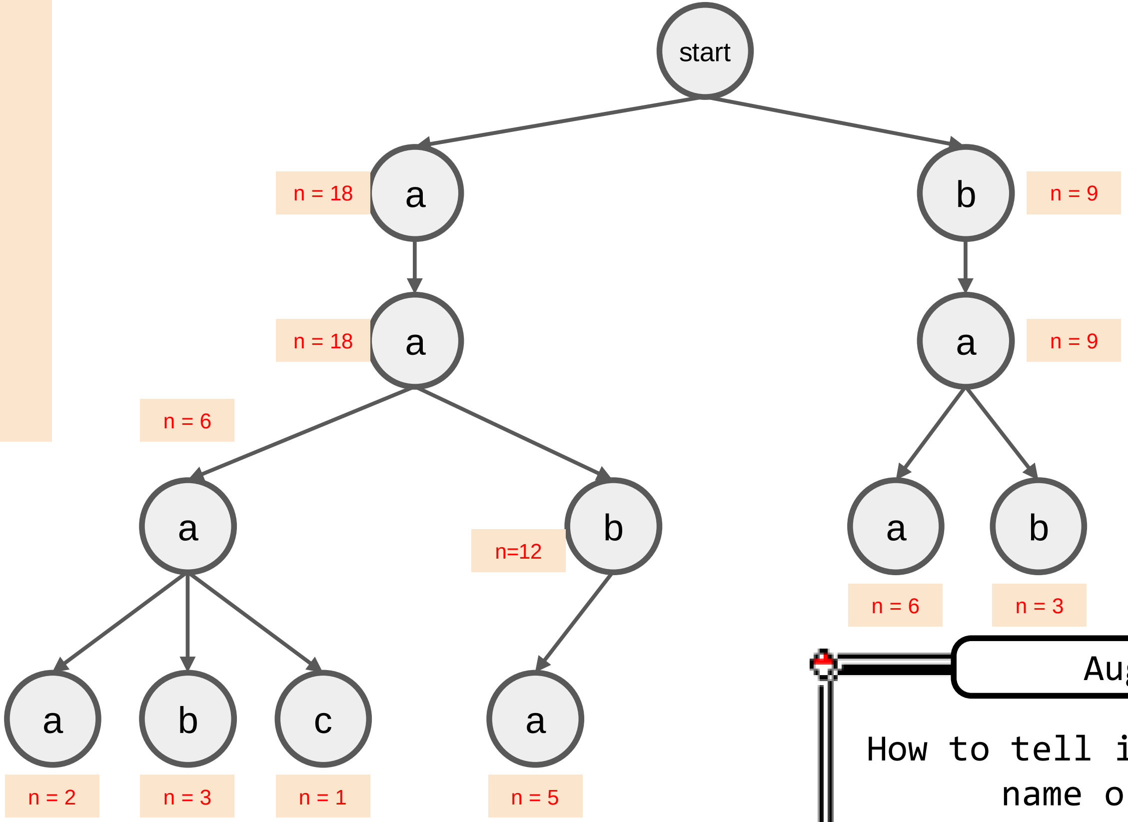


Augment

So every node stores the count of names under that node!

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

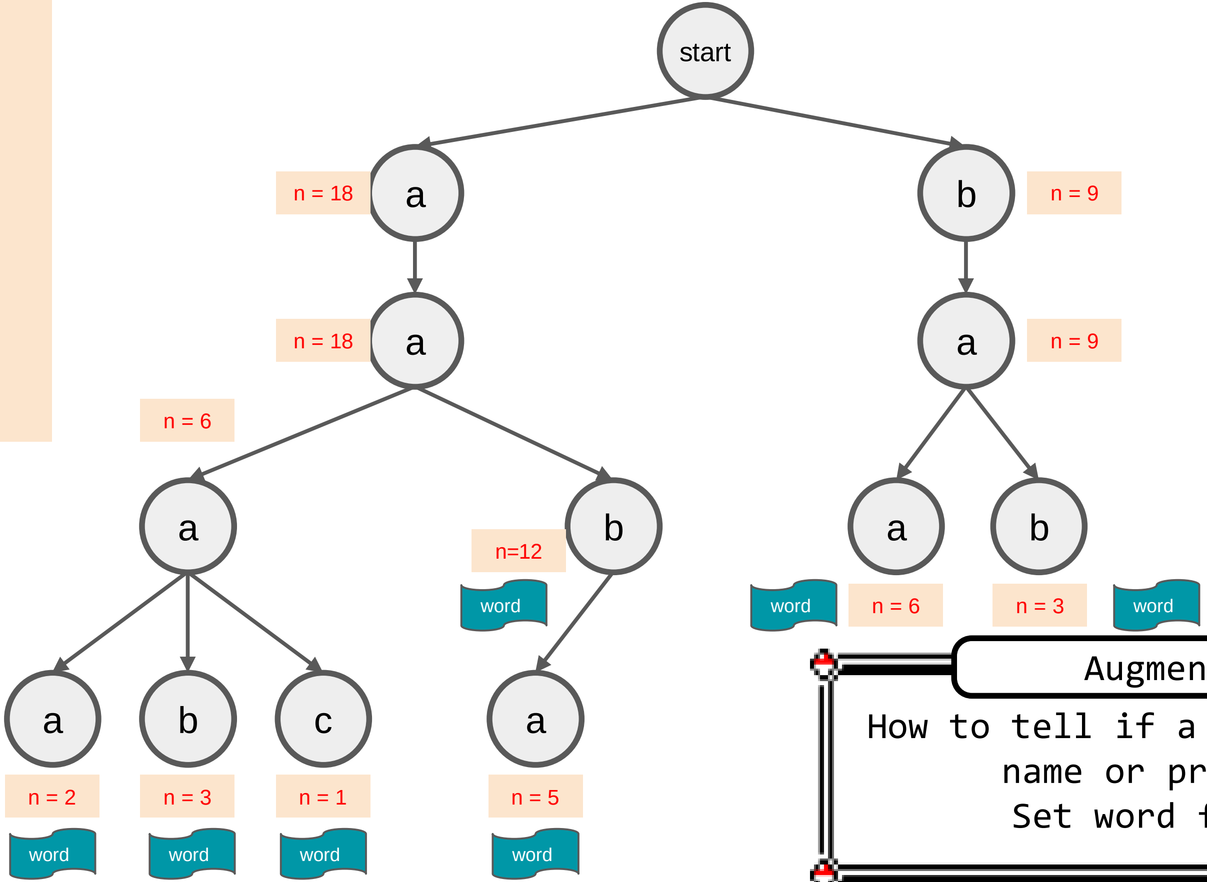


Augment

How to tell if a string is a name or prefix?

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

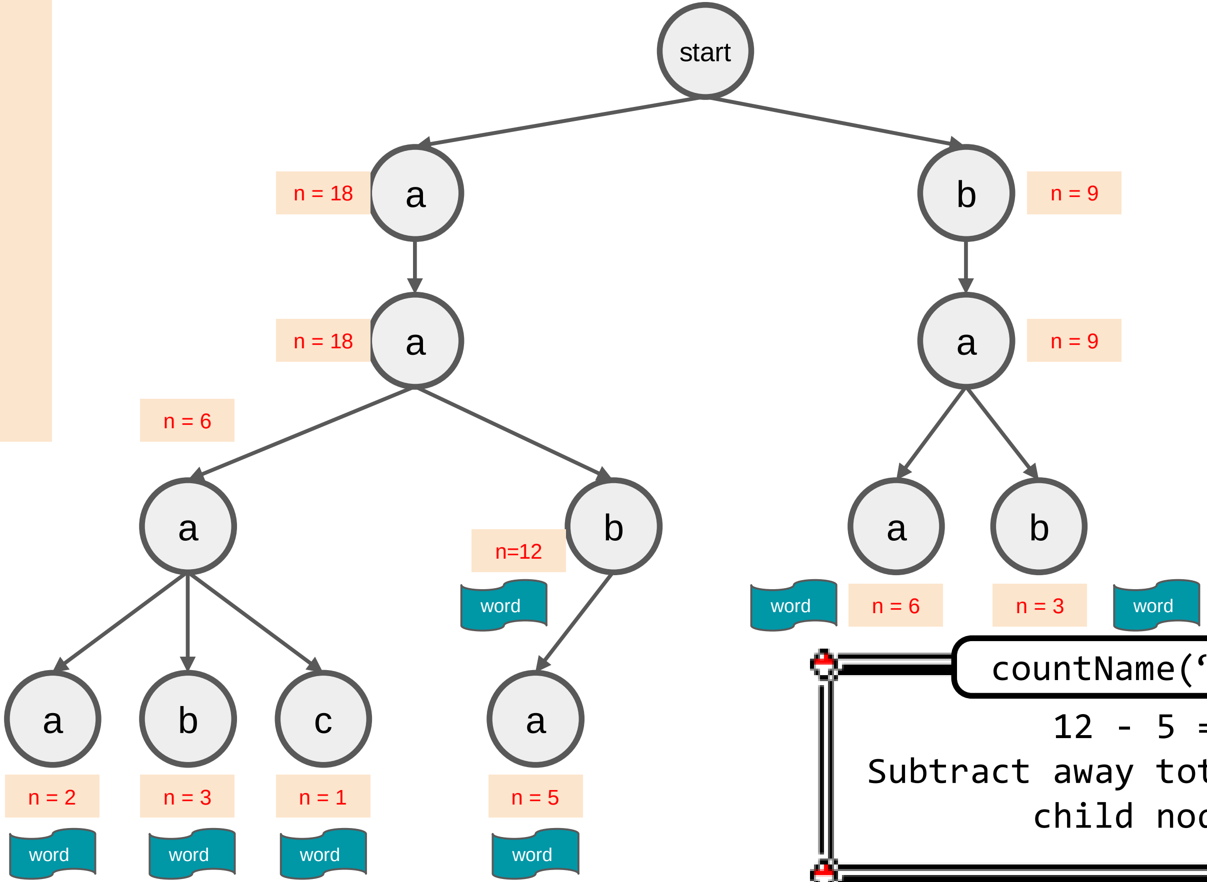


Augment

How to tell if a string is a name or prefix?
Set word flag!

