CSE100: Design and Analysis of Algorithms
Lecture 13 – Sorting Lower Bounds (wrap up) and Binary Search Trees

Mar 03rd 2022

O(n)-time sorting, Binary Search Trees and Red-Black Trees

Sorting Lower Bound (review)

- Theorem:
  - Any deterministic comparison-based sorting algorithm must take $\Omega(n \log(n))$ steps.

- Theorem:
  - Any randomized comparison-based sorting algorithm must take $\Omega(n \log(n))$ steps in expectation.

BucketSort (review)

Original array:

21	345	13	101	50	234	1

Next array is sorted by the first digit.

50	**21**	**101**	**1**	**13**	**234**	**345**

Next array is sorted by the first two digits.

101	**01**	**13**	**21**	**234**	**345**	**50**

Next array is sorted by all three digits.

001	**013**	**021**	**050**	**101**	**234**	**345**

Sorted array

To prove this is correct…

- What is the inductive hypothesis?

Original array:

21	345	13	101	50	234	1

Next array is sorted by the first digit.

50	21	101	1	13	234	345

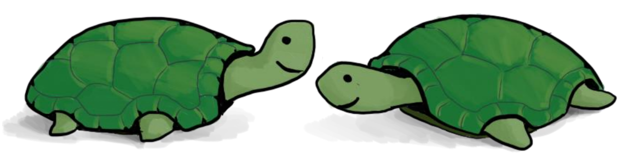
Next array is sorted by the first two digits.

101	01	13	21	234	345	50

Next array is sorted by all three digits.

001	**013**	**021**	**050**	**101**	**234**	**345**

Sorted array



Think-Pair-Share Terrapins

RadixSort is correct

- Inductive hypothesis:
  - After the k'th iteration, the array is sorted by the first k least-significant digits.
- Base case:
  - "Sorted by 0 least-significant digits" means not sorted, so the IH holds for k=0.
- Inductive step:
  - TO DO
- Conclusion:
  - The inductive hypothesis holds for all k, so after the last iteration, the array is sorted by all the digits. Hence, it's sorted!

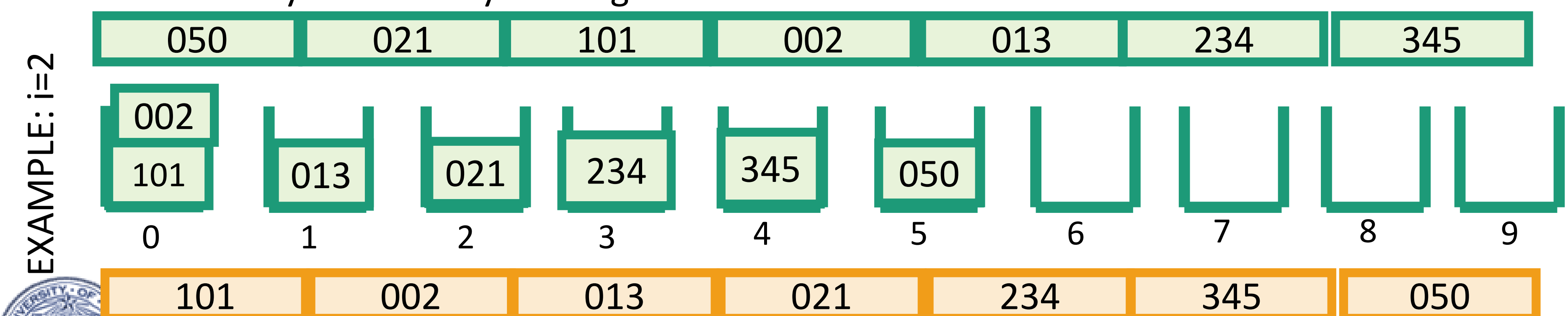
Inductive step

Inductive hypothesis:

After the k'th iteration, the array is sorted by the first k least-significant digits.

- Need to show: if IH holds for k=i-1, then it holds for k=i.
  - Suppose that after the i-1'st iteration, the array is sorted by the first i-1 least-significant digits.
  - Need to show that after the i'th iteration, the array is sorted by the first i least-significant digits.

IH: this array is sorted by first digit.



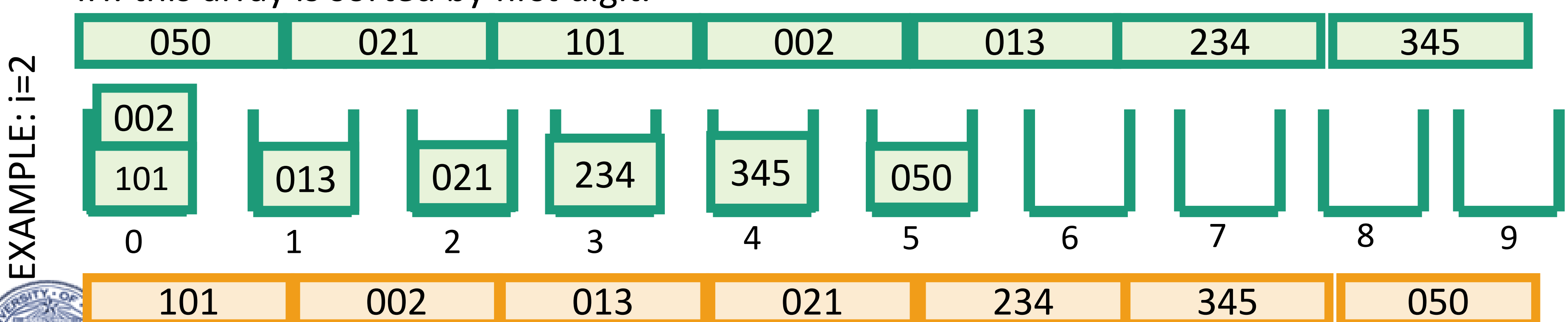
Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

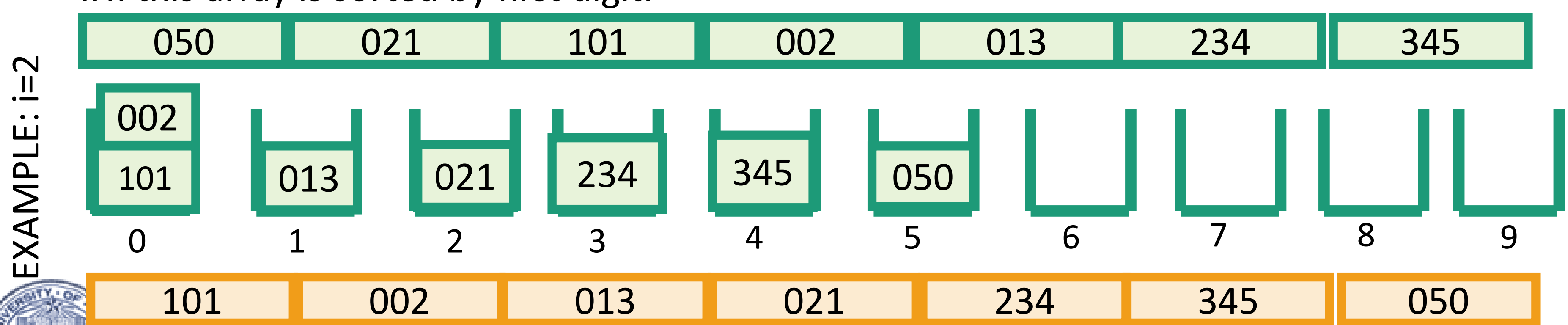
Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

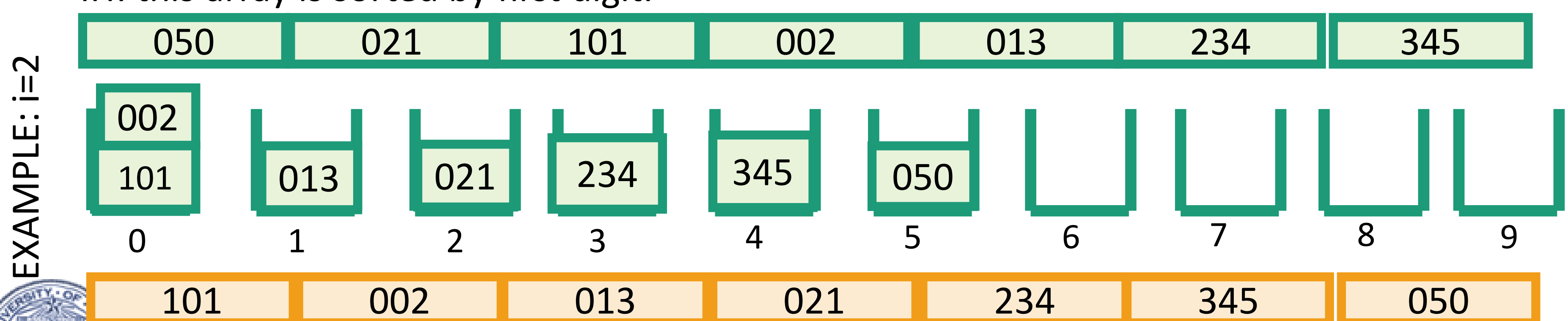
Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

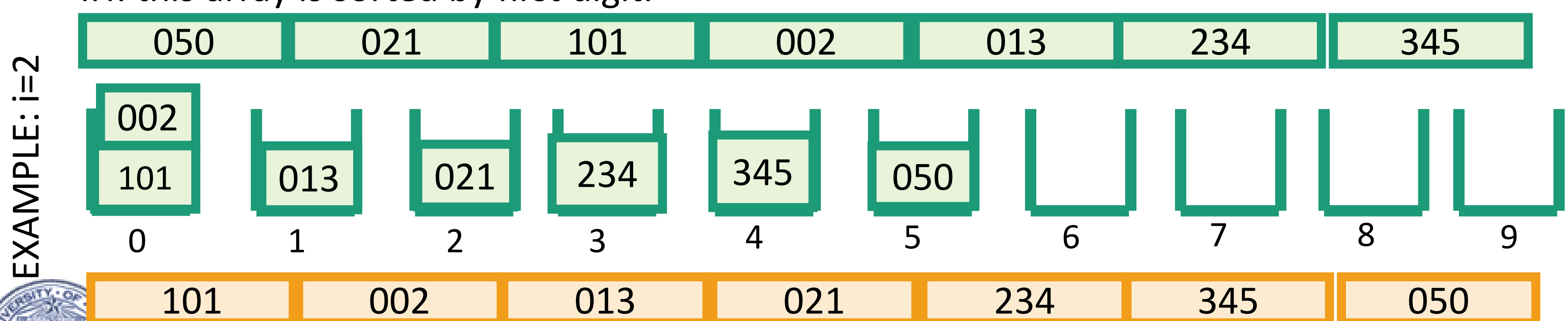
Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

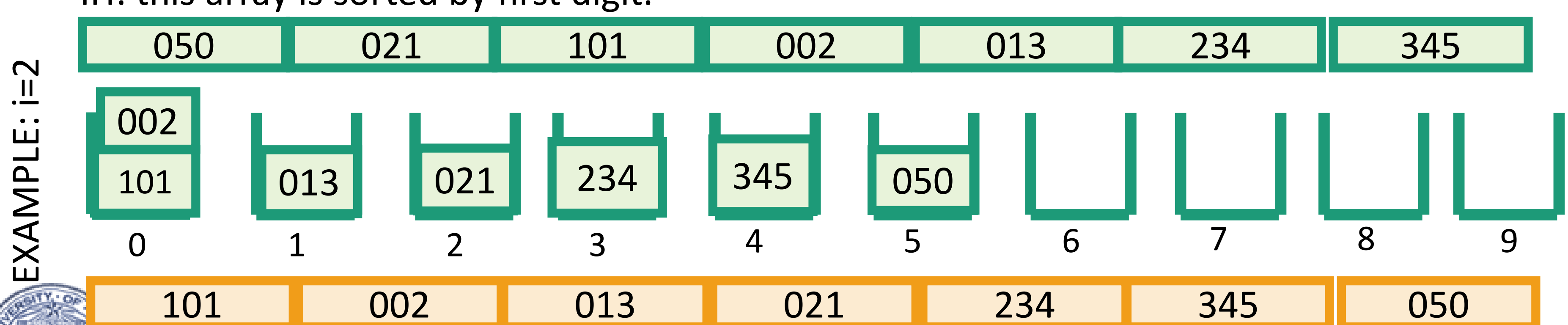
Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$

IH: this array is sorted by first digit.



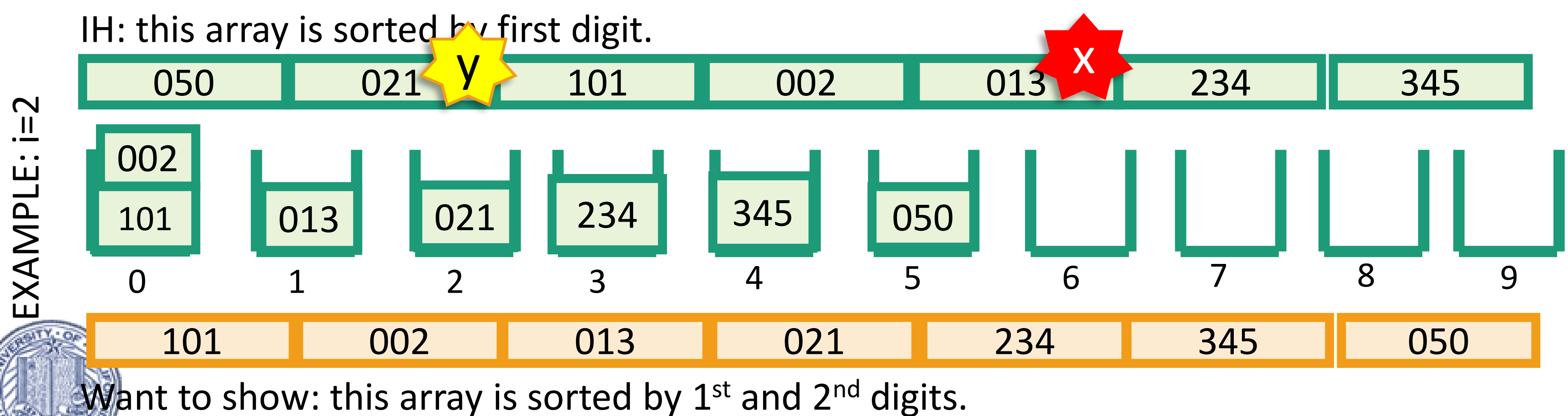
Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$

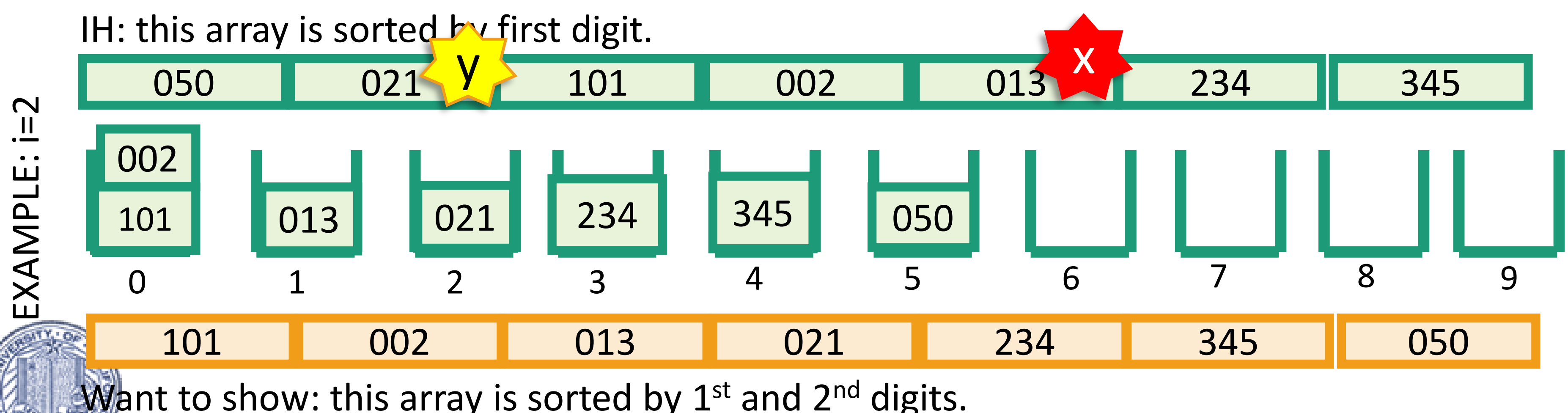


Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.



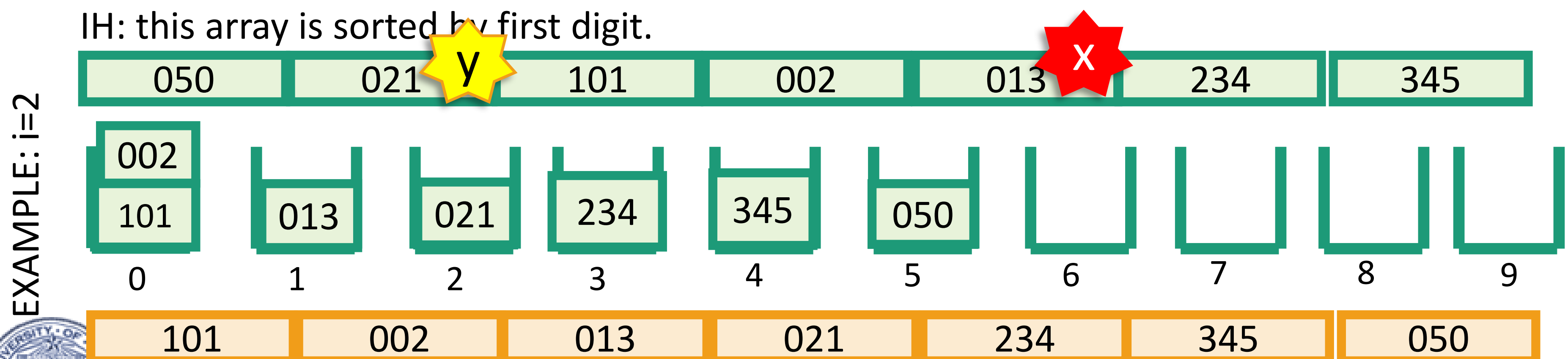
Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.