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)



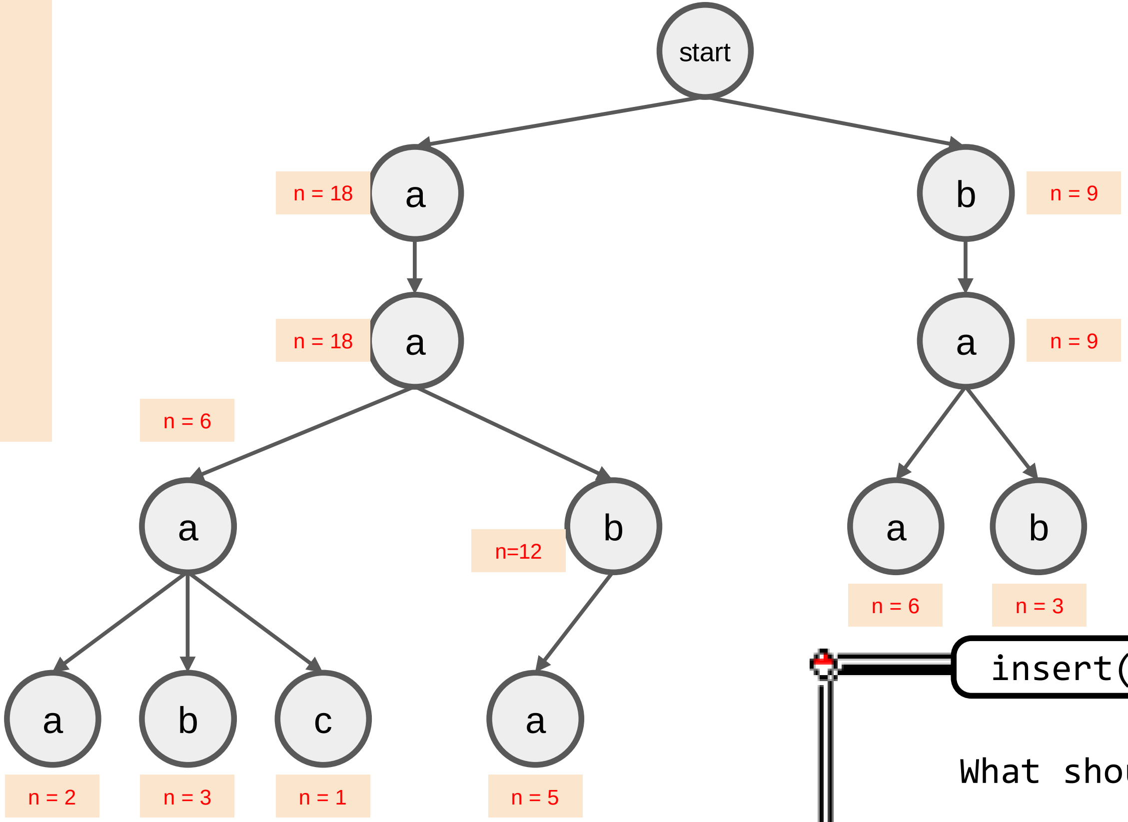
countName(“aab”)

12 - 5 = 7

Subtract away total count of child nodes!

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)

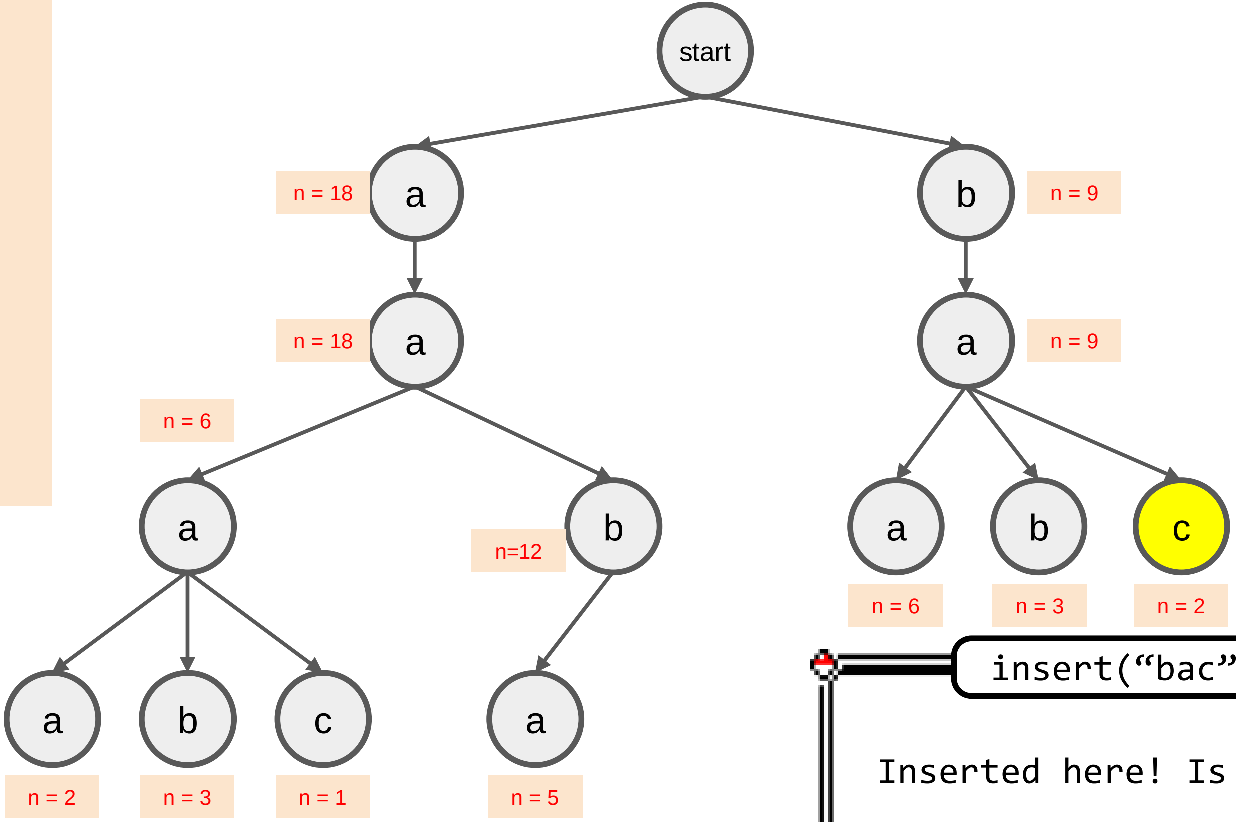


insert(“bac”, 2)

What should happen?

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)
- "bac" (2)

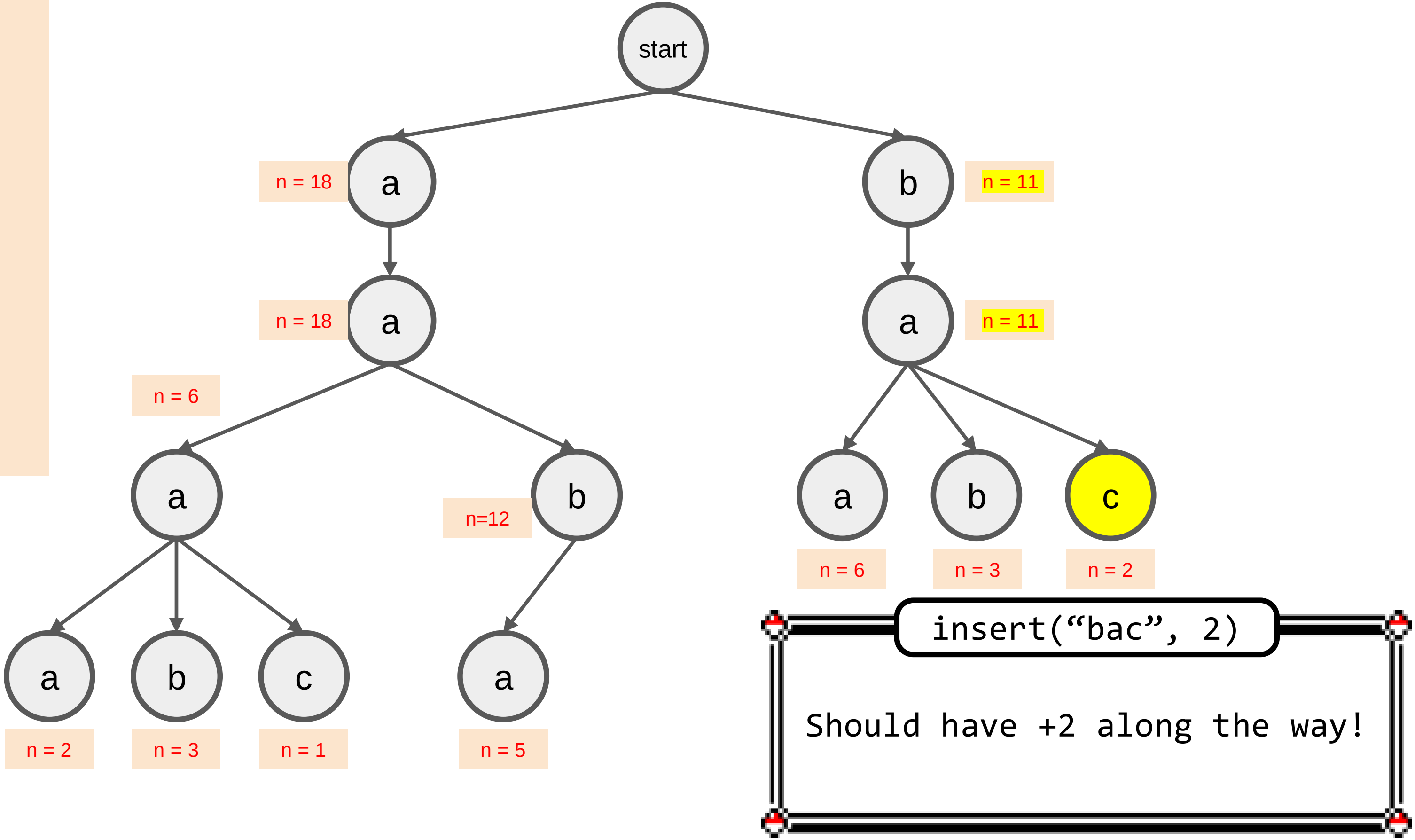


insert("bac", 2)

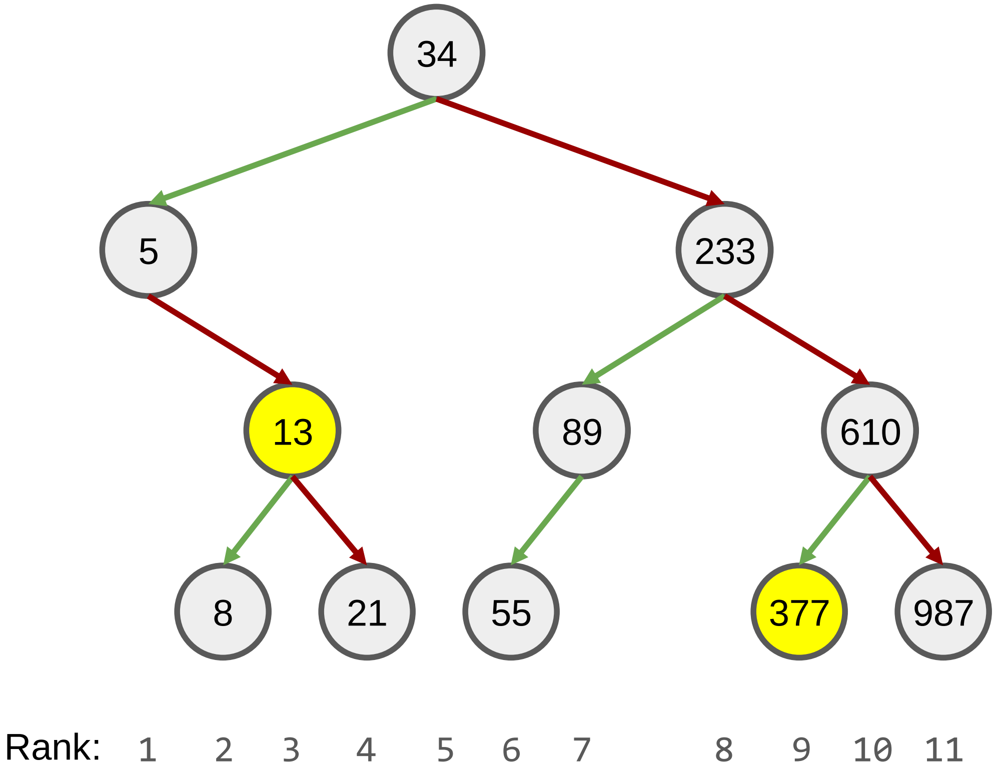
Inserted here! Is that all?

Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)
- "bac" (2)

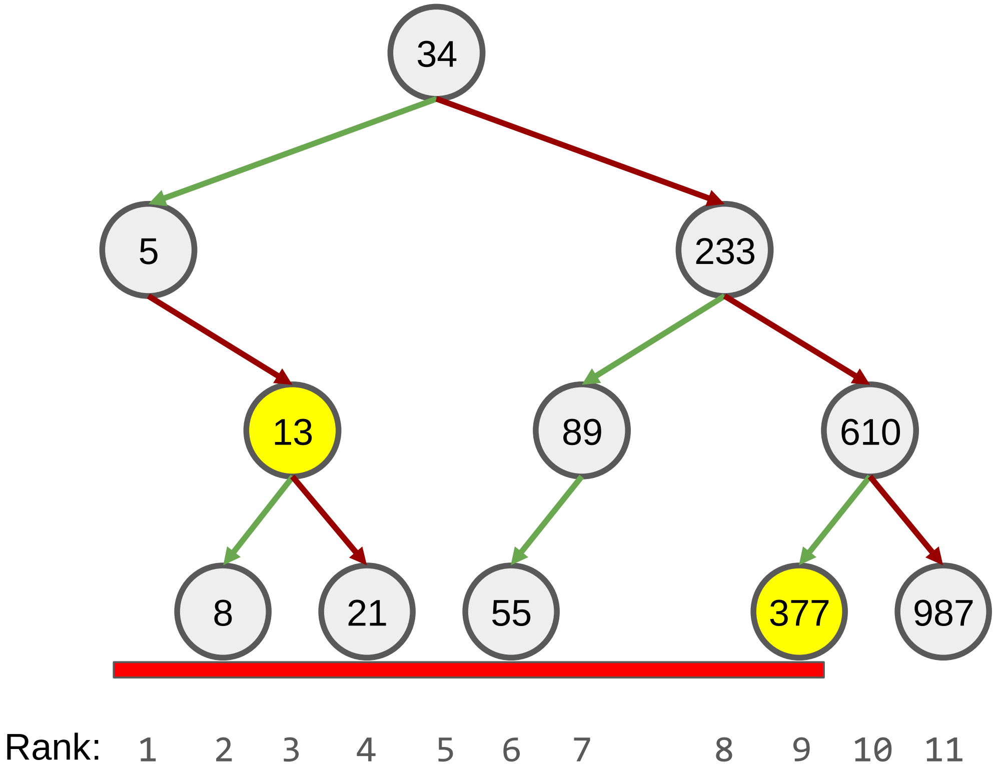


Intermezzo: Suppose given this AVL tree. How can you efficiently count the number of nodes between 13 and 377 (both exclusive)?



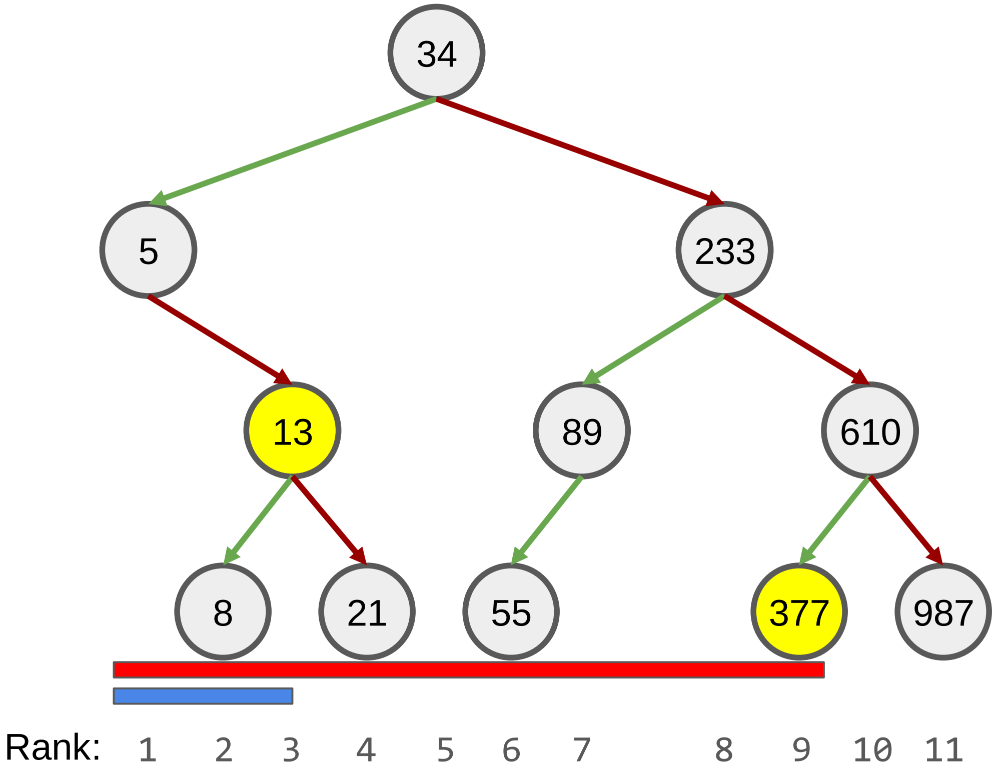
Intermezzo: Suppose given this AVL tree. How can you efficiently count the number of nodes between 13 and 377 (both exclusive)?

Take ranks!
`rank(377)` covers 9 elements



Intermezzo: Suppose given this AVL tree. How can you efficiently count the number of nodes between 13 and 377 (both exclusive)?

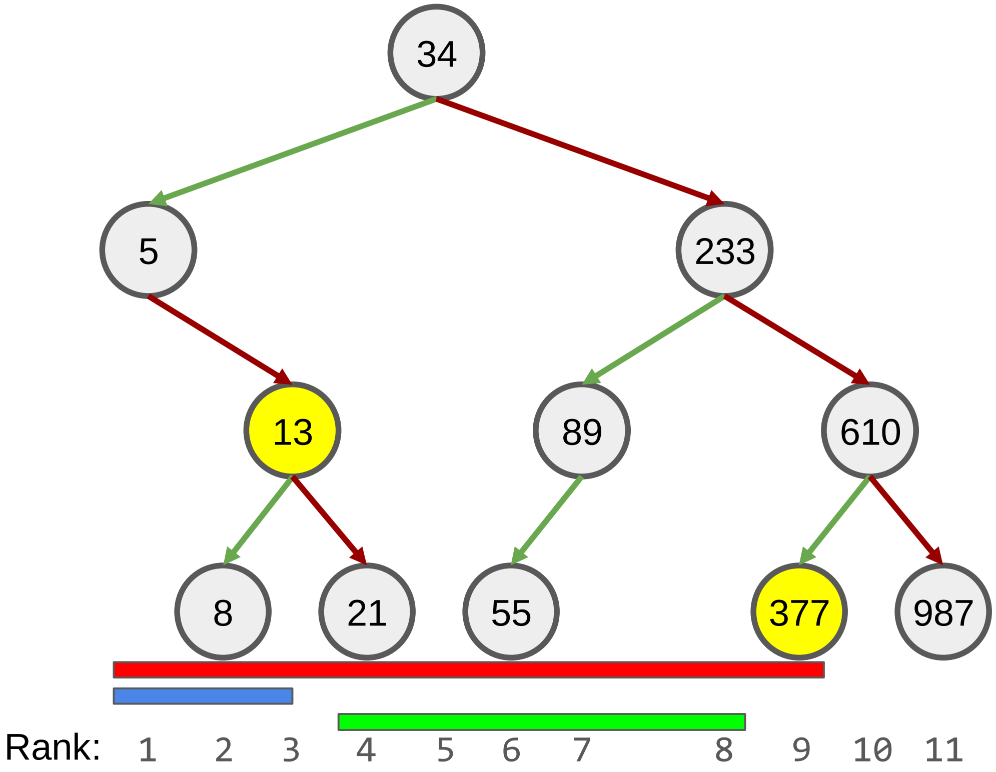
Take ranks!
`rank(377)` covers 9 elements
`rank(13)` covers 3 elements



Intermezzo: Suppose given this AVL tree. How can you efficiently count the number of nodes between 13 and 377 (both exclusive)?