IH: this array is sorted by first digit.



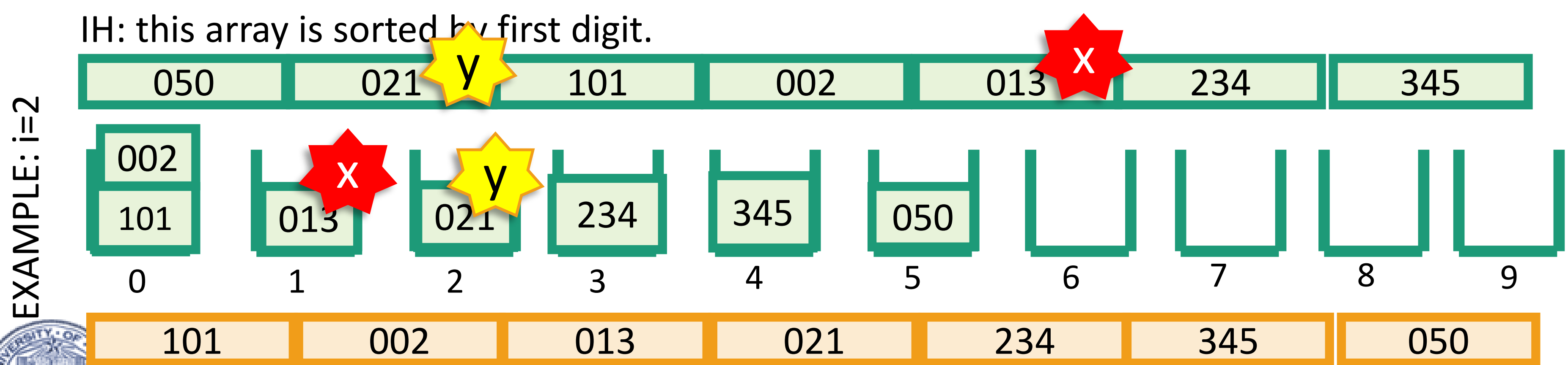
Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.



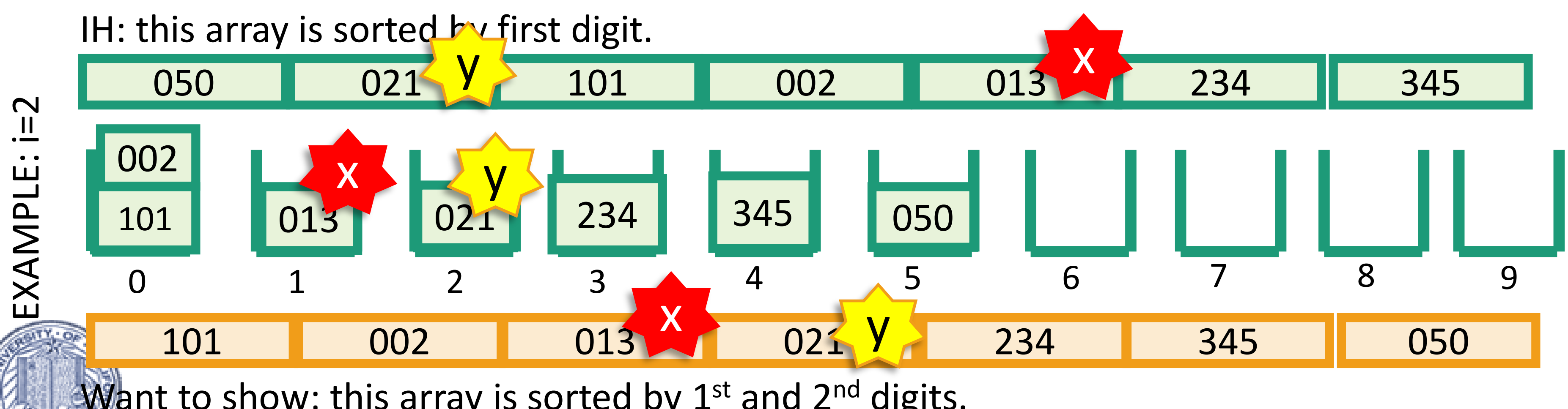
Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next (skipped) slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.

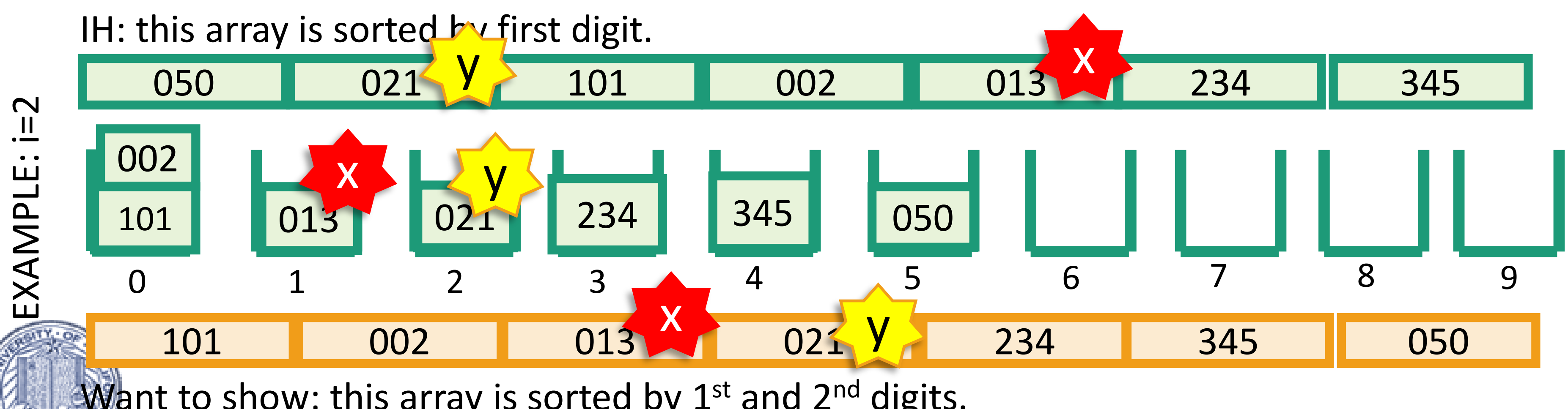


Proof sketch…

proof on next (skipped) slide

Want to show: after the i’th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$
- Suppose $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Want to show that x appears before y at end of i’th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.



Proof sketch…

proof on next slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that

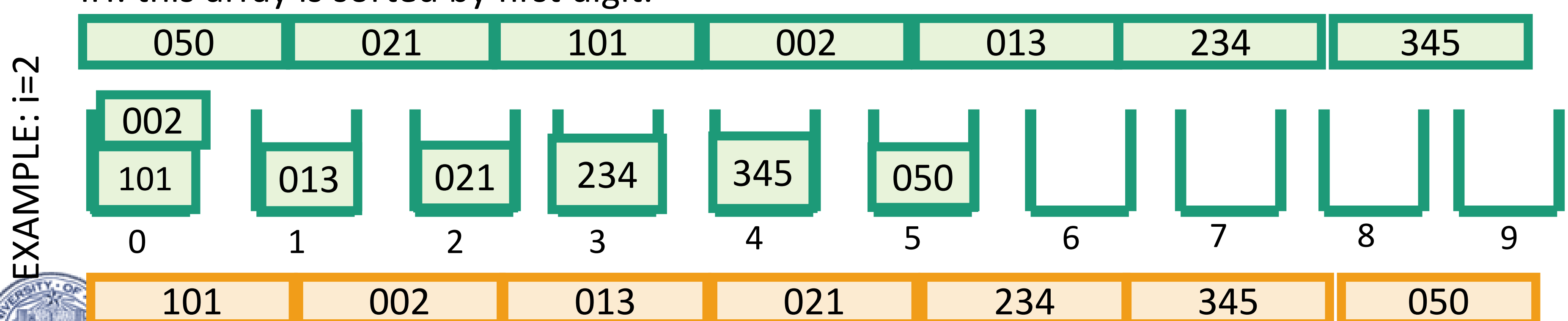
$$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$

- Want to show that x appears before y at end of i'th iteration.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y. ✓

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \dots x_2 x_1]$ and $y=[y_d y_{d-1} \dots y_2 y_1]$ be any x,y so that

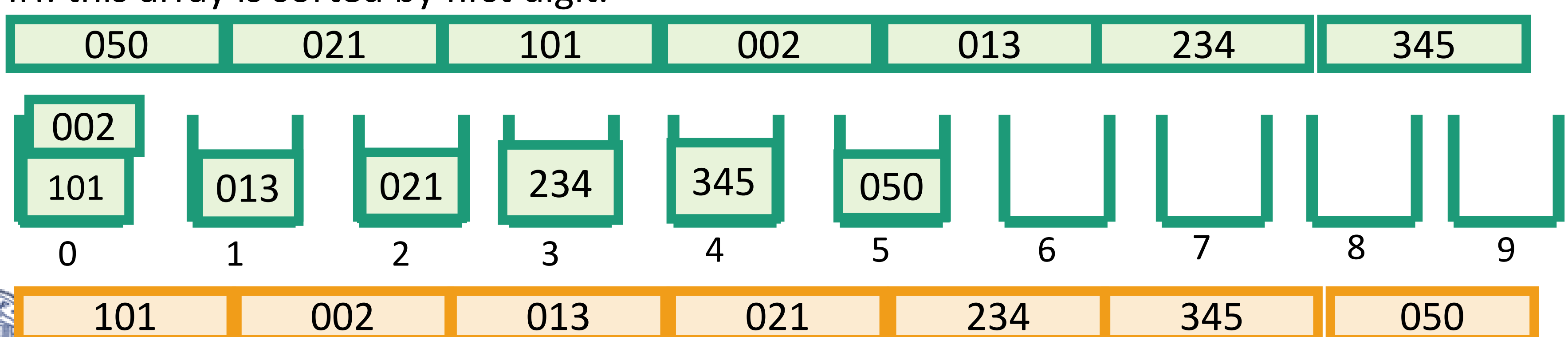
$$[x_i x_{i-1} \dots x_2 x_1] < [y_i y_{i-1} \dots y_2 y_1].$$

- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

EXAMPLE: i=2

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

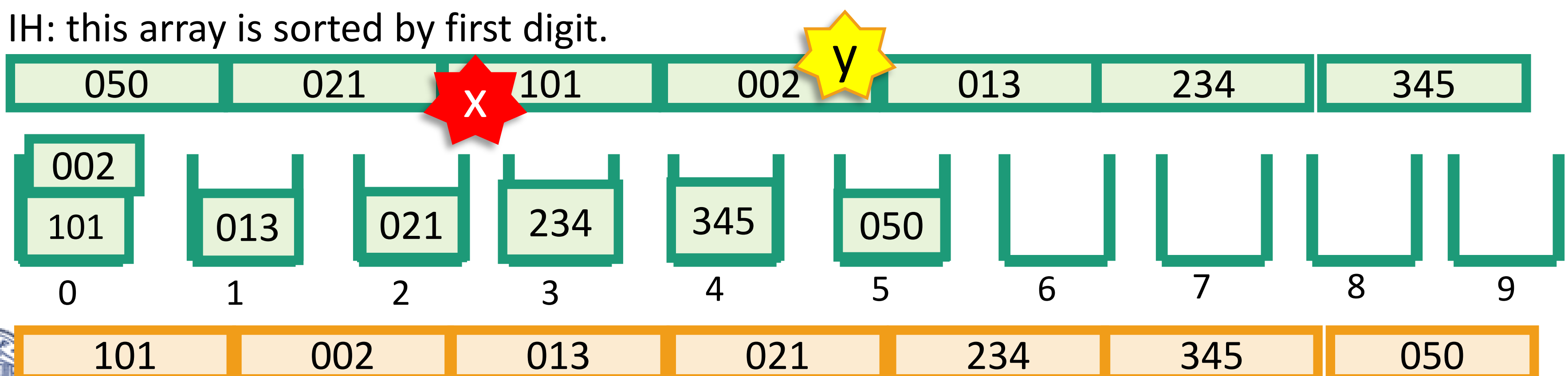
- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that
  $$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$
- Want to show that x appears before y at end of i'th iteration.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$

EXAMPLE: i=2

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

Proof sketch…

proof on next slide

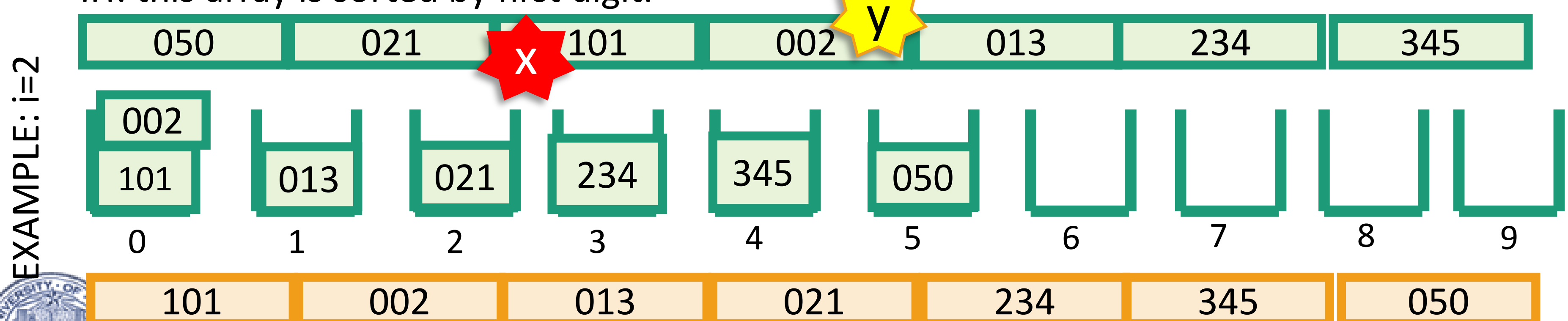
Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that
$$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$
- Want to show that x appears before y at end of i'th iteration.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$
  - x and y in same bucket, but x was put in the bucket first.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

proof on next slide

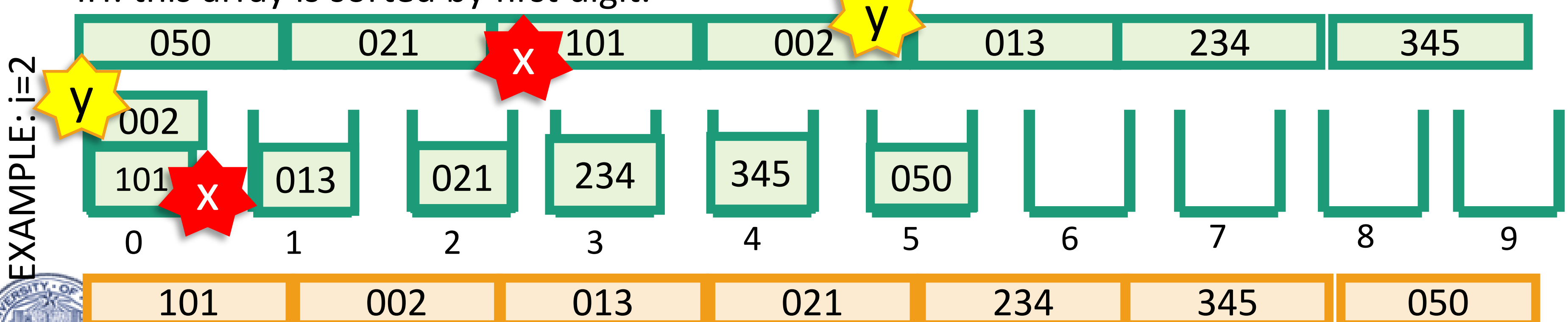
Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that
$$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$
- Want to show that x appears before y at end of i'th iteration.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$
  - x and y in same bucket, but x was put in the bucket first.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1st and 2nd digits.