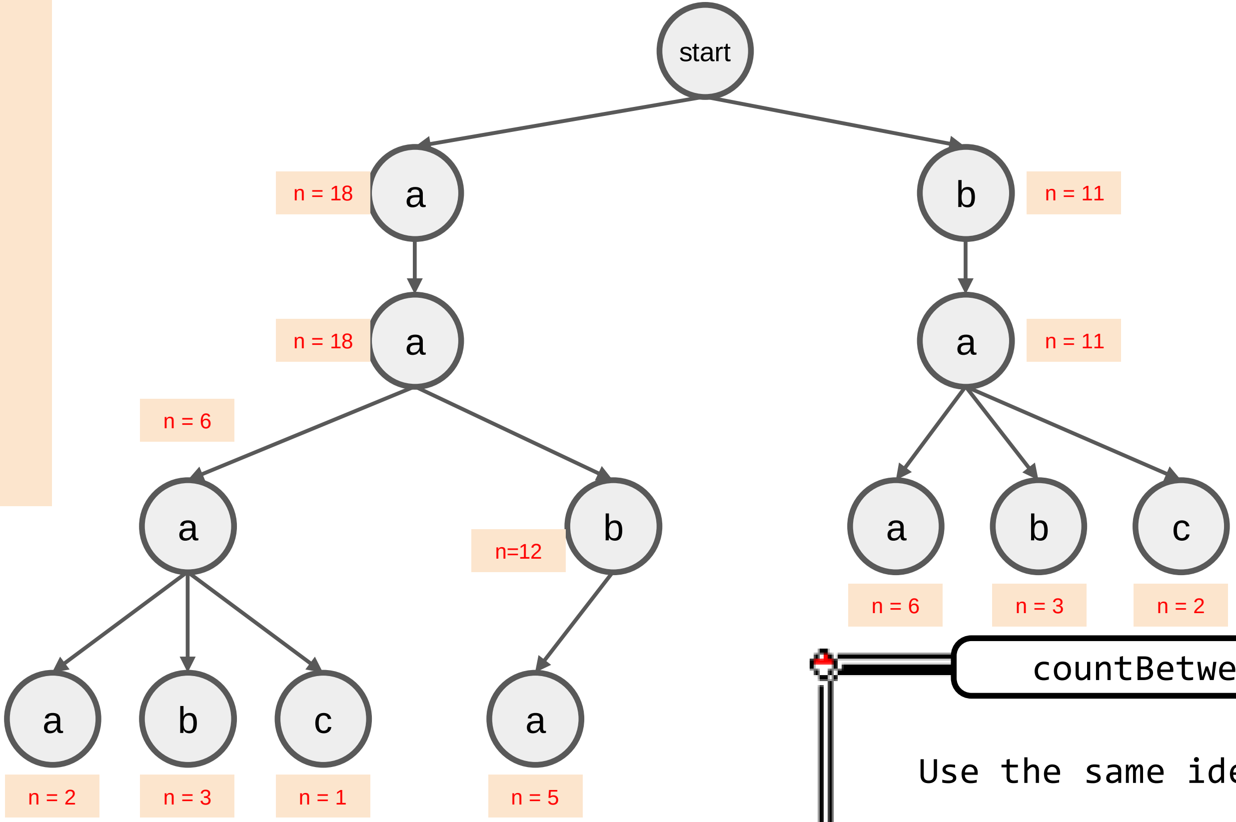
Take ranks!
`rank(377)` covers 9 elements
`rank(13)` covers 3 elements

Ans:
rank(377) - rank(13) - 1



Want to store:

- "aaaa" (2)
- "aaab" (3)
- "aaac" (1)
- "aaba" (5)
- "aab" (7)
- "baa" (6)
- "bab" (3)
- "bac" (2)



countBetween

Use the same idea here!

Problem 4: Bit Tries

Given an array of 32 bits unsigned positive integers, find 2 numbers such that their XOR is maximum.

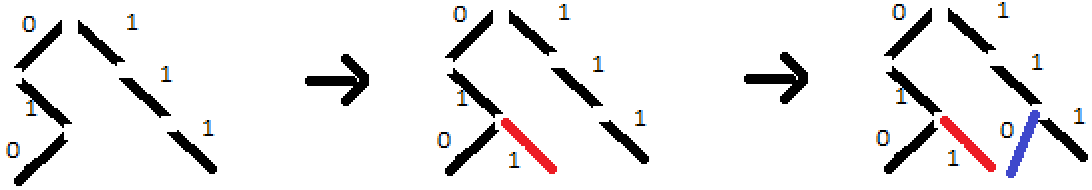
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR TABLE

Problem 4: Bit Tries

Bit tries – where each node is either a 0 or a 1, representing the i-th bit's value where i = depth, counting from the MSB

Adding an integer:



Let's consider 3 bit numbers only. 010 and 111 are already into the trie here.

We are going to insert 011 into the trie.

Since the first two bits, 01 are already there, we just have to add the 3rd bit 1 into the trie.

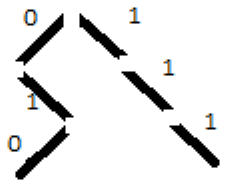
Now suppose we add 110 to the trie.

We add the third bit 0 to the trie.

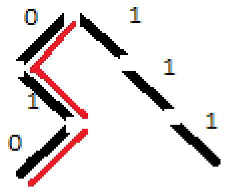
Problem 4: Bit Tries

To find the largest value of an XOR operation, the value of XOR should have every bit to be a set bit i.e 1. In a 32-bit number, the goal is to get the most 1 set starting left to right.

Example with 3 bits.



We have this already existing trie. Now, we want to maximise the xor with the number 100.



First bit is 1. So we'd like to have a 1 after XOR. So we go left side.
So our answer is of the form "1xx".
Next bit is 0. So we go right side. Our answer is now of form "11x".

For 3rd bit a 0 is there.
So we'd like to go right.
But we can't go to right.

Instead we'll continue with left only.

So, our answer is "110".

SORTING JUMBLE (CS2040S 2020 MIDTERM)

Section by Ian Yong

Sorting Jumble

The first column in the table below contains an unsorted list of words. The last column contains a sorted list of words. Each intermediate column contains a partially sorted list.

Each intermediate column was constructed by beginning with the unsorted list at the left and running one of the sorting algorithms that we learned about in class, stopping at some point before it finishes. Each algorithm is executed exactly as described in the lecture notes. One column has been sorted using a sorting algorithm that you have not seen in class. **(Recursive algorithms recurse on the left half of the array before the right half.)**

Identify which column was (partially) sorted with which sorting algorithm.

Bubble
Selection
Insertion
Merge
Quick(first ele pivot)
None

Unsorted	A	B	C	D	E	F	Sorted
Mary	Eddie	Eddie	Alice	Alice	Alice	Eddie	Alice
Harry	Fred	Gina	Bob	Bob	Bob	Gina	Bob
Patty	Gina	Harry	Carol	Carol	Carol	Harry	Carol
Eddie	Harry	Kelly	Eddie	Dave	Dave	Fred	Dave
Gina	Ina	Mary	Gina	Eddie	Eddie	Alice	Eddie
Kelly	Kelly	Patty	Kelly	Fred	Fred	Ina	Fred
Ina	Mary	Ina	Ina	Gina	Ina	Bob	Gina
Fred	Patty	Fred	Fred	Kelly	Harry	Kelly	Harry
Alice	Alice	Alice	Dave	Mary	Gina	Carol	Ina
Noah	Bob	Noah	John	Noah	Kelly	John	John
Bob	Linda	Bob	Harry	Harry	John	Dave	Kelly
Linda	Noah	Linda	Linda	Linda	Ophelia	Linda	Linda
Carol	Carol	Carol	Mary	Patty	Patty	Mary	Mary
John	Dave	John	Noah	John	Noah	Noah	Noah
Dave	John	Dave	Patty	Ina	Mary	Ophelia	Ophelia
Ophelia	Ophelia	Ophelia	Ophelia	Ophelia	Linda	Patty	Patty
Unsorted	**A**	**B**	**C**	**D**	**E**	**F**	**Sorted**

Sorting Jumble

- Do not just execute each sorting algorithm, step-by-step, until it matches one of the columns
  - You will run out of time if you do that
  - Only do so if you’ve already narrowed down to a few possible search algorithms and you cannot make use of any other invariants
- Think in terms of invariants that are true at every step of the algorithm!

Sorting Algorithm	Description	Invariant	Is stable?
Bubble Sort	"Bubble" the largest element to the end of the array through repeated swapping of out-of-order adjacent pairs (inversions)		Yes
Selection Sort	Select the minimum element and add it to the sorted region of the array by swapping. Repeat until all elements have been selected.		No
Insertion Sort	Select the first element in the unsorted region of the array and find where to place it in the sorted region. Repeat until all elements have been selected.		Yes
Merge Sort	Halve the array, recursively sort, then merge	Each subarray is already sorted when merging	Yes
Quick Sort (with first element pivot)	Partition around the first element, then repeat on subarrays	All elements to the left/right of the pivot are smaller/larger	No

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	A	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**A**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	A	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array A is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the largest element (Patty) is not at the end of the array. Thus, sorting algorithm A is not Bubble Sort.

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**A**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**A**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array A is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the smallest element (Alice) is not at the start of the array. Thus, sorting algorithm A is not Selection Sort.

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**A**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	A	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Insertion Sort	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since only the last element (Ophelia) is untouched, at least $n-1$ iterations of the sort was run. However, the first $n-1$ elements of the array A are not sorted. Thus, sorting algorithm A is not Insertion Sort.

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	A	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
~~Insertion Sort~~	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	A	Sorted
Mary	Eddie	Alice
Harry	Fred	Bob
Patty	Gina	Carol
Eddie	Harry	Dave
Gina	Ina	Eddie
Kelly	Kelly	Fred
Ina	Mary	Gina
Fred	Patty	Harry
Alice	Alice	Ina
Noah	Bob	John
Bob	Linda	Kelly
Linda	Noah	Linda
Carol	Carol	Mary
John	Dave	Noah
Dave	John	Ophelia
Ophelia	Ophelia	Patty
Unsorted	A	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
~~Insertion Sort~~	
Merge Sort	Each subarray is already sorted when merging
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since each subarray (when split by powers of 2) is locally sorted, sorting algorithm A is Merge Sort!

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	B	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**B**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	B	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array B is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the largest element (Patty) is not at the end of the array. Thus, sorting algorithm B is not Bubble Sort.

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**B**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	B	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array B is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the smallest element (Alice) is not at the start of the array. Thus, sorting algorithm B is not Selection Sort.

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**B**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	B	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Kelly	Dave
Gina	Mary	Eddie
Kelly	Patty	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Alice	Ina
Noah	Noah	John
Bob	Bob	Kelly
Linda	Linda	Linda
Carol	Carol	Mary
John	John	Noah
Dave	Dave	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**B**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Insertion Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the first 6 elements are sorted and the last 10 elements are untouched, sorting algorithm B is Insertion Sort!

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**C**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	C	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array C is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the largest element (Patty) is not at the end of the array. Thus, sorting algorithm C is not Bubble Sort.

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**C**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	C	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

The first 3 elements of array C are sorted, but not the 4th element. If sorting algorithm C is Selection Sort, it must have been run for **no more than three iterations**. Regardless of whether 1, 2, or 3 iterations were run, Mary should be in Alice's original position in the array (due to swapping). However, this is not the case. Thus, sorting algorithm C is not Selection Sort.

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**C**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**C**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Selection Sort~~	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the first element in the unsorted array (Mary) appears in its sorted location in array C and the elements to the left/right of Mary are smaller/larger, sorting algorithm C is possibly Quick Sort. **However, we cannot say for sure as array F has the same properties!** We can either try to execute Quick Sort step-by-step (not recommended), or try to figure out if array F's identity can be determined to find out the identity of array C through elimination.

Unsorted	D	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Gina	Gina
Fred	Kelly	Harry
Alice	Mary	Ina
Noah	Noah	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Patty	Mary
John	John	Noah
Dave	Ina	Ophelia
Ophelia	Ophelia	Patty
Unsorted	D	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	D	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Gina	Gina
Fred	Kelly	Harry
Alice	Mary	Ina
Noah	Noah	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Patty	Mary
John	John	Noah
Dave	Ina	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**D**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	D	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Gina	Gina
Fred	Kelly	Harry
Alice	Mary	Ina
Noah	Noah	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Patty	Mary
John	John	Noah
Dave	Ina	Ophelia
Ophelia	Ophelia	Patty
Unsorted	D	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array D is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the largest element (Patty) is not at the end of the array. Thus, sorting algorithm D is not Bubble Sort.

Unsorted	D	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Gina	Gina
Fred	Kelly	Harry
Alice	Mary	Ina
Noah	Noah	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Patty	Mary
John	John	Noah
Dave	Ina	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**D**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	D	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Gina	Gina
Fred	Kelly	Harry
Alice	Mary	Ina
Noah	Noah	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Patty	Mary
John	John	Noah
Dave	Ina	Ophelia
Ophelia	Ophelia	Patty
Unsorted	**D**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Selection Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

The smallest k elements are at the start of array D. However, this is also the case for array E. As such, we have no choice but to execute Selection Sort. D:

Unsorted	D	E	Sorted
Mary	Alice	Alice	Alice
Harry	Bob	Bob	Bob
Patty	Carol	Carol	Carol
Eddie	Dave	Dave	Dave
Gina	Eddie	Eddie	Eddie
Kelly	Fred	Fred	Fred
Ina	Gina	Ina	Gina
Fred	Kelly	Harry	Harry
Alice	Mary	Gina	Ina
Noah	Noah	Kelly	John
Bob	Harry	John	Kelly
Linda	Linda	Ophelia	Linda
Carol	Patty	Patty	Mary
John	John	Noah	Noah
Dave	Ina	Mary	Ophelia
Ophelia	Ophelia	Linda	Patty
Unsorted	D	E	Sorted

Selection sort

- Select the minimum element
- Add it to the sorted region of the array by swapping
- Repeat until all elements have been selected

Unsorted	D	E	Sorted
Mary	Alice	Alice	Alice
Harry	Bob	Bob	Bob
Patty	Carol	Carol	Carol
Eddie	Dave	Dave	Dave
Gina	Eddie	Eddie	Eddie
Kelly	Fred	Fred	Fred
Ina	Gina	Ina	Gina
Fred	Kelly	Harry	Harry
$\boxed{\text{Alice}}$	Mary	Gina	Ina
Noah	Noah	Kelly	John
Bob	Harry	John	Kelly
Linda	Linda	Ophelia	Linda
Carol	Patty	Patty	Mary
John	John	Noah	Noah
Dave	Ina	Mary	Ophelia
Ophelia	Ophelia	Linda	Patty
Unsorted	D	E	Sorted

Selection sort

- Select the minimum element
- Add it to the sorted region of the array by swapping
- Repeat until all elements have been selected

Unsorted	D	E	Sorted
Mary	Alice	Alice	Alice
Harry	Bob	Bob	Bob
Patty	Carol	Carol	Carol
Eddie	Dave	Dave	Dave
Gina	Eddie	Eddie	Eddie
Kelly	Fred	Fred	Fred
Ina	Gina	Ina	Gina
Fred	Kelly	Harry	Harry
Alice	Mary	Gina	Ina
Noah	Noah	Kelly	John
Bob	Harry	John	Kelly
Linda	Linda	Ophelia	Linda
Carol	Patty	Patty	Mary
John	John	Noah	Noah
Dave	Ina	Mary	Ophelia
Ophelia	Ophelia	Linda	Patty
Unsorted	D	E	Sorted

Selection sort

- Select the minimum element
- Add it to the sorted region of the array by swapping
- Repeat until all elements have been selected

Unsorted	D	E	Sorted
Mary	Alice	Alice	Alice
Harry	Bob	Bob	Bob
Patty	Carol	Carol	Carol
Eddie	Dave	Dave	Dave
Gina	Eddie	Eddie	Eddie
Kelly	Fred	Fred	Fred
Ina	Gina	Ina	Gina
Fred	Kelly	Harry	Harry
Alice	Mary	Gina	Ina
Noah	Noah	Kelly	John
Bob	Harry	John	Kelly
Linda	Linda	Ophelia	Linda
Carol	Patty	Patty	Mary
John	John	Noah	Noah
Dave	Ina	Mary	Ophelia
Ophelia	Ophelia	Linda	Patty
Unsorted	**D**	**E**	**Sorted**

Selection sort

- Select the minimum element
- Add it to the sorted region of the array by swapping
- Repeat until all elements have been selected

When executing Selection Sort, the position of Mary in array E is wrong. Thus, sorting algorithm D is Selection Sort!

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	E	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	**E**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	E	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the ordering of elements in array E is different from the unsorted array, we know that **at least one** iteration of the sort was run. However, the largest element (Patty) is not at the end of the array. Thus, sorting algorithm E is not Bubble Sort.

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	**E**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	**E**	**Sorted**

Sorting Algorithm	Invariant
~~Bubble Sort~~	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the position of the pivot element Mary is incorrect, sorting algorithm E cannot be Quick Sort.

Unsorted	E	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Dave	Dave
Gina	Eddie	Eddie
Kelly	Fred	Fred
Ina	Ina	Gina
Fred	Harry	Harry
Alice	Gina	Ina
Noah	Kelly	John
Bob	John	Kelly
Linda	Ophelia	Linda
Carol	Patty	Mary
John	Noah	Noah
Dave	Mary	Ophelia
Ophelia	Linda	Patty
Unsorted	E	Sorted

Sorting Algorithm	Invariant
~~Bubble Sort~~	
~~Quick Sort (with first element pivot)~~	~~All elements to the left/right of the pivot are smaller/larger~~

By elimination, sorting algorithm E is none of the 5 sorting algorithms!

Unsorted	F	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Fred	Dave
Gina	Alice	Eddie
Kelly	Ina	Fred
Ina	Bob	Gina
Fred	Kelly	Harry
Alice	Carol	Ina
Noah	John	John
Bob	Dave	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Ophelia	Ophelia
Ophelia	Patty	Patty
Unsorted	F	Sorted

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	F	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Fred	Dave
Gina	Alice	Eddie
Kelly	Ina	Fred
Ina	Bob	Gina
Fred	Kelly	Harry
Alice	Carol	Ina
Noah	John	John
Bob	Dave	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Ophelia	Ophelia
Ophelia	Patty	Patty
Unsorted	**F**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Unsorted	F	Sorted
Mary	Eddie	Alice
Harry	Gina	Bob
Patty	Harry	Carol
Eddie	Fred	Dave
Gina	Alice	Eddie
Kelly	Ina	Fred
Ina	Bob	Gina
Fred	Kelly	Harry
Alice	Carol	Ina
Noah	John	John
Bob	Dave	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Ophelia	Ophelia
Ophelia	Patty	Patty
Unsorted	**F**	**Sorted**

Sorting Algorithm	Invariant
Bubble Sort	
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

Since the largest k elements are at the end of array F (and no other arrays have Patty at the end), sorting algorithm F is Bubble Sort!

Unsorted	C	Sorted
Mary	Alice	Alice
Harry	Bob	Bob
Patty	Carol	Carol
Eddie	Eddie	Dave
Gina	Gina	Eddie
Kelly	Kelly	Fred
Ina	Ina	Gina
Fred	Fred	Harry
Alice	Dave	Ina
Noah	John	John
Bob	Harry	Kelly
Linda	Linda	Linda
Carol	Mary	Mary
John	Noah	Noah
Dave	Patty	Ophelia
Ophelia	Ophelia	Patty
Unsorted	C	Sorted

Sorting Algorithm	Invariant
Quick Sort (with first element pivot)	All elements to the left/right of the pivot are smaller/larger

By the process of elimination, sorting algorithm C is Quick Sort!

ALGORITHM ANALYSIS (CS2040S 2020 MIDTERM)

Algorithm Analysis

For each of the following, choose the best (tightest) asymptotic function from among the given options. Some of the following may appear more than once, and some may appear not at all. Write the letter indicating the proper bound in the answer box.