Proof sketch…

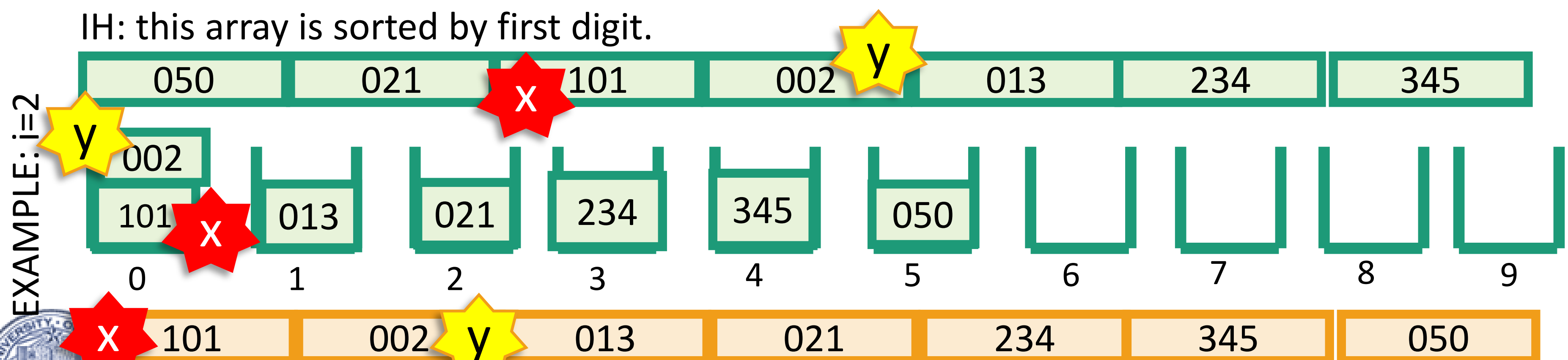
proof on next slide

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that
$$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$
  - x and y in same bucket, but x was put in the bucket first.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

Proof sketch…

proof on next slide

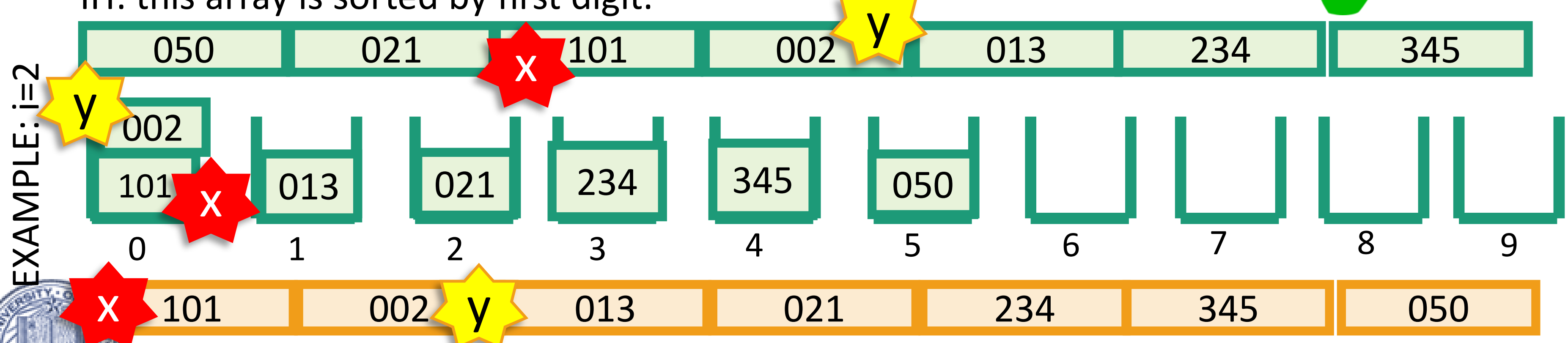
Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Let $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ be any x,y so that
  $$[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1].$$
- Want to show that x appears before y at end of i'th iteration.

Aka, we want to show that for any x and y so that x belongs before y, we put x before y.

- CASE 1: $x_i < y_i$
  - x is in an earlier bucket than y.
- CASE 2: $x_i = y_i$
  - x and y in same bucket, but x was put in the bucket first.

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

Want to show: after the i'th iteration, the array is sorted by the first i least-significant digits.

- Write $x=[x_d x_{d-1} \dots x_2 x_1]$ and $y=[y_d y_{d-1} \dots y_2 y_1]$
- Suppose $[x_i x_{i-1} \dots x_2 x_1] < [y_i y_{i-1} \dots y_2 y_1]$.
- Want to show that x appears before y at end of i'th iteration.
- CASE 1: $x_i < y_i$.
  - x appears in an earlier bucket than y, so x appears before y after the i'th iteration.
- CASE 2: $x_i = y_i$.
  - x and y end up in the same bucket.
  - In this case, $[x_{i-1} \dots x_2 x_1] < [y_{i-1} \dots y_2 y_1]$, so by the inductive hypothesis, x appeared before y after i-1'st iteration.
  - Then x was placed into the bucket before y was, so it also comes out of the bucket before y does.
    - Recall that the buckets are FIFO queues.
  - So x appears before y in the i'th iteration.

Inductive step

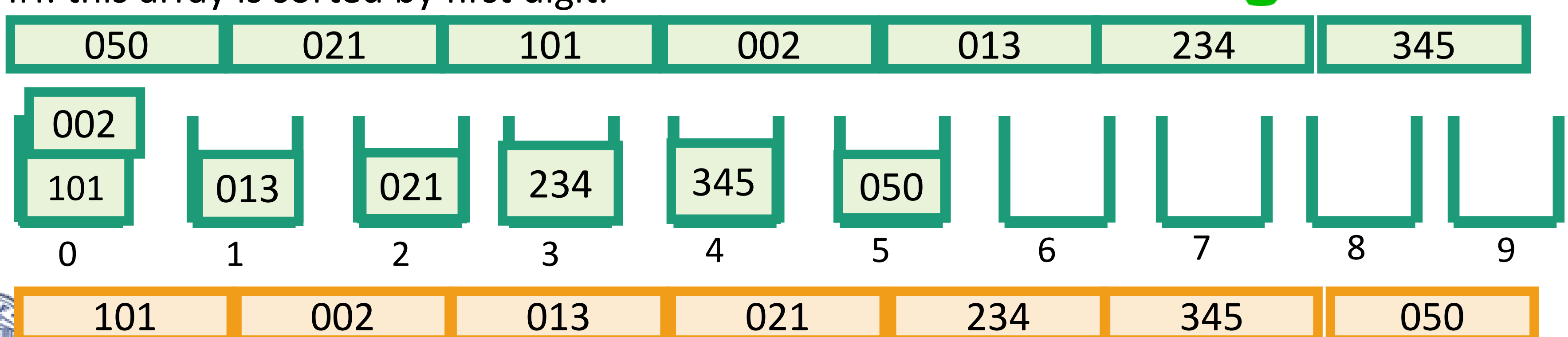
Inductive hypothesis:

After the k'th iteration, the array is sorted by the first k least-significant digits.

- Need to show: if IH holds for k=i-1, then it holds for k=i.
  - Suppose that after the i-1'st iteration, the array is sorted by the first i-1 least-significant digits.
  - Need to show that after the i'th iteration, the array is sorted by the first i least-significant digits.

EXAMPLE: i=2

IH: this array is sorted by first digit.



Want to show: this array is sorted by 1^{st} and 2^{nd} digits.

RadixSort is correct

- Inductive hypothesis:
  - After the k'th iteration, the array is sorted by the first k least-significant digits.
- Base case:
  - "Sorted by 0 least-significant digits" means not sorted, so the IH holds for k=0.
- Inductive step:
  - TO DO ✓
- Conclusion:
  - The inductive hypothesis holds for all k, so after the last iteration, the array is sorted by all the digits. Hence, it's sorted!

What is the running time?

for RadixSorting numbers base-10.

- Suppose we are sorting n d-digit numbers (in base 10).

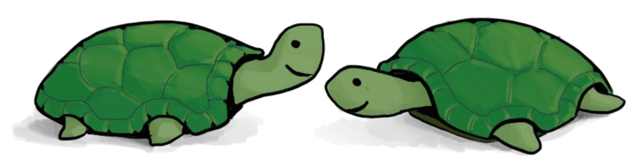
e.g., n=7, d=3:

021	345	013	101	050	234	001

1. How many iterations are there?

2. How long does each iteration take?

3. What is the total running time?



Think-Pair-Share Terrapins

What is the running time?

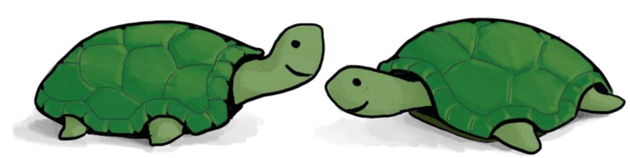
for RadixSorting numbers base-10.

- Suppose we are sorting n d-digit numbers (in base 10).

e.g., n=7, d=3:

021	345	013	101	050	234	001

1. How many iterations are there?
   - d iterations
2. How long does each iteration take?
   - Time to initialize 10 buckets, plus time to put n numbers in 10 buckets. O(n).
3. What is the total running time?
   - O(nd)



Think-Pair-Share Terrapins

This doesn't seem so great

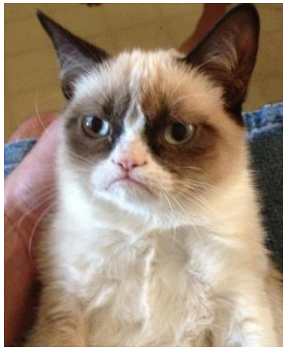
- To sort n integers, each of which is in {1,2,…,n}…
- d = $\lfloor \log_{10}(n) \rfloor + 1$
  - For example:
    - n = 1234
    - $\lfloor \log_{10}(1234) \rfloor + 1 = 4$
  - More explanation on next slide.

Aside: why $d = \lfloor \log_{10}(n) \rfloor + 1$?

- When we write a number $x = [x_d x_{d-1} \ldots x_1]$ base 10, that means:
  $$x = x_1 + x_2 \cdot 10 + \cdots + x_{d-1} \cdot 10^{d-2} + x_d \cdot 10^{d-1}$$
  where $x_i \in \{0,1, \ldots, 9\}$
- Suppose that $x_d \neq 0$. Then we have
  - $x \geq x_d \cdot 10^{d-1}$ Since x is bigger than just the last term in that sum!
  - $\log_{10}(x) + 1 - \log_{10}(x_d) \geq d$ (take logs$_{10}$ of both sides and rearrange)
  - $\log_{10}(x) + 1 > d$ $\log_{10}(x_d) > 0$ since $x_d > 0$
  - $\lfloor \log_{10}(n) \rfloor + 1 \geq d$ Since d is an integer
- On the other hand, we also have
  - $x < (x_d + 1) \cdot 10^{d-1}$ Since if $x \geq (x_d + 1) \cdot 10^{d-1}$ then the d'th digit would have been x_d +1 instead of x_d
  - $\log_{10}(x) + 1 - \log_{10}(x_d + 1) < d$ (take logs$_{10}$ of both sides and rearrange)
  - $\log_{10}(x) < d$ $\log_{10}(x_d + 1) \leq 1$ since $x_d < 10$
  - $\lfloor \log_{10}(n) \rfloor + 1 \leq d$ Since d is an integer

This doesn’t seem so great

- To sort n integers, each of which is in {1,2,…,n}…
- d = $\lfloor \log_{10}(n) \rfloor + 1$
  - For example:
    - n = 1234
    - $\lfloor \log_{10}(1234) \rfloor + 1 = 4$
- Time = $O(nd) = O(n\log(n))$.
  - Same as MergeSort!



Can we do better?

- RadixSort base 10 doesn't seem to be such a good idea…
- But what if we change the base? (Let's say base r)
- We will see there's a trade-off:
  - Bigger r means more buckets
  - Bigger r means fewer digits

Example: base 100

Original array:

21	345	13	101	50	234	1

Example: base 100

Original array:

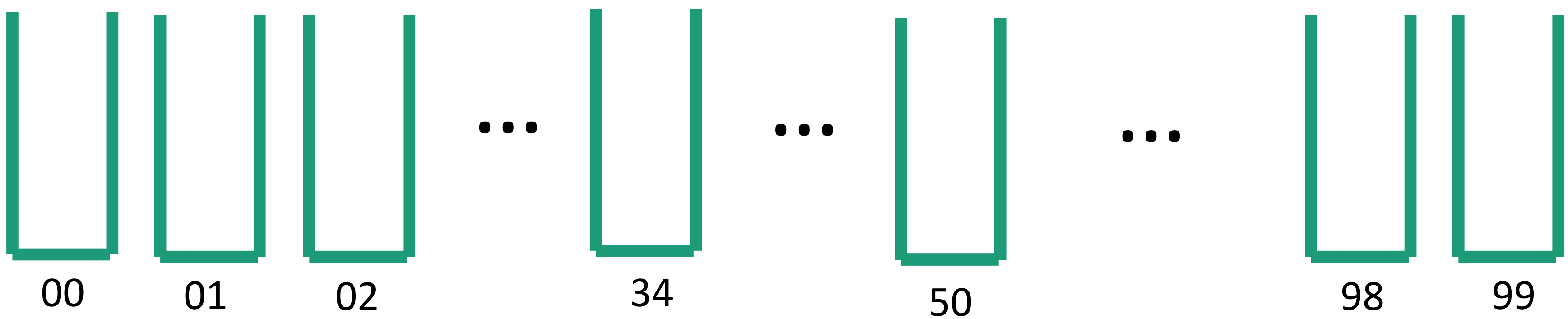
0021	0345	0013	0101	0050	0234	0001

Example: base 100

Original array:

0021	0345	0013	0101	0050	0234	0001

100 buckets:

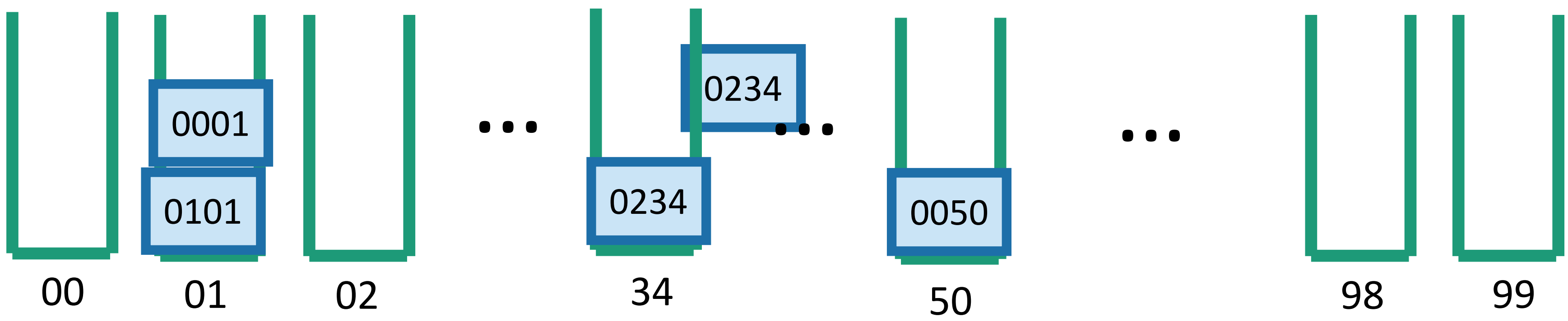


Example: base 100

Original array:

0021	0345	0013	0101	0050	0234	0001

100 buckets:

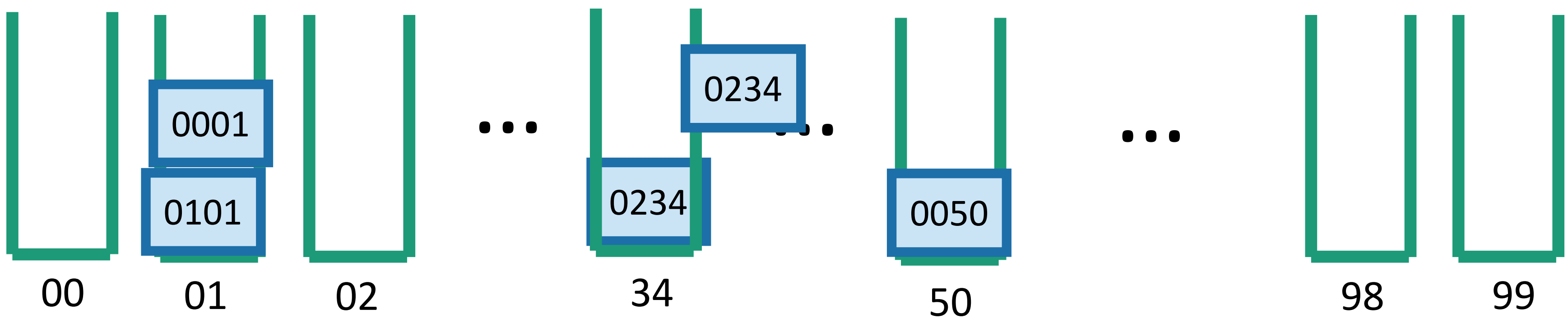


Example: base 100

Original array:

0021	0345	0013	0101	0050	0234	0001

100 buckets:

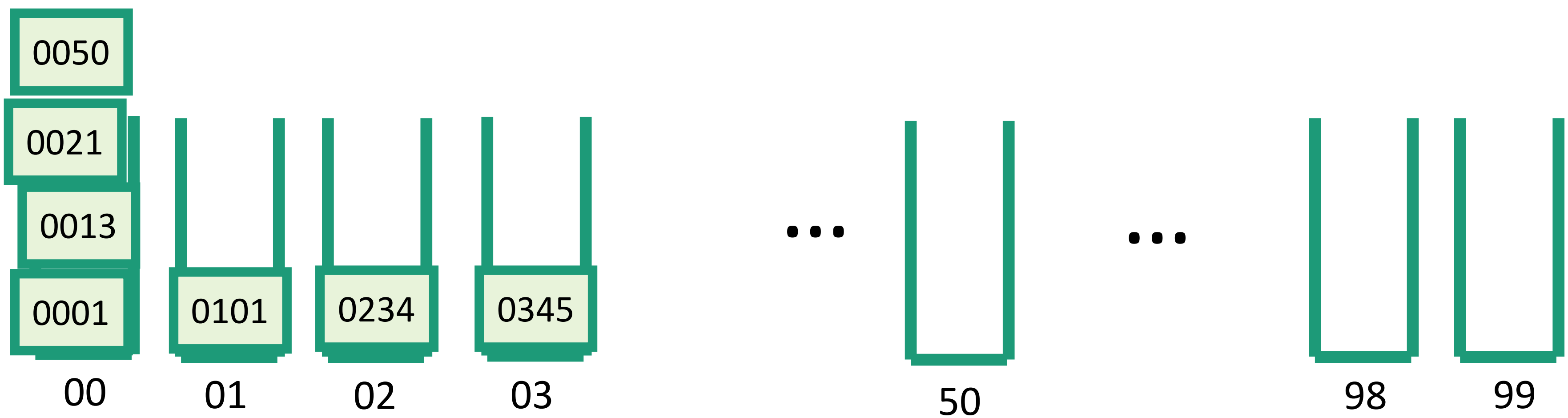


0101	0001	0013	0021	0234	0345	0050

Example: base 100

0101	0001	0013	0021	0234	0345	0050

100 buckets:



0001	0013	0021	0050	0101	0234	0345

Sorted!

Example: base 100

Original array

0021	0345	0013	0101	0050	0234	0001

0101	0001	0013	0021	0234	0345	0050

0001	0013	0021	0050	0101	0234	0345

Sorted array

Base 100:

- d=2, so only 2 iterations.
- 100 buckets

vs.