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

Algorithm Analysis

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

- $$T(n) = \left(\frac{\sqrt{n}}{17}\right)\left(\frac{\sqrt{n}}{4}\right) + \frac{n \log n}{1000}$$

Algorithm Analysis

- $$\begin{aligned} T(n) &= \left(\frac{\sqrt{n}}{17}\right)\left(\frac{\sqrt{n}}{4}\right) + \frac{n \log n}{1000} \\ &= \frac{n}{68} + \frac{n \log n}{1000} \\ &= \Theta(n) + \Theta(n \log n) \\ &= \Theta(n \log n) \end{aligned}$$

Algorithm Analysis

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

- $$T(n) = (2^n)(2^n)$$

Algorithm Analysis

- $$\begin{aligned} T(n) &= (2^n)(2^n) \\ &= 2^{2n} \\ &= \Theta(2^{2n}) \end{aligned}$$

Algorithm Analysis

- $$\begin{aligned} T(n) &= (2^n)(2^n) \\ &= 2^{2n} \\ &= \Theta(2^{2n}) \end{aligned}$$

Note: $\Theta(2^{2n}) \neq \Theta(2^n)$

The constant in the exponent matters! $\Theta(2^{2n}) = \Theta((2^n)^2)$

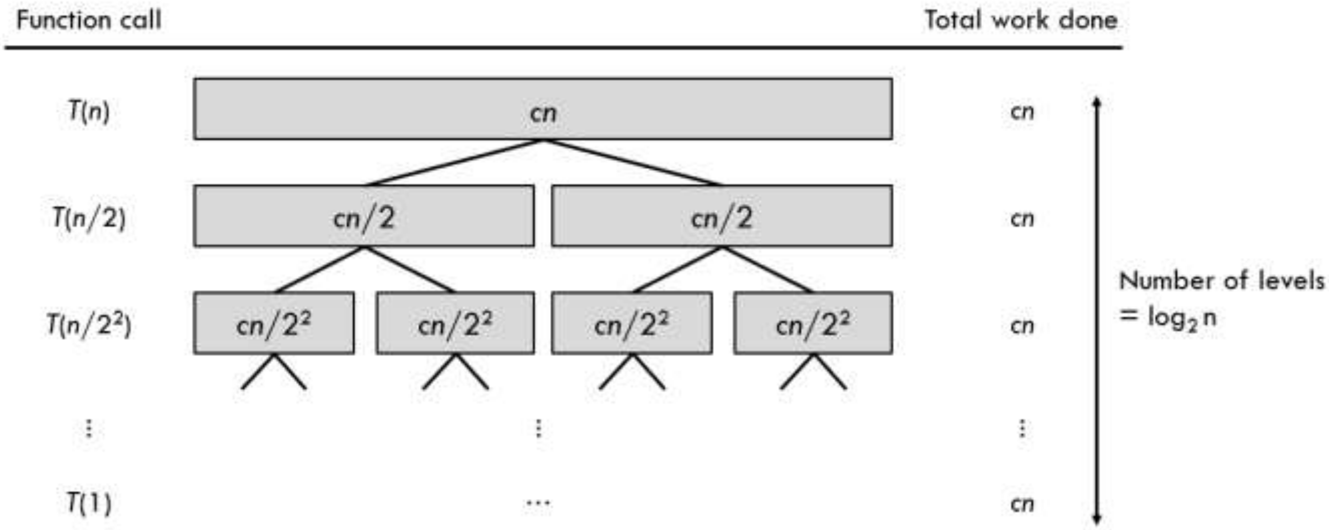
Algorithm Analysis

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

- $$T(n) = 2T\left(\frac{n}{2}\right) + n$$
  $$T(1) = 1$$

Algorithm Analysis

- $$T(n) = 2T\left(\frac{n}{2}\right) + n$$
  $$T(1) = 1$$



Algorithm Analysis

- $$T(n) = 2T\left(\frac{n}{2}\right) + n$$
  $$T(1) = 1$$

- Given that
  - the total amount of work done in each level sums up to cn, and that
  - the height of the tree is $h = \log_2 n$,
- we can then calculate the total amount of work done across all levels by multiplying the total amount of work done by the height of the tree.

$$cn \log_2 n = \Theta(n \log n)$$

Algorithm Analysis

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

```
public static int loopy(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.println("Hello.");
        }
    }
}
```

Algorithm Analysis

- During the i-th iteration of the outer for loop, the inner for loop runs for i iterations. If we add up all the iterations, we get:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1 \\
&= \sum_{i=0}^{n-1} i \\
&= 0 + 1 + 2 + 3 + \cdots + (n-1) \\
&= \frac{n(n-1)}{2} \\
&= \Theta(n^2)
\end{aligned}
$$

Algorithm Analysis

A. $\Theta(1)$	B. $\Theta(\log n)$	C. $\Theta(n)$	D. $\Theta(n \log n)$
E. $\Theta(n^2)$	F. $\Theta(n^3)$	G. $\Theta(2^n)$	H. None of the above.

```
public static int recursiveloopy(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println(“Hello.”);
        }
    }

    if (n <= 2) {
        return 1;
    } else if (n % 2 == 0) {
        return recursiveloopy(n + 1);
    } else {
        return recursiveloopy(n – 2);
    }
}
```

Algorithm Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(“Hello.”);
    }
}
```

Algorithm Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println("Hello.");
    }
}
```

- During the i-th iteration of the outer for loop, the inner for loop runs for n iterations. If we add up all the iterations, we get:

$$\begin{aligned} T(n) &= n(n) \\ &= n^2 \\ &= \Theta(n^2) \end{aligned}$$

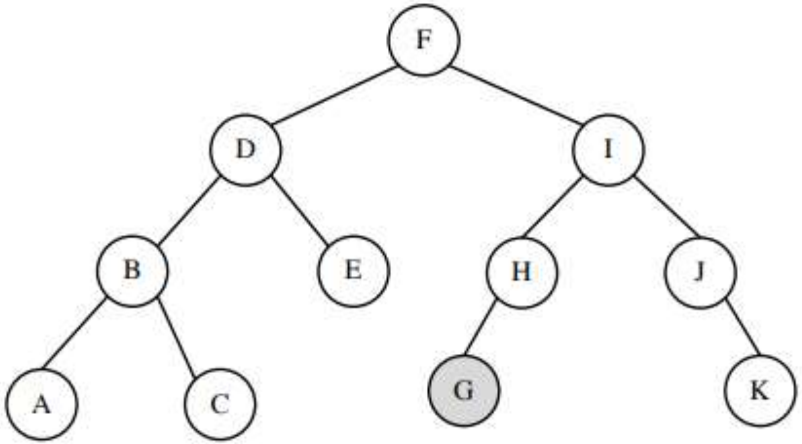
Algorithm Analysis

```
if (n <= 2) {
    return 1;
} else if (n % 2 == 0) {
    return recursiveloopy(n + 1);
} else {
    return recursiveloopy(n – 2);
}
```

Algorithm Analysis

```
if (n <= 2) {
    return 1;
} else if (n % 2 == 0) {
    return recursiveloopy(n + 1);
} else {
    return recursiveloopy(n – 2);
}
```

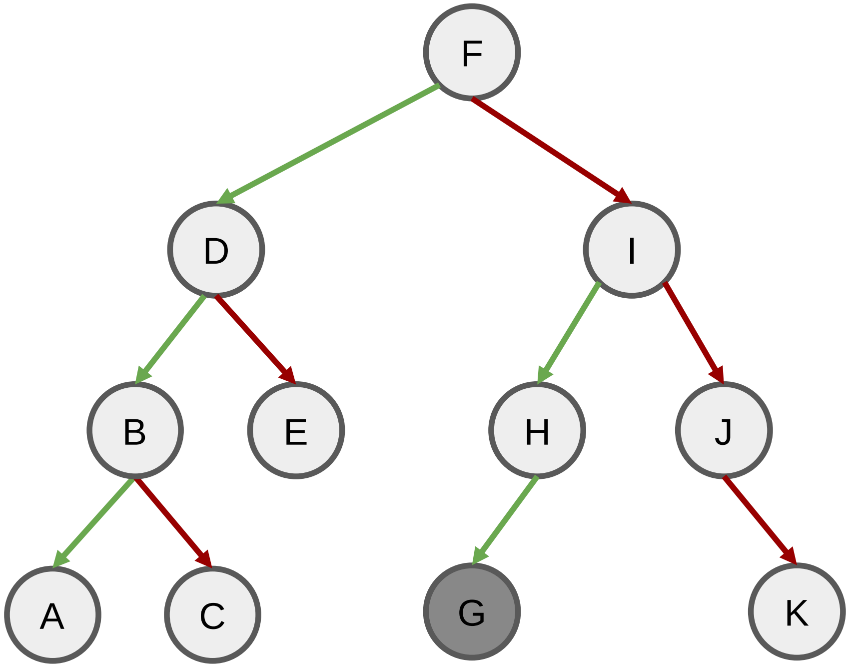
- `recursiveloopy` is called $\Theta(n)$ times
- Overall, we get $\Theta(n) \times \Theta(n^2) = \Theta(n^3)$



The previous operation inserted the node G, which then triggered a double-rotation. Which node was the grandparent of G (i.e., the parent of the parent of G) immediately after G was inserted and before the rotations were performed?

1. D
2. E
3. F
4. H
5. I
6. J.

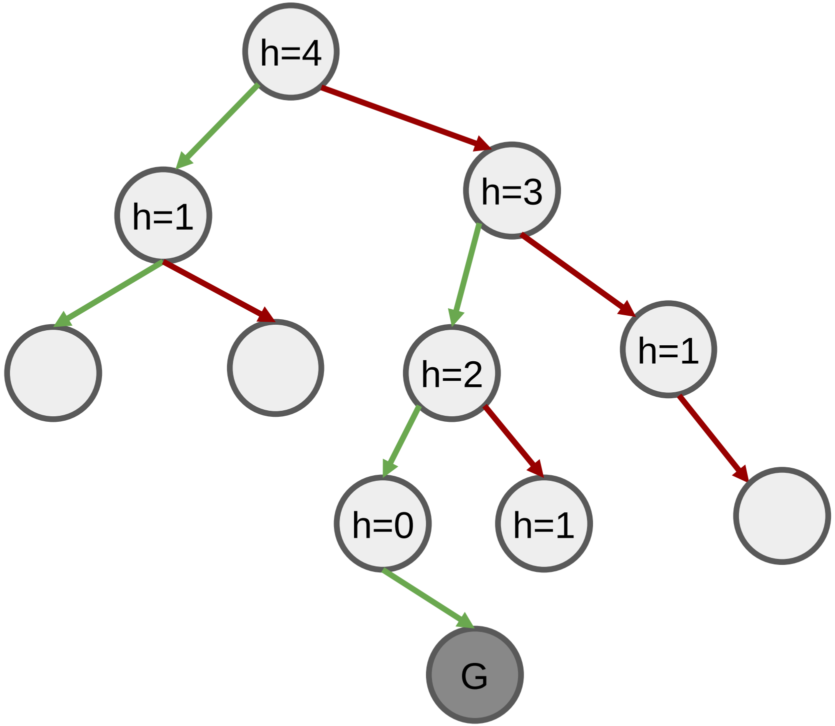
2021 5D



Okay, so G made the right subtree of the grandparent left heavy

(great grandparent of G is F → F is parent of G)