Base 10:

- d=3, so 3 iterations.
- 10 buckets

Bigger base means more buckets but fewer iterations.

General running time of RadixSort

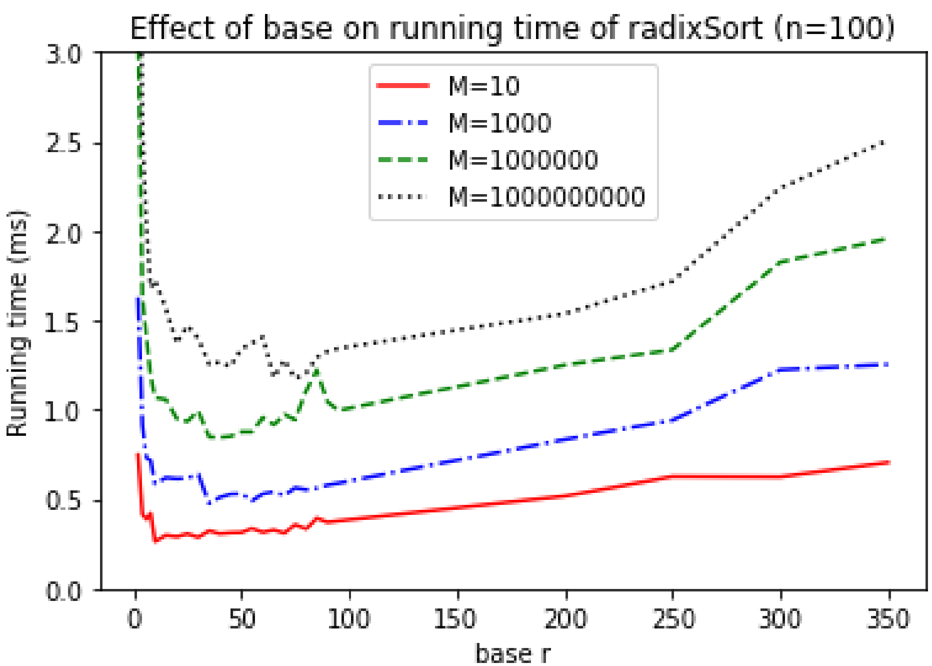
- Say we want to sort:
  - n integers,
  - maximum size M,
  - in base r.
- Number of iterations of RadixSort:
  - Same as number of digits, base r, of an integer x of max size M.
  - That is $d = \lfloor \log_r(M) \rfloor + 1$
- Time per iteration:
  - Initialize r buckets, put n items into them
  - $O(n + r)$ total time.
- Total time:
  - $O\big(d \cdot (n + r)\big) = O\big((\lfloor \log_r(M) \rfloor + 1) \cdot (n + r)\big)$

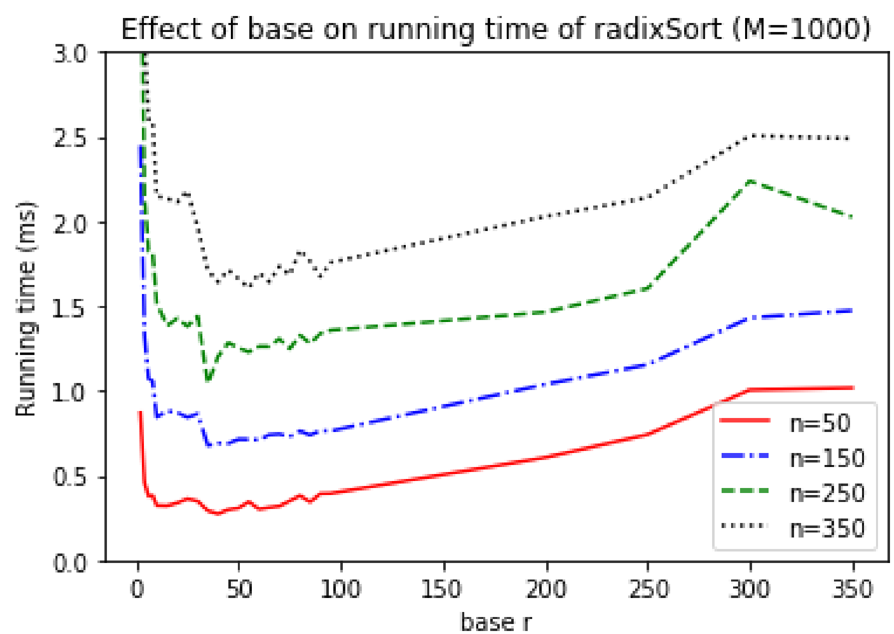
Convince yourself that this is the right formula for d.

Siggi the Studious Stork

Trade-offs

- Given n, M, how should we choose r?
- Looks like there's some sweet spot:





A reasonable choice: r=n

- Running time:

$$O\Big(\big(\lfloor \log_r(M) \rfloor + 1\big) \cdot (n + r)\Big)$$

Intuition: balance n and r here.

- Choose n=r:

$$O\Big(n \cdot \big(\lfloor \log_n(M) \rfloor + 1\big)\Big)$$

Choosing r = n is pretty good. What choice of r optimizes the asymptotic running time? What if I also care about space?

Ollie the over-achieving ostrich

Running time of RadixSort with r=n

- To sort n integers of size at most M, time is

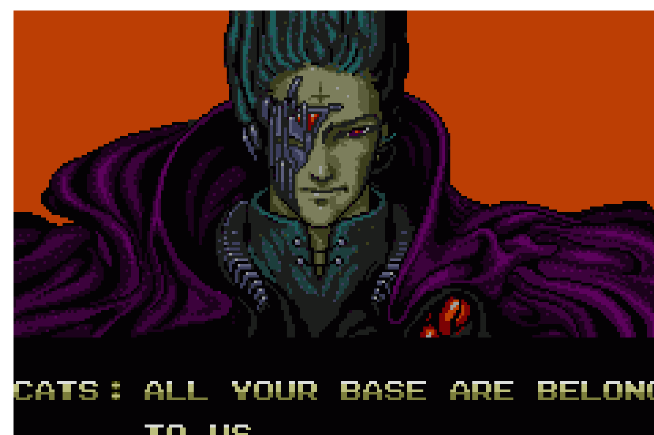
$$O\left(n \cdot \left(\lfloor \log_n(M) \rfloor + 1 \right)\right)$$

- So the running time (in terms of n) depends on how big M is in terms of n:
  - If $M \leq n^c$ for some constant c, then this is O(n).
  - If $M = 2^n$, then this is $O\left(\frac{n^2}{\log(n)}\right)$
- The number of buckets needed is r=n.

What have we learned?

You can put any constant here instead of 100.

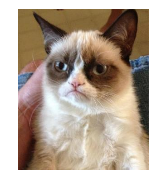
- RadixSort can sort n integers of size at most n^{100} in time O(n), and needs enough space to store O(n) integers.
- If your integers have size much much bigger than n (like 2^n), maybe you shouldn't use RadixSort.
- It matters how we pick the base.



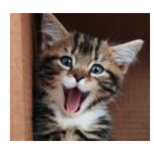
Recap

- How difficult sorting is depends on the model of computation.
- How reasonable a model of computation is is up for debate.

- Comparison-based sorting model
  - This includes MergeSort, QuickSort, InsertionSort
  - Any algorithm in this model must use at least $\Omega(n \log(n))$ operations. ☹
  - But it can handle arbitrary comparable objects. ☺



- If we are sorting small integers (or other reasonable data):
  - BucketSort and RadixSort
  - Both run in time $O(n)$ ☺
  - Might take more space and/or be slower if integers get too big ☹

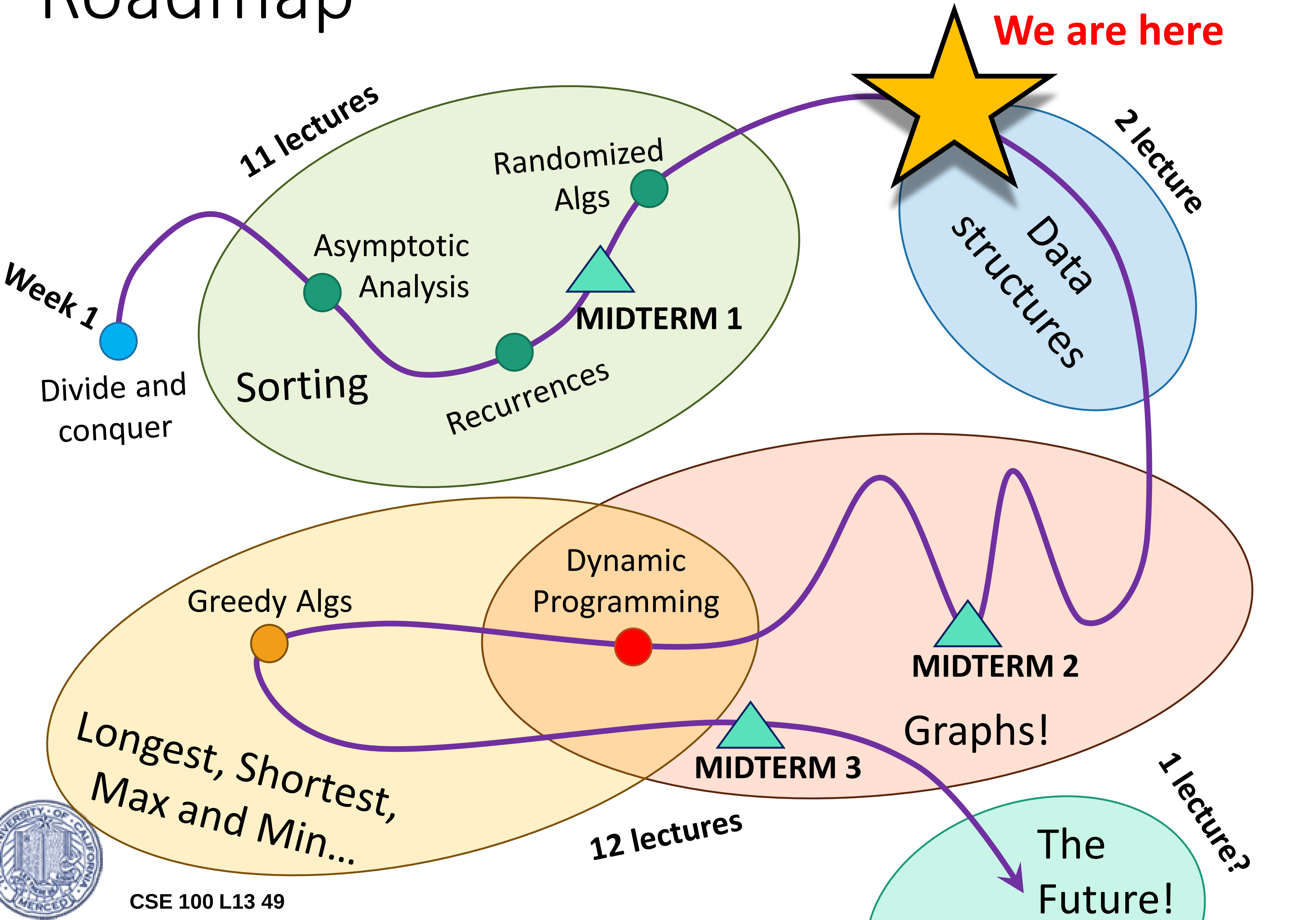


Next Part

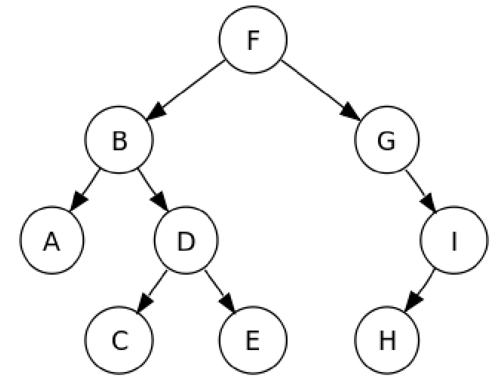
- Binary search trees!
- Balanced binary search trees!


CHUCK NORRIS
QUICKSORTS STICKS
IN TIME O(1)
imgflip.com

Roadmap


We are here
Week 1
Divide and conquer
11 lectures
Sorting
Asymptotic Analysis
Recurrences
Randomized Algs
MIDTERM 1
2 lecture
Data structures
Greedy Algs
Dynamic Programming
MIDTERM 2
Graphs!
MIDTERM 3
Longest, Shortest, Max and Min…
12 lectures
1 lecture?
The Future!

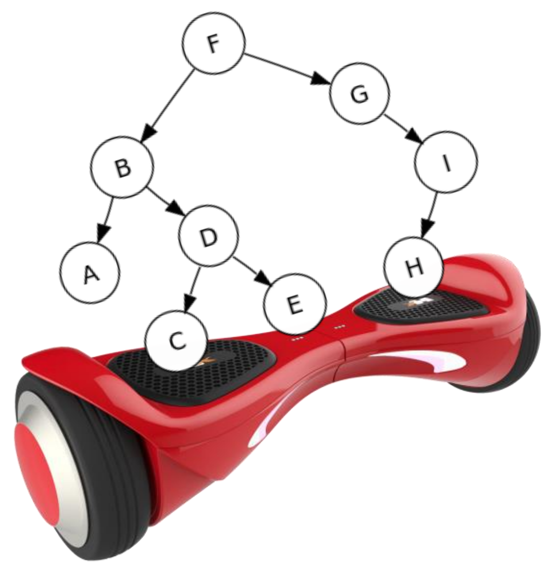
Today (part 2)



- Begin a brief foray into data structures!
- Binary search trees
  - You may remember these from CSE 30
  - They are better when they're balanced.

this will lead us to…

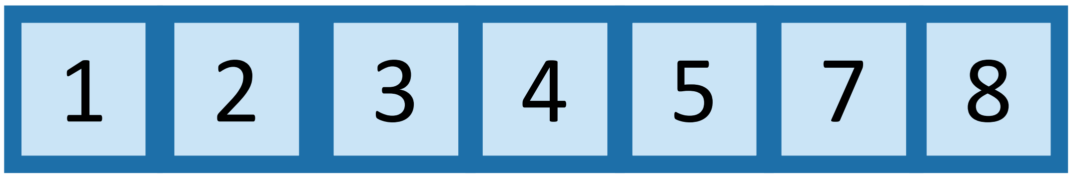
- Self-Balancing Binary Search Trees
  - **Red-Black** trees.



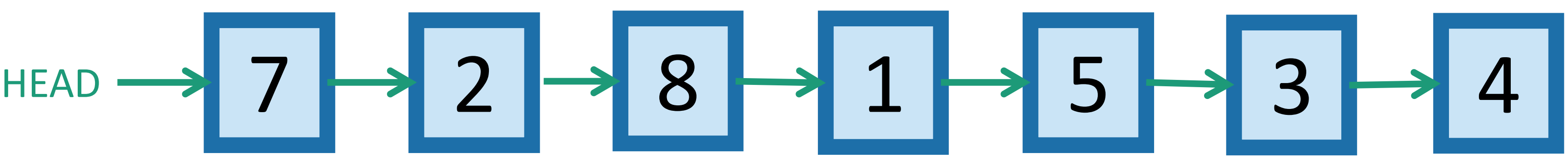
Some data structures

for storing objects like 5 (aka, nodes with keys)

- (Sorted) arrays:



- (UnSorted) linked lists:



- Some basic operations:
  - INSERT, DELETE, SEARCH

Sorted Arrays

1	2	3	4	5	7	8

Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

1	2	3	4	5	7	8

Sorted Arrays

1	2	3	4	5	7	8

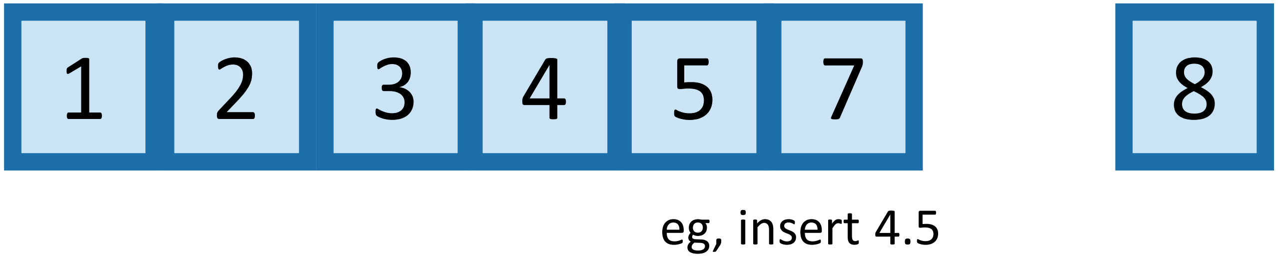
- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

1	2	3	4	5	7	8

eg, insert 4.5

Sorted Arrays

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:



Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

1	2	3	4	5		7	8

eg, insert 4.5

Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

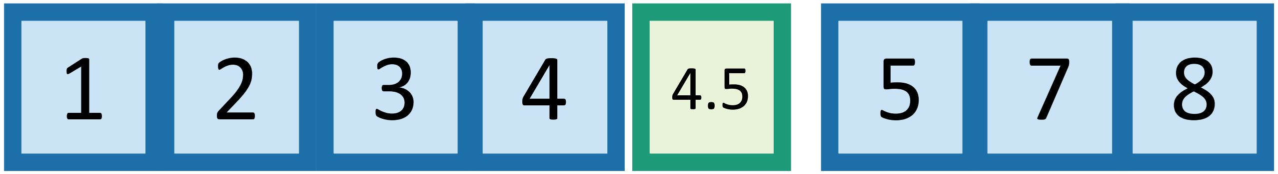
1	2	3	4

5	7	8

eg, insert 4.5

Sorted Arrays

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:



eg, insert 4.5

Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

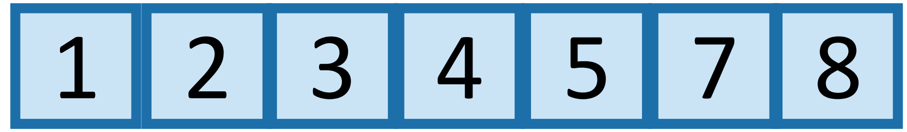
1	2	3	4	4.5	5	7	8

eg, insert 4.5

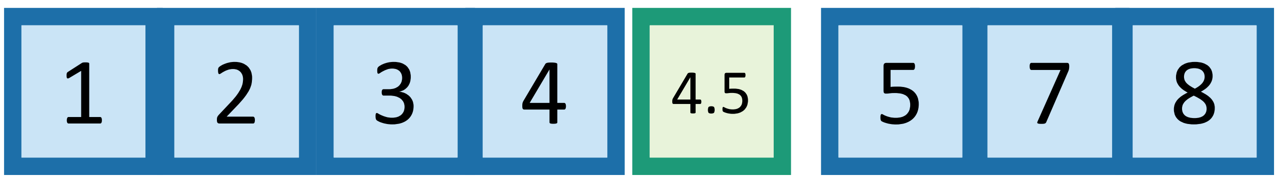
- O(log(n)) SEARCH:

1	2	3	4	5	7	8

Sorted Arrays

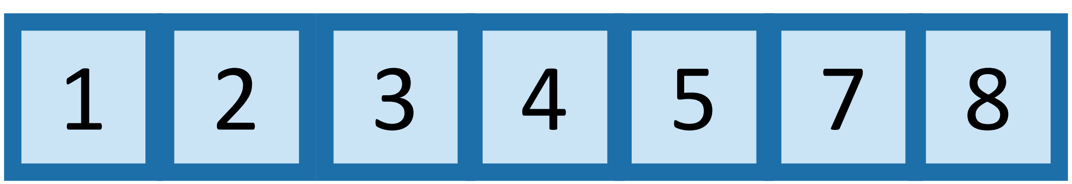


- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:



eg, insert 4.5

- O(log(n)) SEARCH:



eg, Binary search to see if 3 is in A.

Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

1	2	3	4	4.5	5	7	8

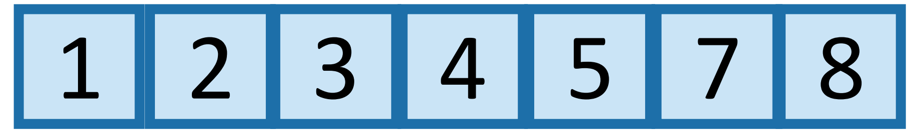
eg, insert 4.5

- O(log(n)) SEARCH:

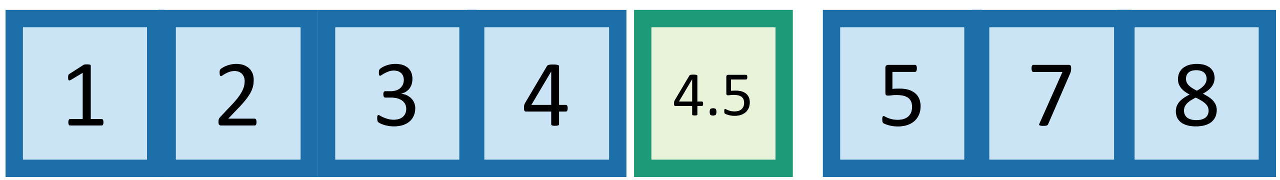
1	2	3	4	5	7	8

eg, Binary search to see if 3 is in A.

Sorted Arrays

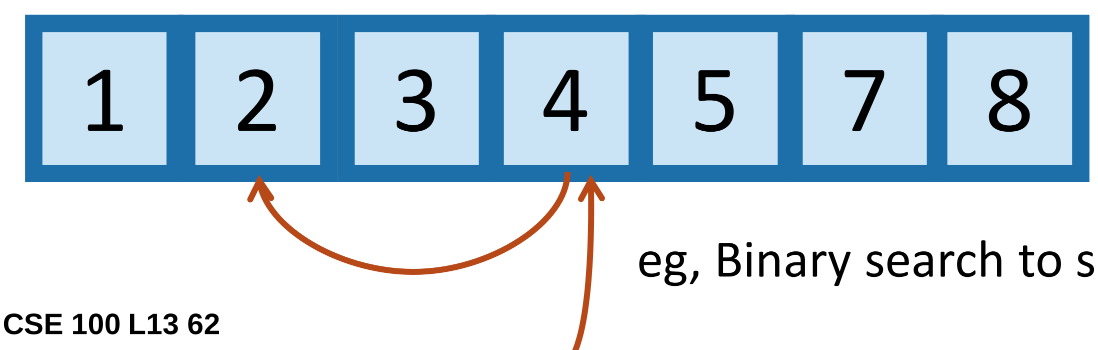


- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:



eg, insert 4.5

- O(log(n)) SEARCH:



eg, Binary search to see if 3 is in A.

Sorted Arrays

1	2	3	4	5	7	8

- O(n) INSERT/DELETE:
  - First, find the relevant element (time O(log(n)) as below), and then move a bunch elements in the array:

1	2	3	4	4.5	5	7	8

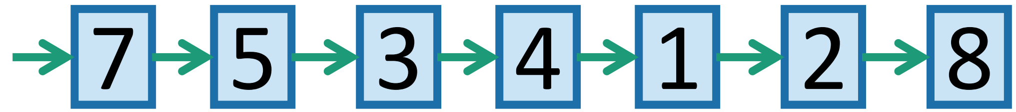
eg, insert 4.5

- O(log(n)) SEARCH:

1	2	3	4	5	7	8

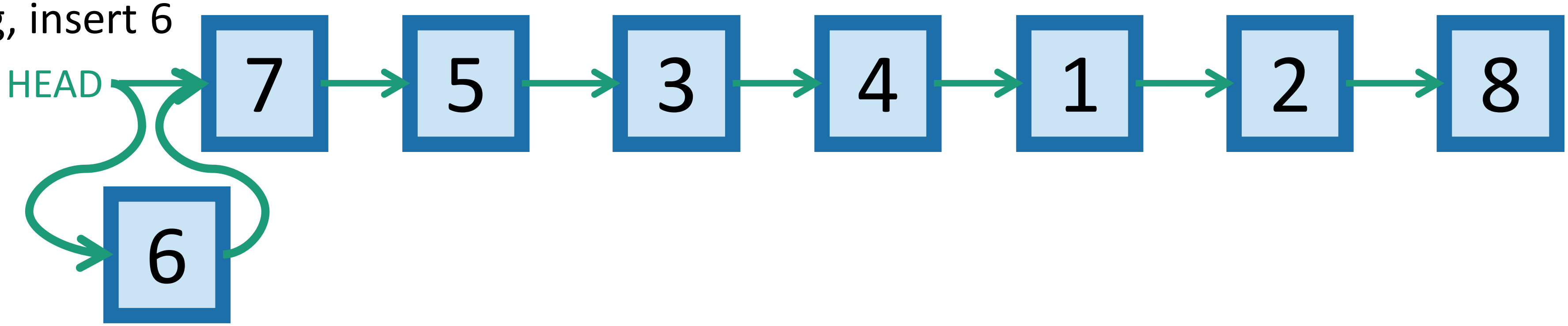
eg, Binary search to see if 3 is in A.

UNSorted linked lists

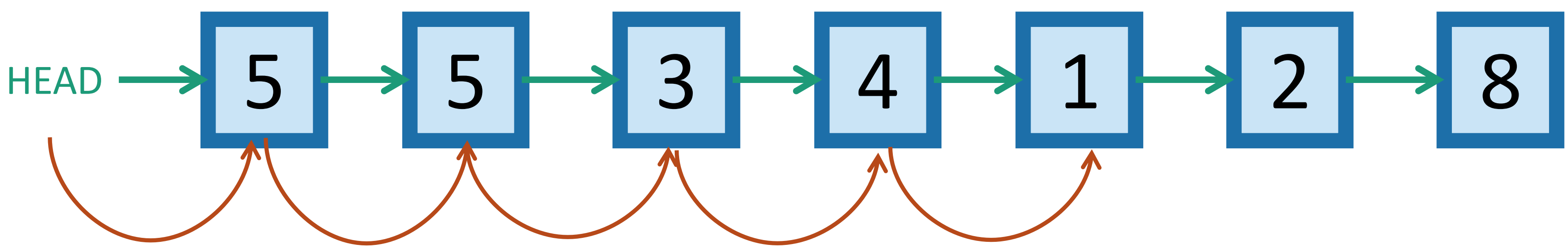


- O(1) INSERT:

eg, insert 6



- O(n) SEARCH/DELETE:



eg, search for 1 (and then you could delete it by manipulating pointers).

Motivation for Binary Search Trees

TODAY!

	Sorted Arrays	Linked Lists	Binary Search Trees*
Search	O(log(n)) 😃	O(n) ☹️	O(log(n)) 😃
Delete	O(n) ☹️	O(n) ☹️	O(log(n)) 😃
Insert	O(n) ☹️	O(1) 😃	O(log(n)) 😃

Binary tree terminology

Each node has two children.

The left child of 3 is 2

The right child of 3 is 4

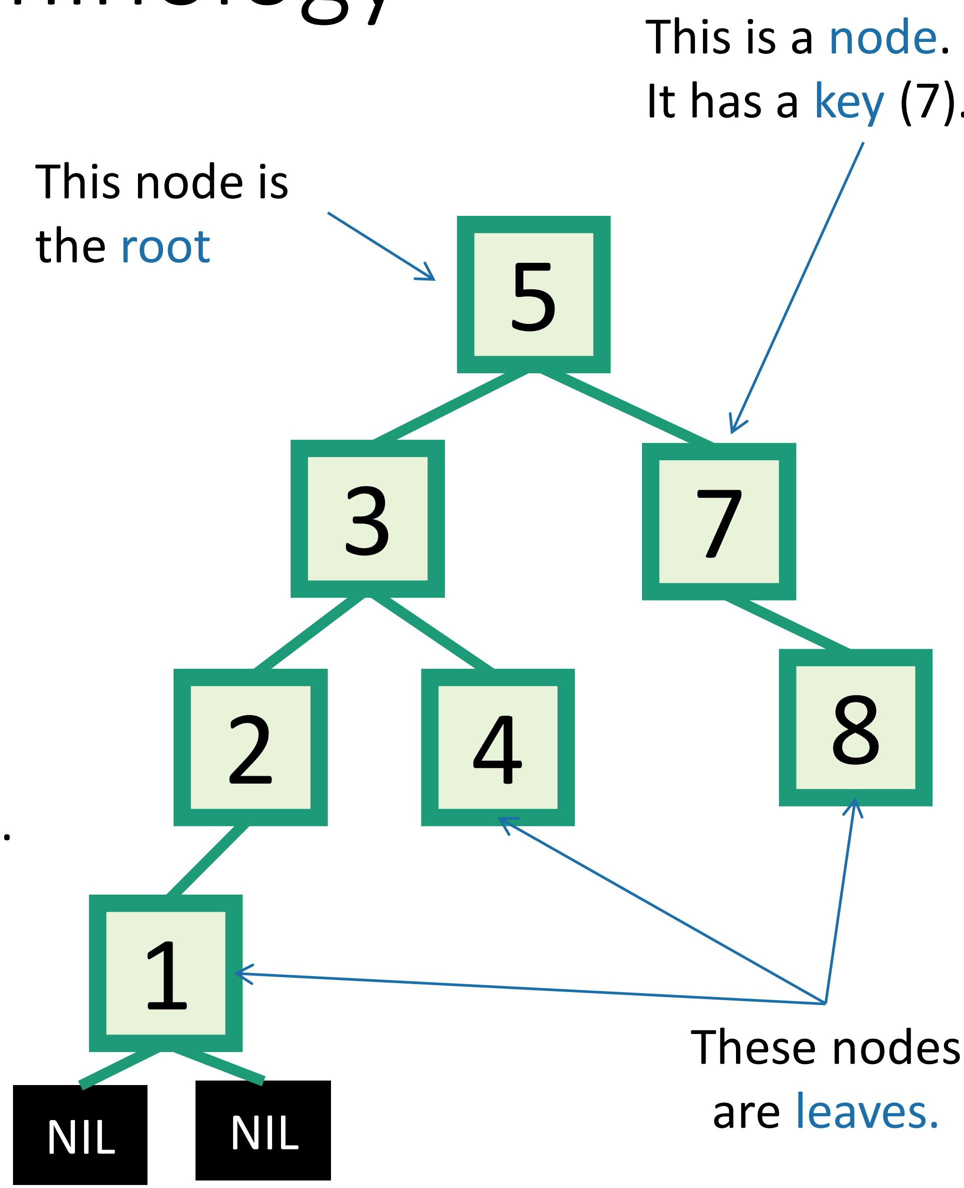
The parent of 3 is 5

2 is a descendant of 5

Each node has a pointer to its left child, right child, and parent.

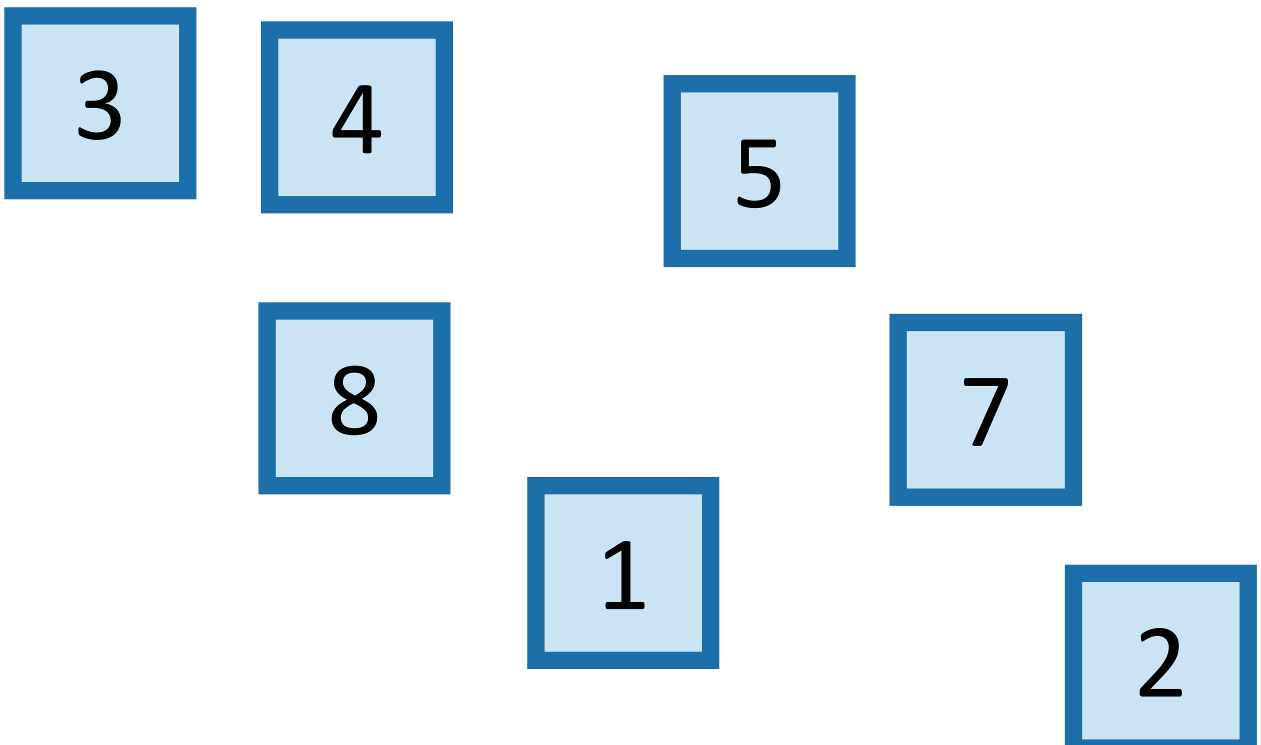
Both children of 1 are NIL.
(We won't usually draw them).

The height of this tree is 3.
(Max number of edges from the root to a leaf).



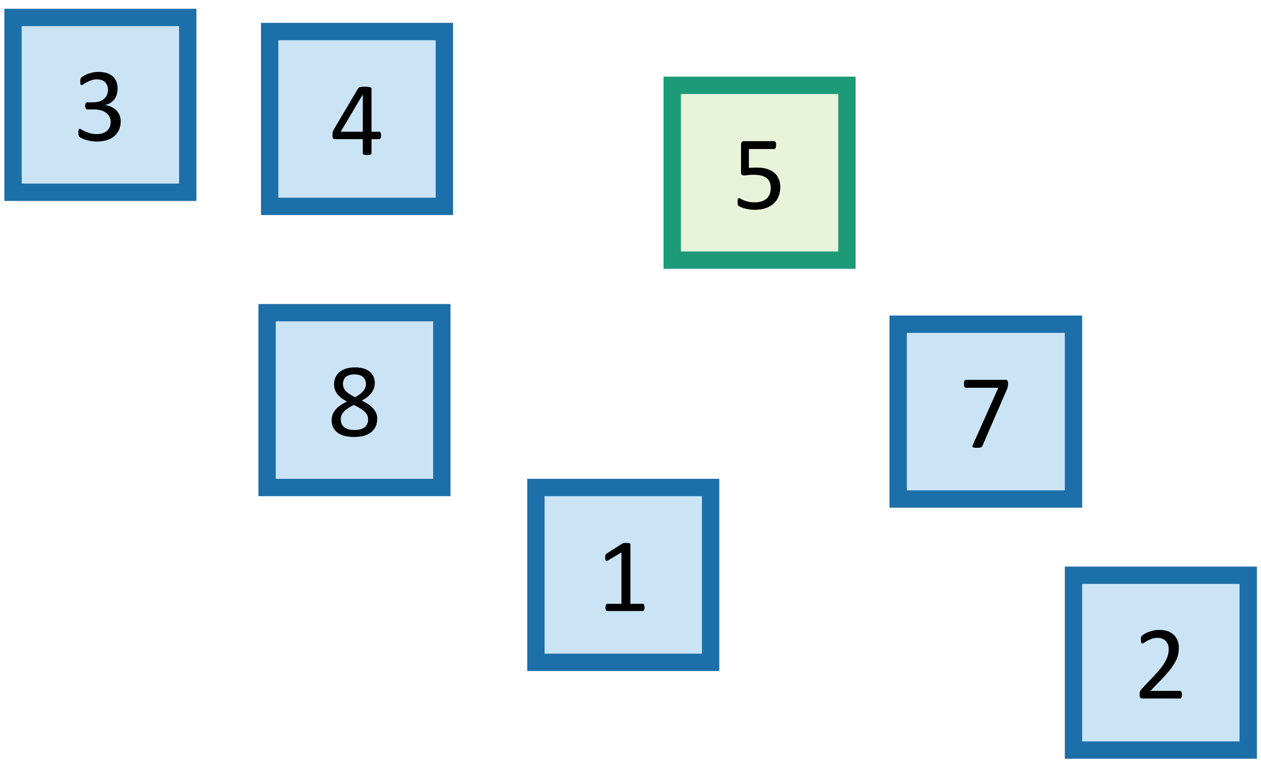
Binary Search Trees

- A BST is a binary tree so that:
    - Every LEFT descendant of a node has key less than that node.
    - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



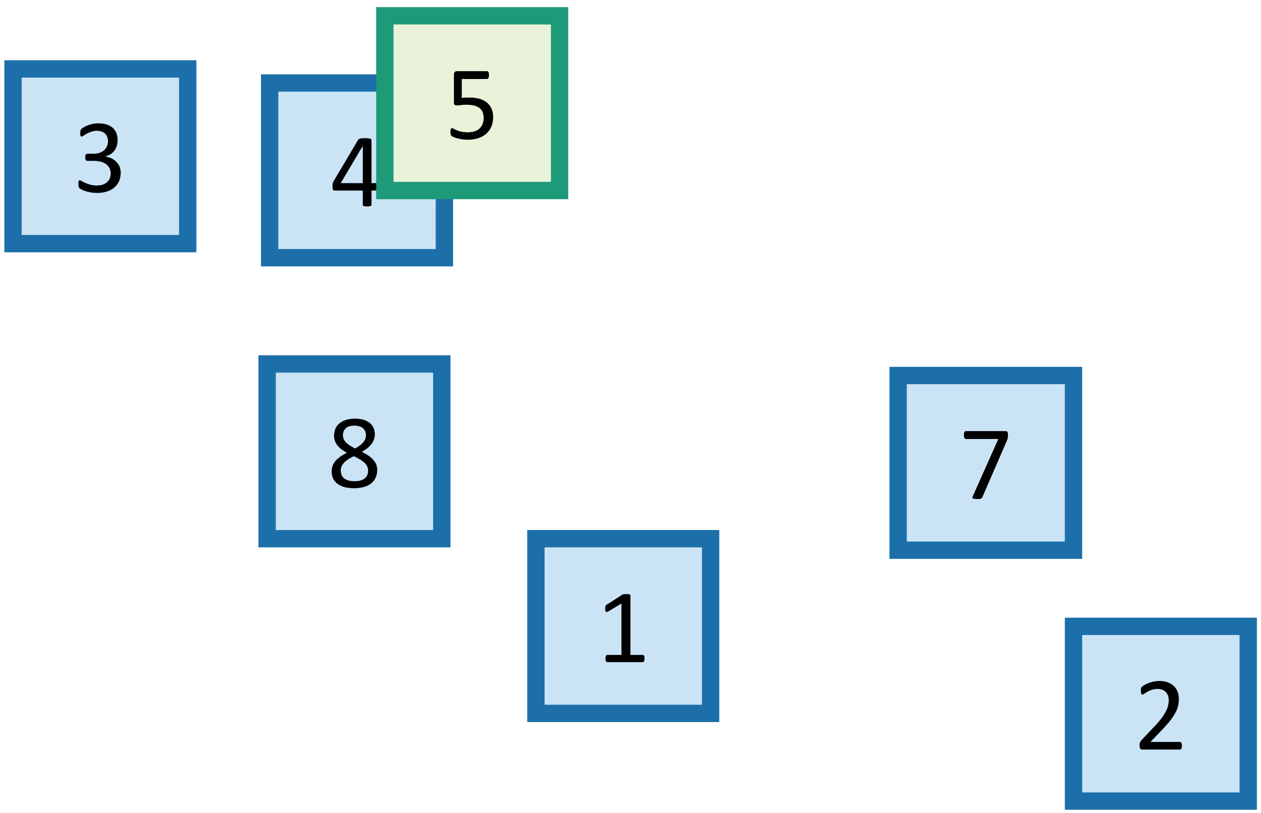
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



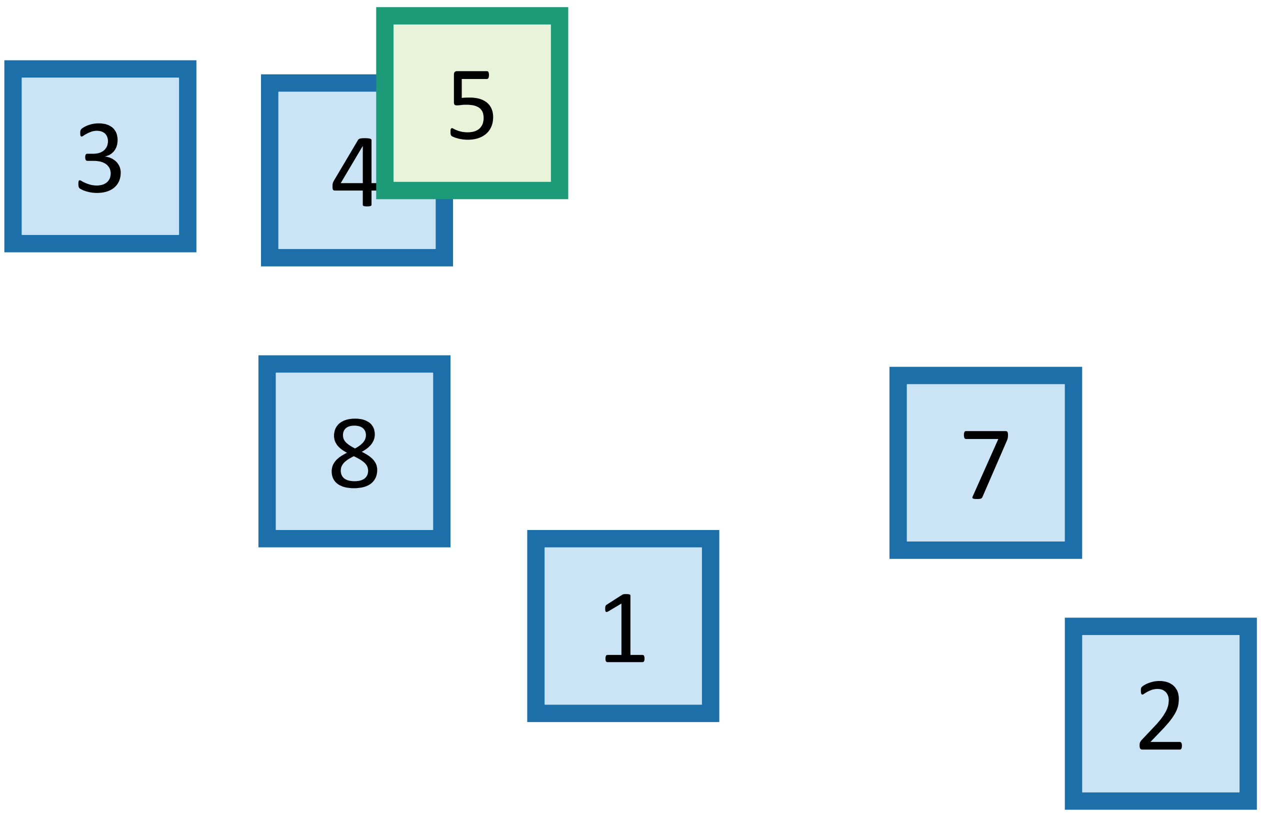
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



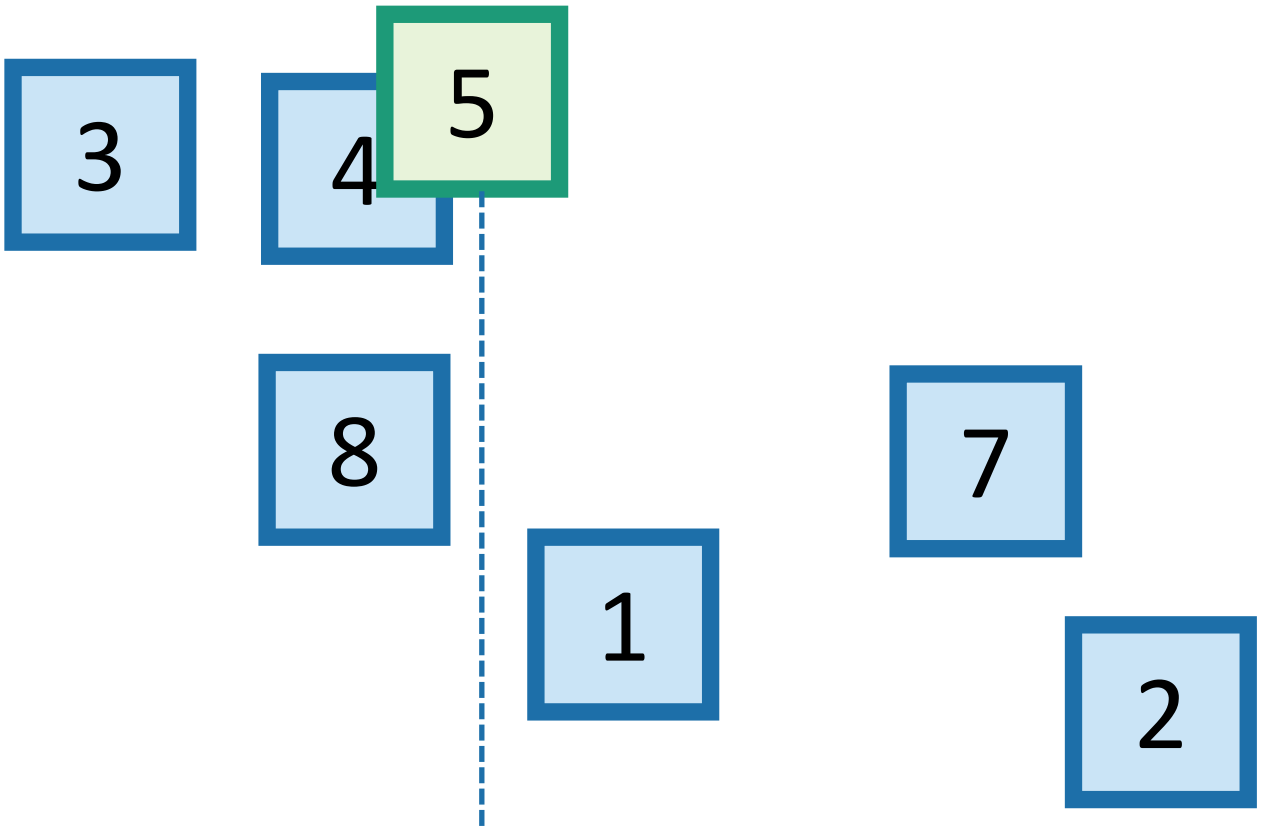
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



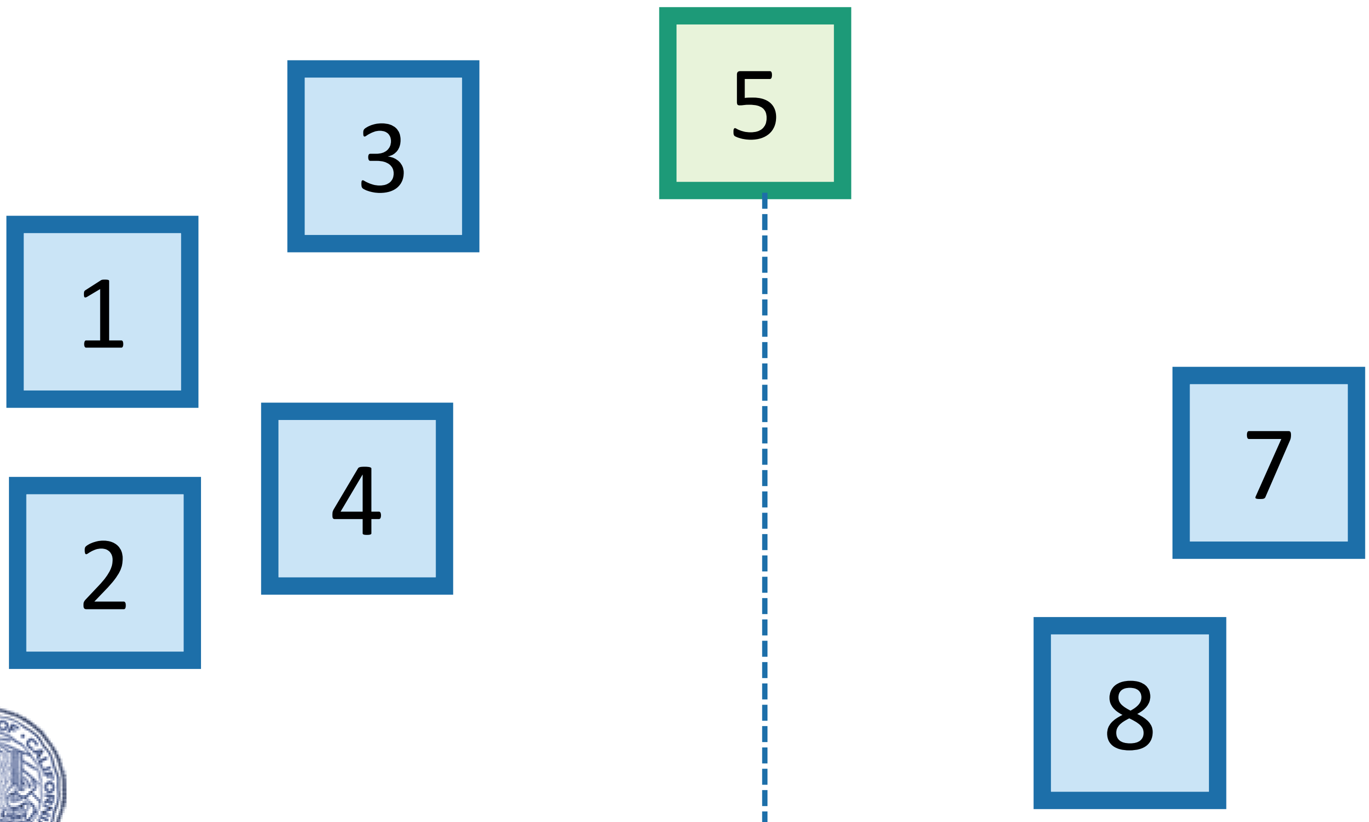
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



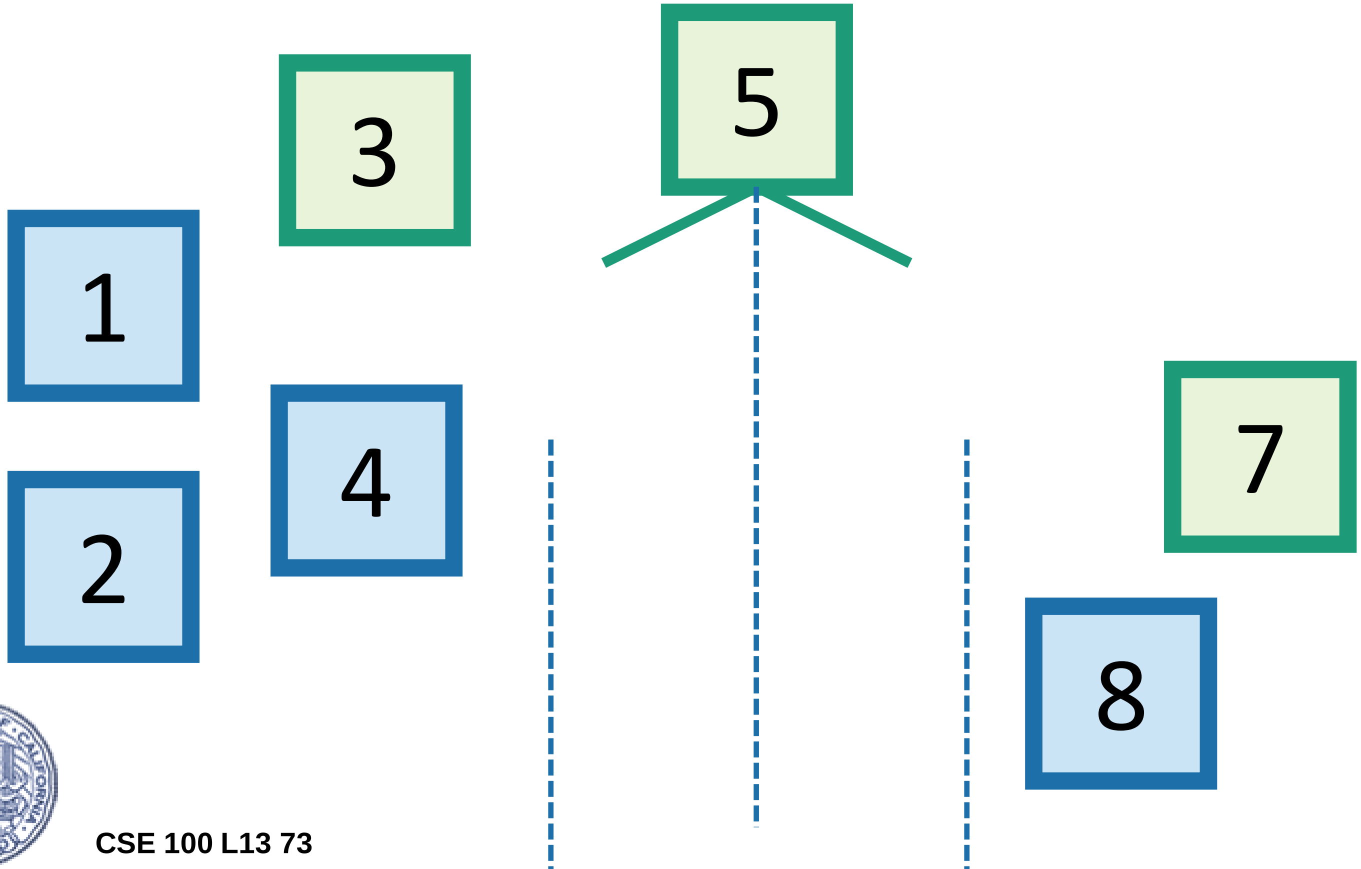
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