2021 5D



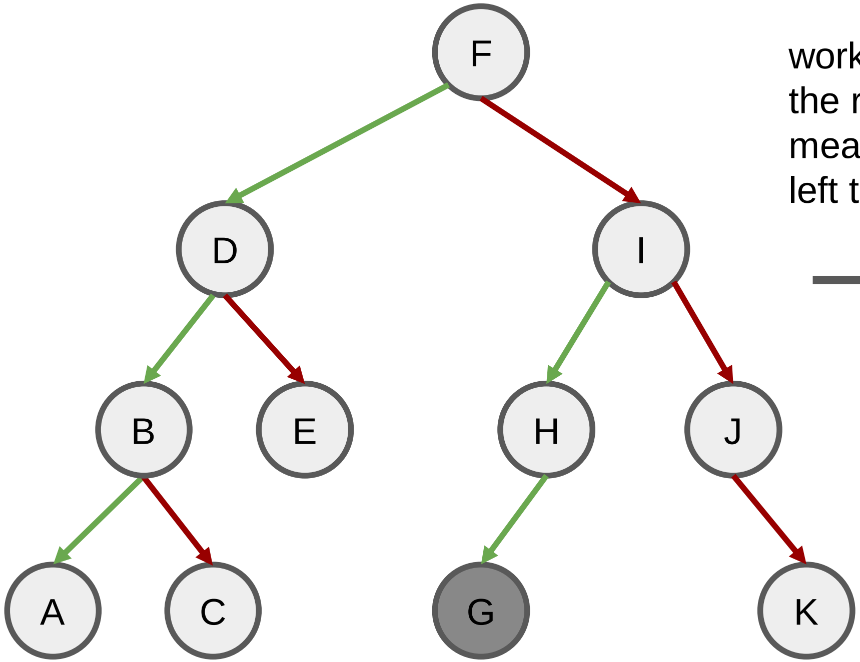
Okay, so G made the right subtree of the grandparent left heavy

-> some shape like this?

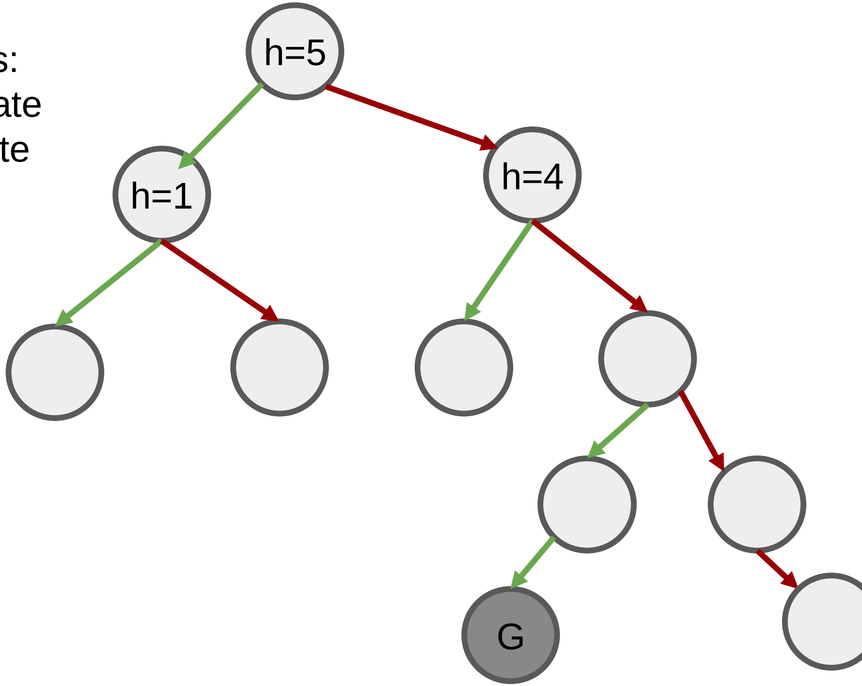
-> via a right rotate and a left rotate.

Let's just try to get this shape by picking a random place to reverse-left rotate and then reverse-right rotate

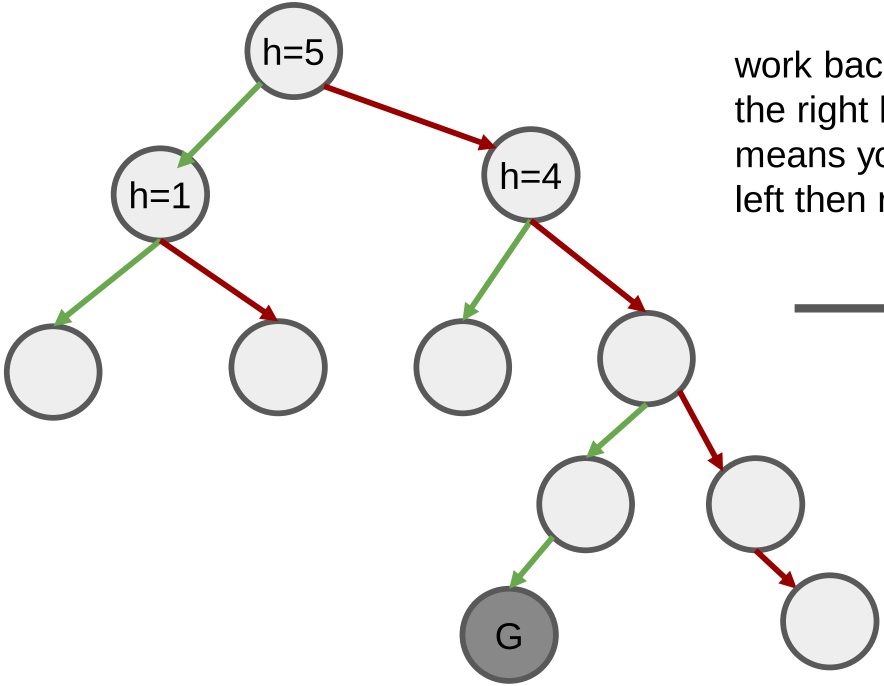
2021 5D



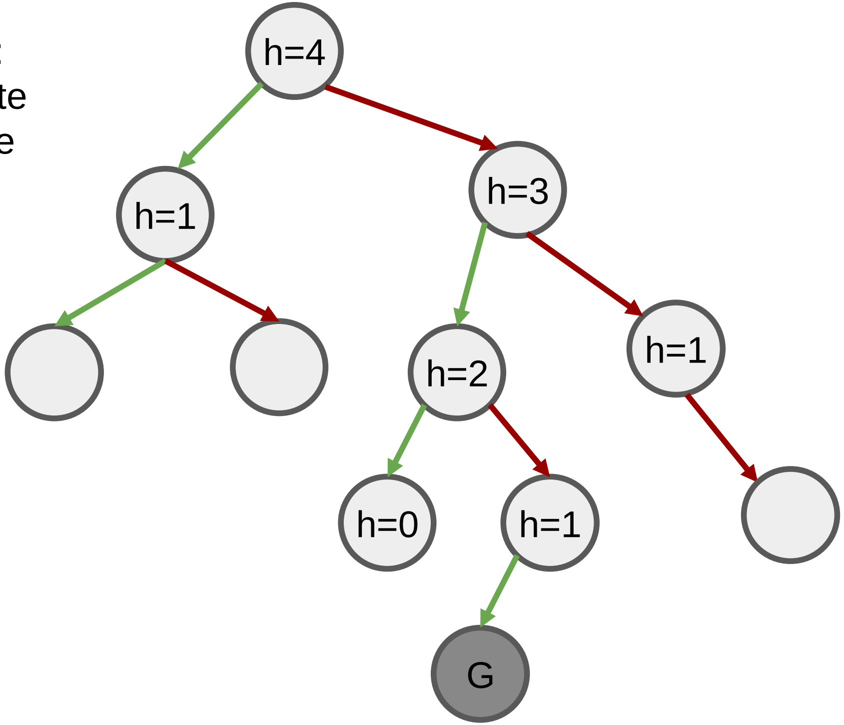
work backwards:
the right left rotate
means you rotate
left then right



2021 5D



work backwards:
the right left rotate
means you rotate
left then right



MIDTERM SORTING (CS2040S 2021 MIDTERM)

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

Assume we are using `midtermSort(A, k)` to sort an array `A` of unique elements where each item is in an array position at a distance $\leq k$ from its array position in the sorted array. For example, in the following array:

4	5	6	1	2	3	10	11	12	7	8	9

Notice that each element is within distance $k = 3$ of its final position. The value in $A[4] = 2$ belongs in position $A[1]$, and $4 - 1 \leq 3$. Assume $k \geq 2$.

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

midterm sort idea: splits the array into chunks of k elements

In each loop, sort two of the chunks.

since the items are at most k distance away from their true position (ie worst case they are in the previous chunk or the next chunk), after sorting each section twice (with the previous chunk, and the next chunk) they will get into the right place

Assume we are using `midtermSort(A, k)` to sort an array **A** of unique elements where each item is in an array position at a distance $\leq k$ from its array position in the sorted array. For example, in the following array:

4	5	6	1	2	3	10	11	12	7	8	9

Notice that each element is within distance $k = 3$ of its final position. The value in $A[4] = 2$ belongs in position $A[1]$, and $4 - 1 \leq 3$. Assume $k \geq 2$.

Midterm Sorting

A. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ is sorted.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ is sorted.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

Midterm Sorting

A. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ is sorted.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ is sorted.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

Midterm Sorting

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

A. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ is sorted.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ is sorted.

1. Statement I.
2. Statement II.
3. $\boxed{\text{Both Statements I and II.}}$
4. Neither statement is true.

[2 marks]

Because each item in the array is at an array position $\leq k$ from its array position in the sorted array, after each iteration of the loop, the elements in A[0, (j+1)k-1] are the same as the sorted array.

Midterm Sorting

B. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ contains the $(j+1)k-1$ smallest elements in the array.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ contains the $(j+2)k-1$ smallest elements in the array.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

Midterm Sorting

B. Which of the following is always true:

I. After each iteration of the loop, the array prefix $A[0,(j+1)k-1]$ contains the $(j+1)k-1$ smallest elements in the array.