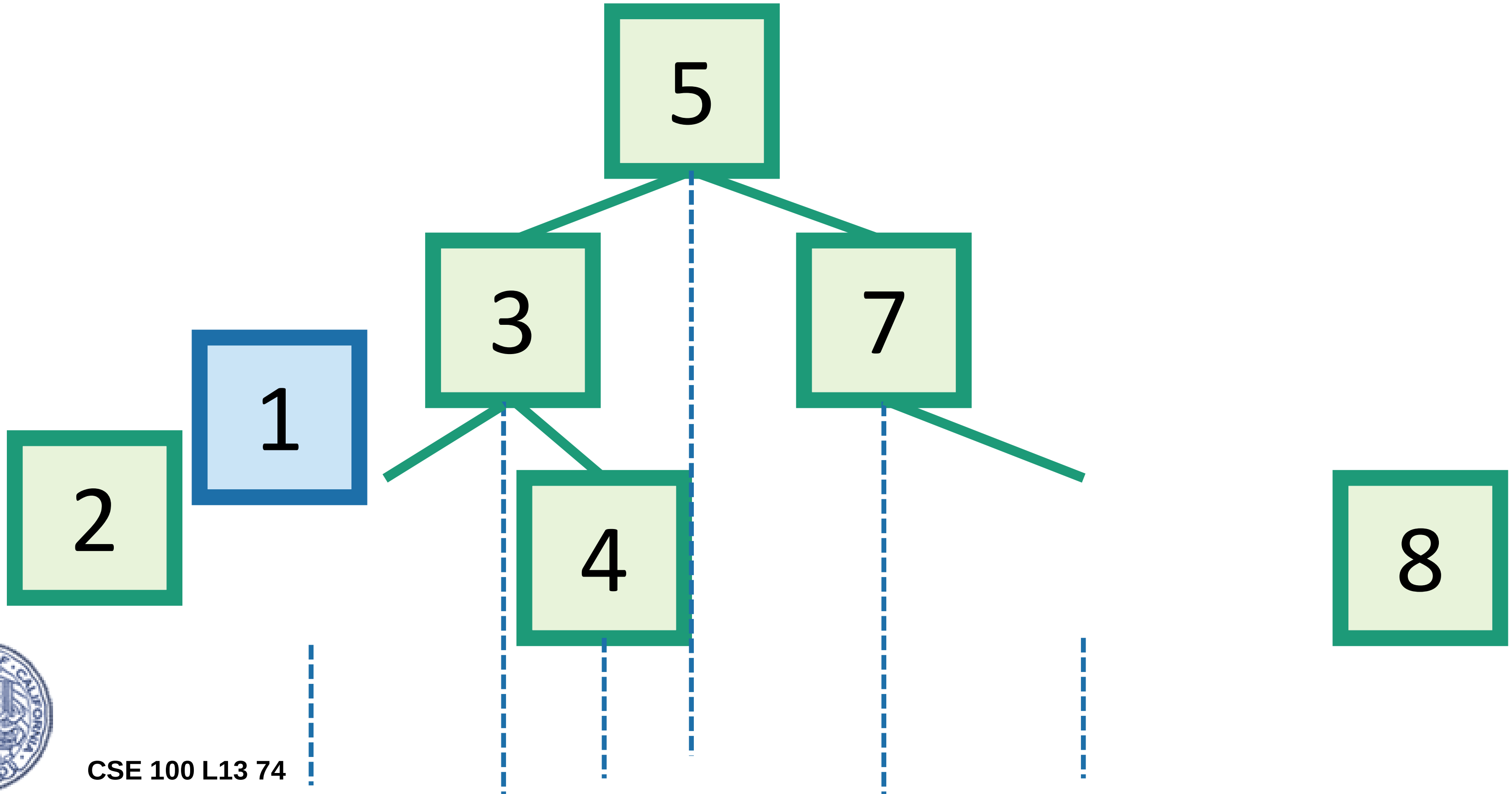
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



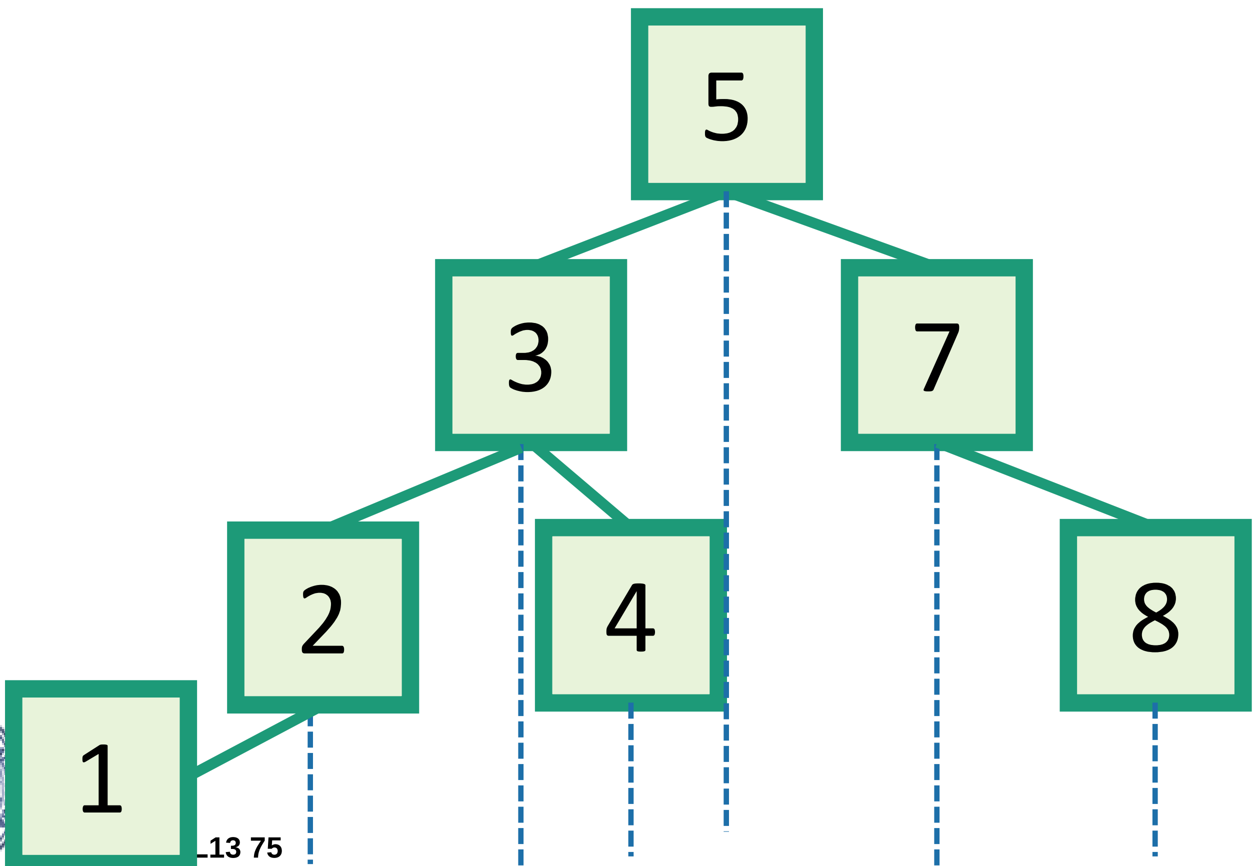
Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



Binary Search Trees

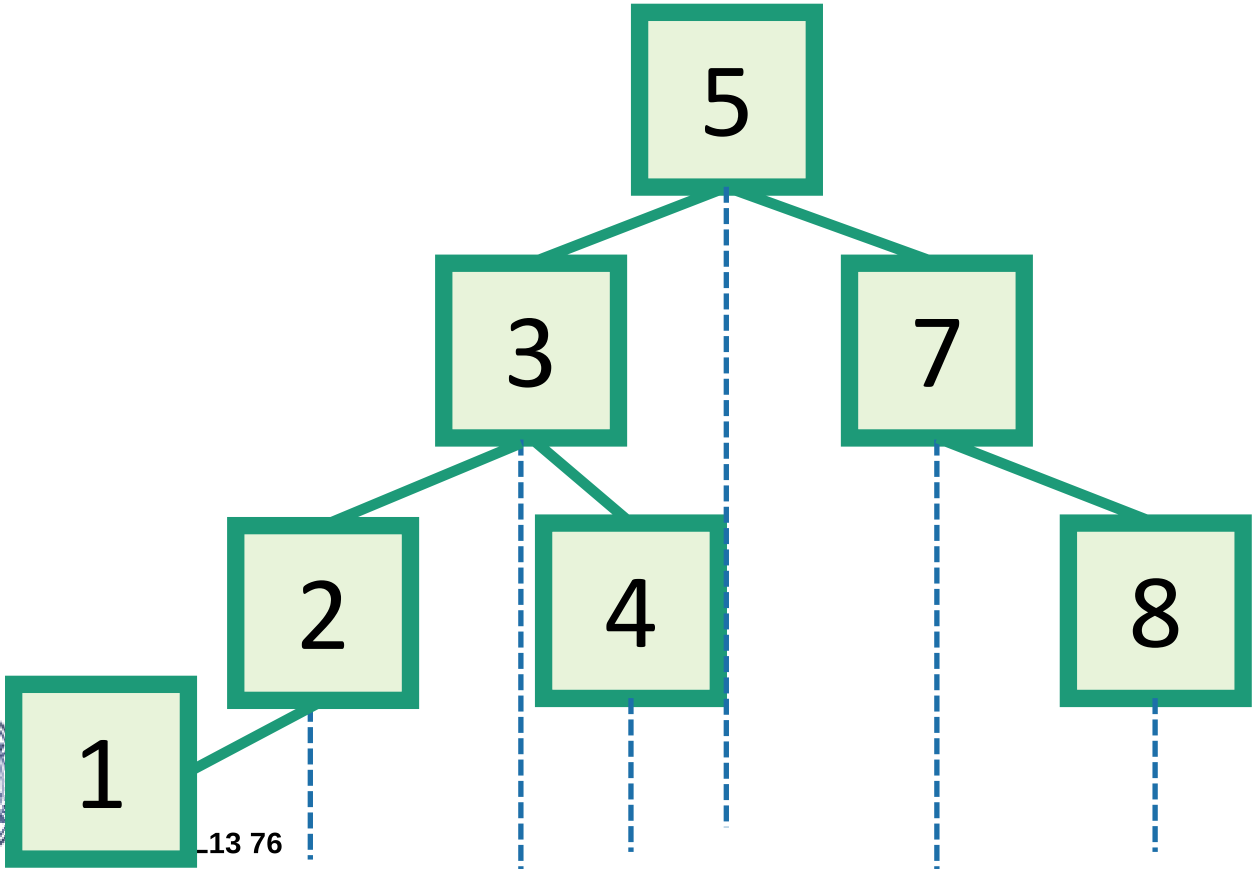
- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:



Q: Is this the only binary search tree I could possibly build with these values?

Binary Search Trees

- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.
- Example of building a binary search tree:

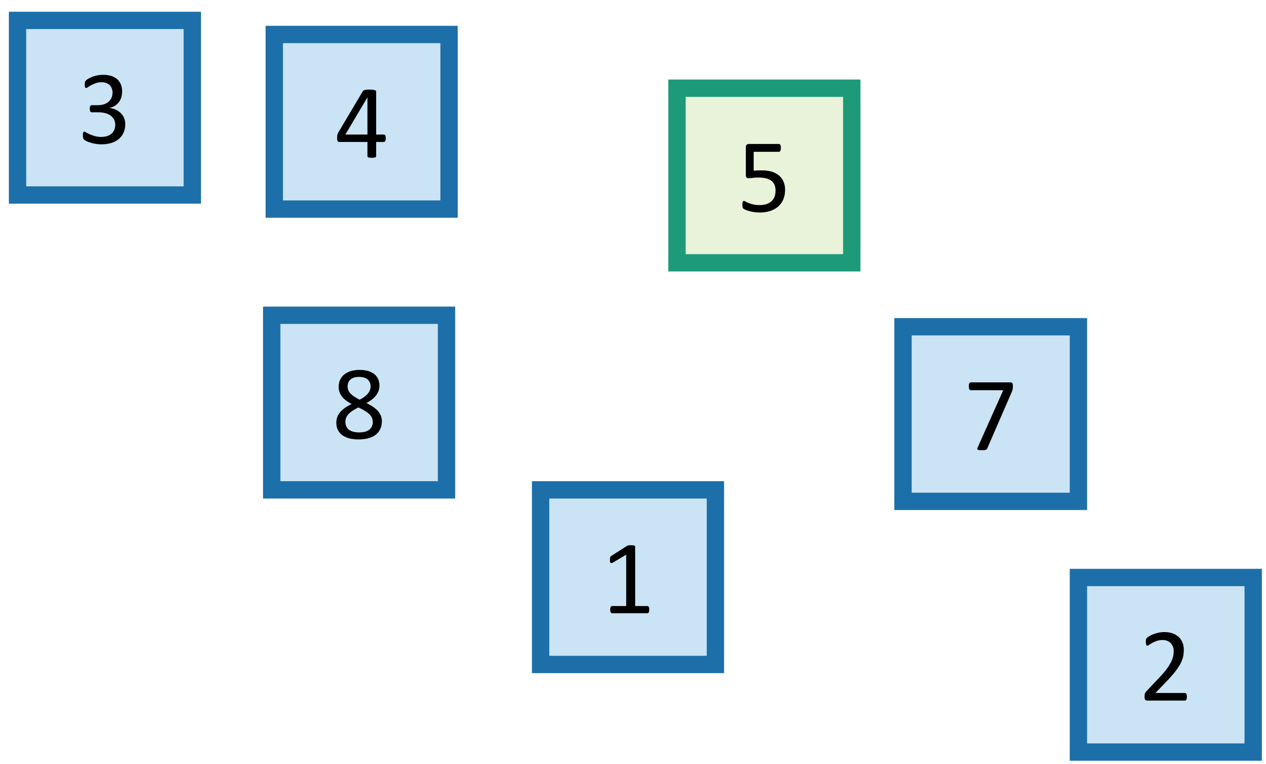


Q: Is this the only binary search tree I could possibly build with these values?

A: **No.** I made choices about which nodes to choose when. Any choices would have been fine.