II. After each iteration of the loop, the array prefix $A[0,(j+2)k-1]$ contains the $(j+2)k-1$ smallest elements in the array.

1. Statement I.
2. Statement II.
3. Both Statements I and II.
4. Neither statement is true.

[2 marks]

Midterm Sorting

C. If an element $A[i]$ is initially in position i and ends in position $i_\ell \leq i$ (when the sorting algorithm completes), then at the end of every iteration of the loop it is never moved to a position $i_h > i$: True or False? [2 marks]

Midterm Sorting

C. If an element $A[i]$ is initially in position i and ends in position $i_\ell \leq i$ (when the sorting algorithm completes), then at the end of every iteration of the loop it is never moved to a position $i_h > i$: True or False? [2 marks]

Midterm Sorting

C. If an element $A[i]$ is initially in position i and ends in position $i_\ell \leq i$ (when the sorting algorithm completes), then at the end of every iteration of the loop it is never moved to a position $i_h > i$: True or False? [2 marks]

- Let us look at the first iteration that overlaps A[i]
  - A[i] must be in the second half of the subset of the array.
  - Case 1: A[i] ends up in the first half of the subset of the array that is being sorted
    - Then, its position is fixed after just 1 iteration and the statement is true
  - Case 2: A[i] ends up in the second half of the subset of the array that is being sorted
    - Then, in order for $i_l \leq i$ and $i_h > i$, the number of elements smaller than A[i] must be smaller in the second iteration than in the first iteration
    - This is impossible as the number of elements smaller than A[i] can only increase in the second iteration
    - Thus, the statement is true

Midterm Sorting

D. Assume we run `superMidtermSort(A)` to sort an array `A` of unique elements where each item is in an array position at at distance $\leq k$ from its array position in the sorted array. (In this case, note that k is not given to the algorithm.) What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\Theta(nk)$.
8. None of the above.

Midterm Sorting

D. Assume we run `superMidtermSort(A)` to sort an array `A` of unique elements where each item is in an array position at at distance $\leq k$ from its array position in the sorted array. (In this case, note that k is not given to the algorithm.) What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\Theta(nk)$.
8. None of the above.

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

D. Assume we run s … elements where each item is in an array pos … e sorted array. (In this case, note that k is not … of the algorithm as a function of n and k? G … [3 marks]

1. $\Theta(k)$ … (nk).
2. $\Theta(k \log k)$ … one of the above.
3. $\boxed{\Theta(n \log k)}$

Midterm S

D. Assume we run s elements where each item is in an array pos e sorted array. (In this case, note that k is not of the algorithm as a function of n and k? G [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

(nk).

one of the above.

$O\left(\frac{n}{k}\right)$

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Midterm S

$O\left(\frac{n}{k}\right)$

D. Assume we run s… elements where each item is in an array pos… this case, note that k is not… as a function of n and k? G… rks]

$O(2k\log 2k)$
$= O(k\log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k\log 2 + k\log k)$
$= O(k + k\log k)$
$= O(k\log k)$

1. $\Theta(k)$
2. $\Theta(k\log k)$
3. $\boxed{\Theta(n\log k)}$

…one of the above.

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

Midterm S

D. Assume we run s… elements where each item is in an array pos… this case, note that k is not… as a function of n and k? G… rks]

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O(k(\log 2 + \log k))$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

…one of the above.

Midterm S

D. Assume we run s
item is in an array pos
case, note that k is not
function of n and k? G

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

elements where each
this
as a
rks]

one of the above.

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n \log d$

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

D. Assume we run s ... elements where each item is in an array pos ... this case, note that k is not ... as a function of n and k? G ... rks]

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n \log d$

$O(n)$

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

D. Assume we run s… elements where each item is in an array pos… this case, note that k is not… as a function of n and k? G… rks]

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

…one of the above.

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n \log d$

$O(n)$

$O(\log \log n)$

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

D. Assume we run s… elements where each item is in an array pos… this case, note that k is not… as a function of n and k? G… rks]

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n \log d$

$O(n)$

$\cancel{O(\log \log n)}$

However, d is actually bounded by k as once $d > k$, midtermSort(A, d) sorts the array and isSorted(A) returns true

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n \log k)$

$O\left(\frac{n}{k}\right)$

$O(2k \log 2k)$
$= O(k \log 2k)$
$= O\big(k(\log 2 + \log k)\big)$
$= O(k \log 2 + k \log k)$
$= O(k + k \log k)$
$= O(k \log k)$

D. Assume we run s… elements where each item is in an array pos… this case, note that k is not… as a function of n and k? G… rks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\boxed{\Theta(n \log k)}$

…one of the above.

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n \log d$

$O(n)$

$O(\log \log k)$ $\cancel{O(\log \log n)}$

However, d is actually bounded by k as once $d > k$, midtermSort(A, d) sorts the array and isSorted(A) returns true

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(n\log k)$

Midterm S

$O\left(\frac{n}{k}\right)$

D. Assume we run s ... elements where each ... this ... as a ... rks]

$O(2k\log 2k)$
$= O(k\log 2k)$
$= O(k(\log 2 + \log k))$
$= O(k\log 2 + k\log k)$
$= O(k + k\log k)$
$= O(k\log k)$

$$O(n\log 2^{2^0}) + O(n\log 2^{2^1}) + O(n\log 2^{2^2}) + \cdots$$
$$+ O\left(n\log 2^{2^{\log\log k}}\right) = O\left(n(2^0 + 2^1 + 2^2 + \cdots + 2^{\log\log k})\right)$$
$$= O\left(n\sum_{i=0}^{\log\log k} 2^i\right) = O\left(n\left(\frac{2^{\log\log k+1} - 1}{2-1}\right)\right)$$
$$= O(n(2\log k - 1)) = O(n\log k)$$

one of the above.

3. $\boxed{\Theta(n\log k)}$

```
midtermSort(A, n, d)
    if isSorted(A) return
    j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(n\log d$

$O(n)$

$O(\log\log k)$ ~~$O(\log\log n)$~~

However, d is actually bounded by k as once $d > k$, midtermSort(A, d) sorts the array and isSorted(A) returns true

Midterm Sorting

E. Assume we run `superMidtermSort(A)` to sort an array `A` with the possibility of repeated elements. Is the resulting sorting algorithm stable? [2 marks]

Midterm Sorting

E. Assume we run `superMidtermSort(A)` to sort an array `A` with the possibility of repeated elements. Is the resulting sorting algorithm stable? [2 marks]

True, because merge sort is stable

Midterm Sorting

F. Is the `superMidtermSort(A)` an in-place sorting algorithm? [2 marks]

Midterm Sorting

F. Is the `superMidtermSort(A)` an in-place sorting algorithm? [2 marks]

False, because merge sort (as defined in class) is not in-place

Midterm Sorting

G. Assume instead we run `InsertionSort(A)` to sort the array `A` of unique elements where each item is in an array position at at distance $\leq k$ from its array position in the sorted array. What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\Theta(nk)$.
8. None of the above.

Midterm Sorting

G. Assume instead we run `InsertionSort(A)` to sort the array `A` of unique elements where each item is in an array position at at distance $\leq k$ from its array position in the sorted array. What is the running time of the algorithm as a function of n and k? Give the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$
4. $\Theta(k \log n)$
5. $\Theta(n \log^2 k)$
6. $\Theta(n^2)$
7. $\boxed{\Theta(nk).}$
8. None of the above.

Midterm S

G. Assume instead v nique elements where
each item is in an arra on in the sorted array.
What is the running ti ive the tightest bound
possible. [3 marks]

1. $\Theta(k)$ $\boxed{(nk).}$
2. $\Theta(k \log k)$ one of the above.
3. $\Theta(n \log k)$

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

Midterm S

$O\left(\frac{n}{k}\right)$

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)

superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

G. Assume instead v … nique elements where each item is in an arra … on in the sorted array. What is the running ti … ive the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$

… (nk).

… one of the above.

Midterm S

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

$O\left(\frac{n}{k}\right)$

$O((2k)^2)$
$= O(4k^2)$
$= O(k^2)$

G. Assume instead v … r … here
each item is in an arra … o … array.
What is the running ti … ive the tightest bound
possible. [3 marks]

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$

(nk).

one of the above.

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(nk)$

$O\left(\frac{n}{k}\right)$

$O((2k)^2)$
$= O(4k^2)$
$= O(k^2)$

Midterm S

G. Assume instead w... here each item is in an arra... array. What is the running ti... ive the tightest bound possible. [3 marks]

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$

$\boxed{(nk).}$

one of the above.

Midterm S

G. Assume instead v ... here each item is in an arra ... array. What is the running ti ... ive the tightest bound possible. [3 marks]

1. $\Theta(k)$
2. $\Theta(k \log k)$
3. $\Theta(n \log k)$

... (nk).

... one of the above.

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(nk)$

$O\left(\frac{n}{k}\right)$

$O((2k)^2)$
$= O(4k^2)$
$= O(k^2)$

```
superMidtermSort(int[] A)
    int d = 2
    int j = 0
    int n = A.length
    repeat
        d = 2^{2^j}
        midtermSort(A, n, d)
        if isSorted(A) return
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```

$O(nd)$

```
midtermSort(int[] A, int k)
    int n = A.length;
    for (int j = 0; j<=n/k-1; j++)
        MergeSort(A, jk, (j+2)k-1)
```

Overall $O(nk)$

Midterm S

$O\left(\frac{n}{k}\right)$

$O((2k)^2)$
$= O(4k^2)$
$= O(k^2)$

G. Assume instead v ... ɾ ... 'here ... ɔ ... ırray. ... ive the tightest bound [3 marks]

$$O(2^{2^0}n) + O(2^{2^1}n) + O(2^{2^2}n) + \cdots + O\left(2^{2^{\log\log k}}n\right)$$
$$= O\left(n\left(2^{2^0} + 2^{2^1} + 2^{2^2} + \cdots + 2^{2^{\log\log k}}\right)\right)$$
$$\le O\left(n(2^1 + 2^2 + 2^3 + 2^{\log k})\right) = O\left(n\sum_{i=0}^{\log k} 2^i - 1\right)$$
$$= O\left(n\left(\frac{2^{\log k+1} - 1}{2-1}\right)\right) = O(n(2k-1)) = O(nk)$$

)

$\boxed{(nk).}$

ɔne of the above.

```
, d)
eturn
        j = j + 1
    until d > n
    MergeSort(A, 0, n-1)
```