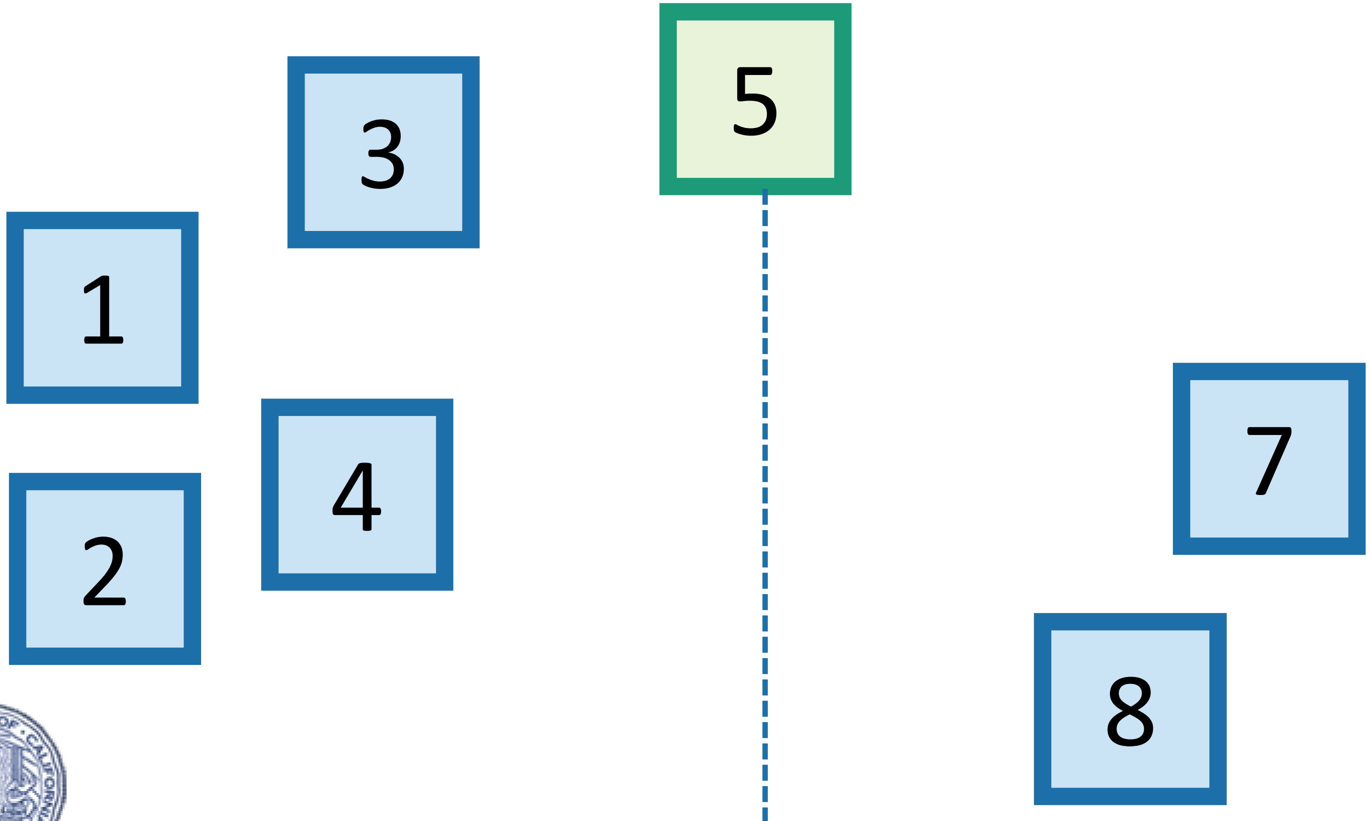
Aside: this should look familiar

kinda like QuickSort



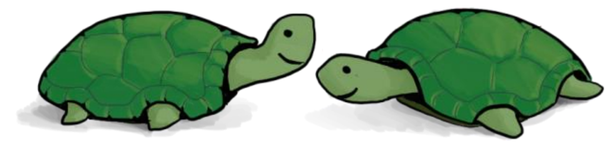
Aside: this should look familiar

kinda like QuickSort

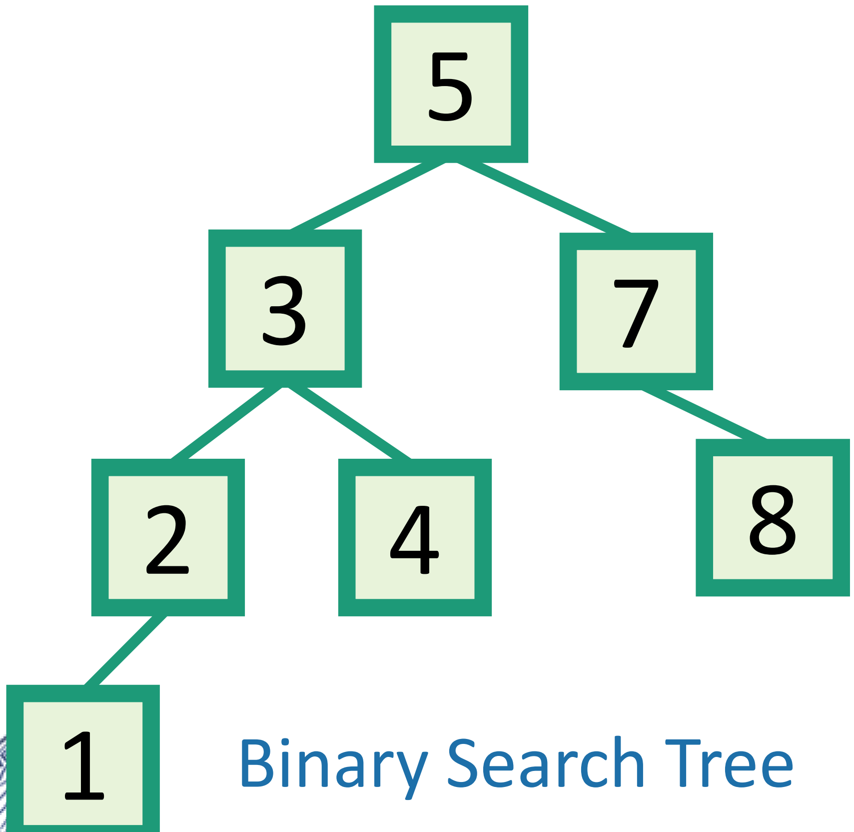


Binary Search Trees

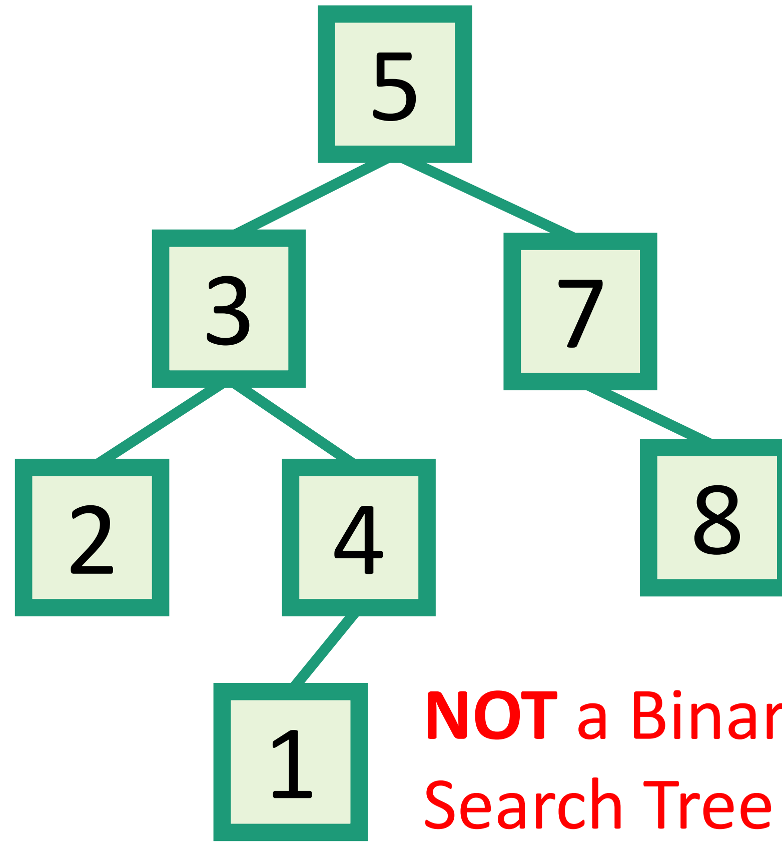
Which of these is a BST?



- A BST is a binary tree so that:
  - Every LEFT descendant of a node has key less than that node.
  - Every RIGHT descendant of a node has key larger than that node.



Binary Search Tree

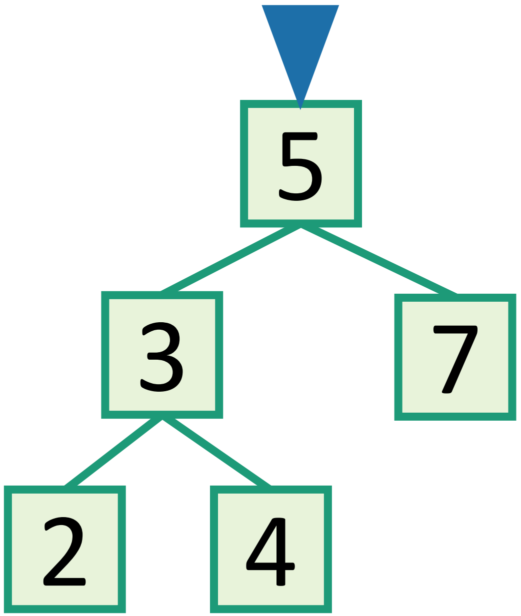


NOT a Binary Search Tree

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

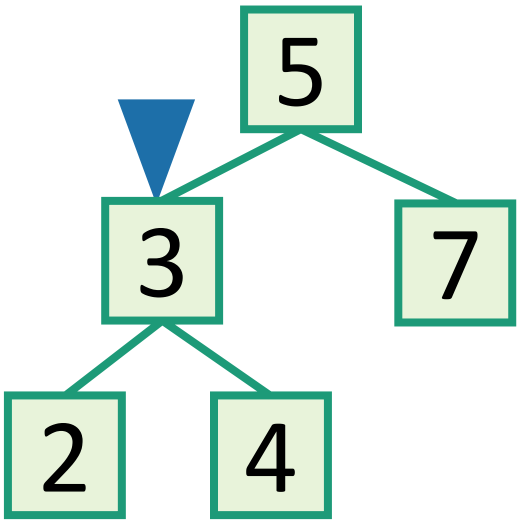
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

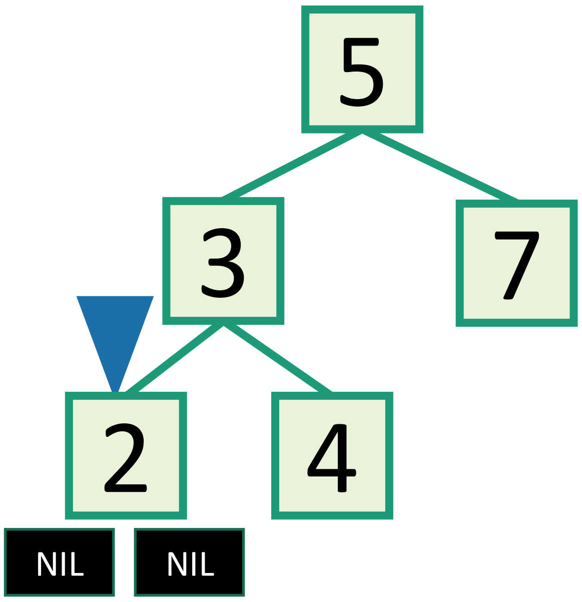
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

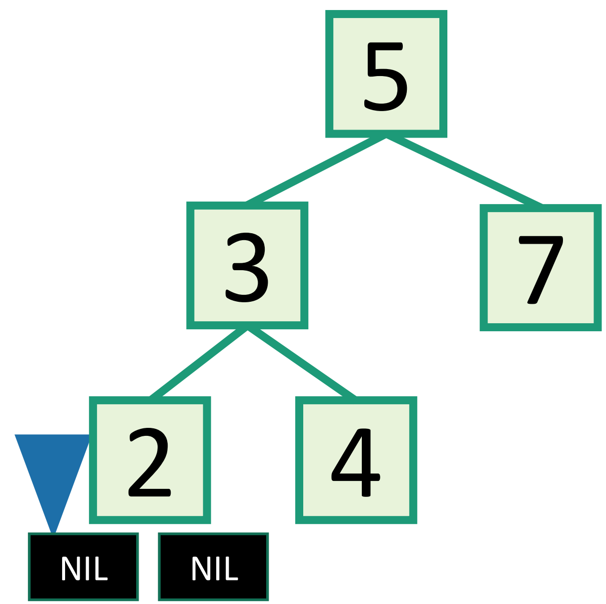
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

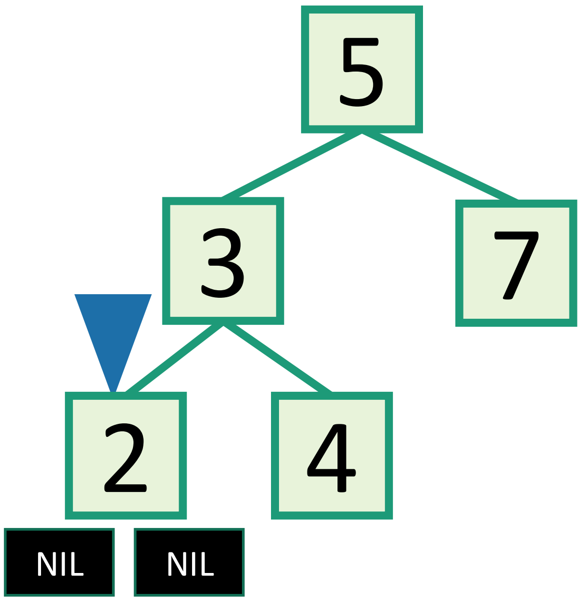
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)

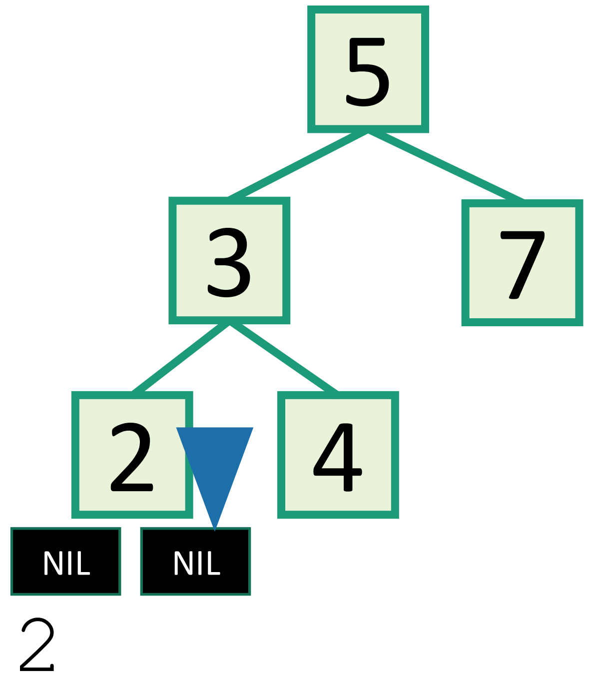


2

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

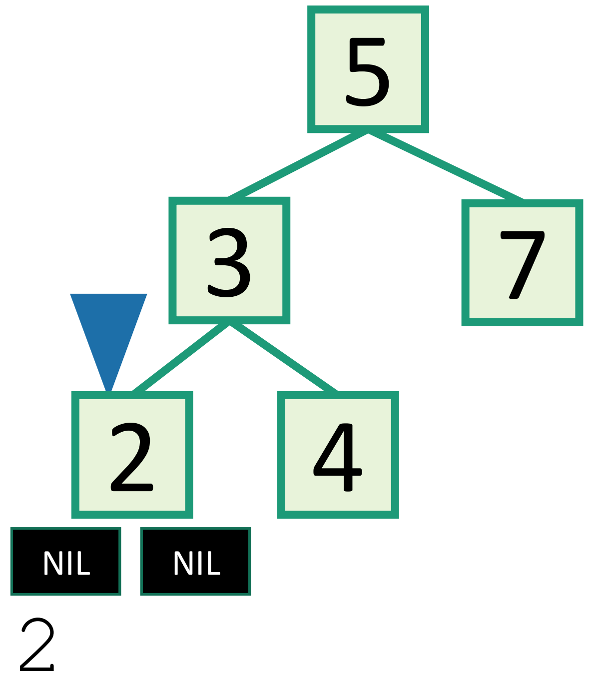
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

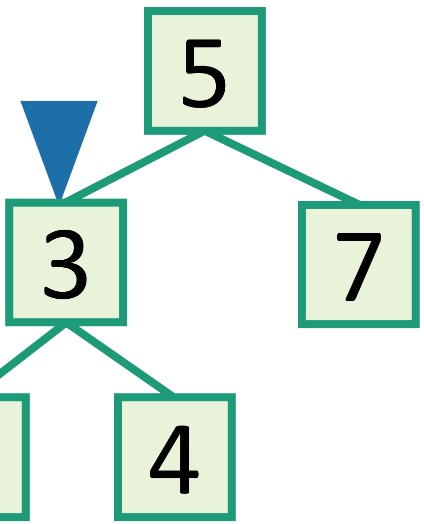
- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)

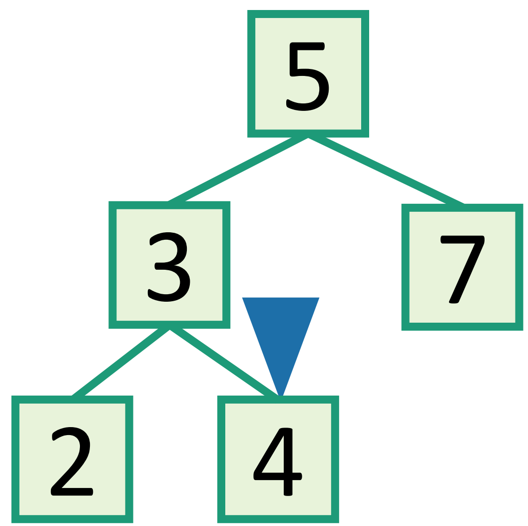


2 3

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)

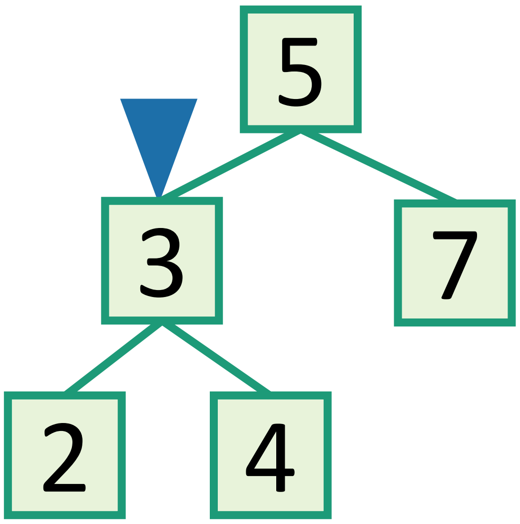


2 3 4

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)

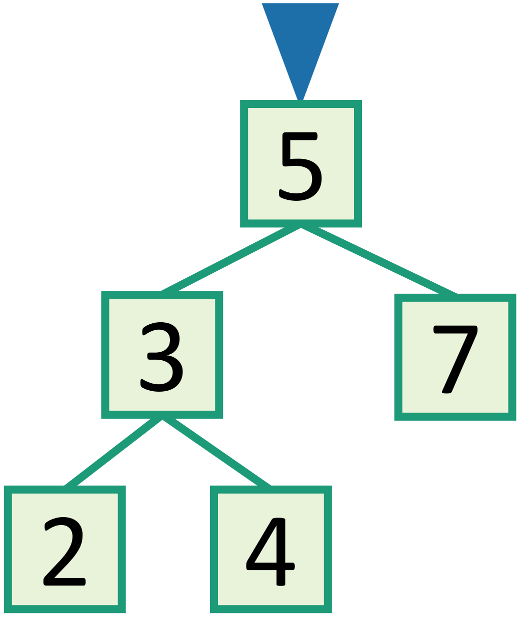


2 3 4

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)

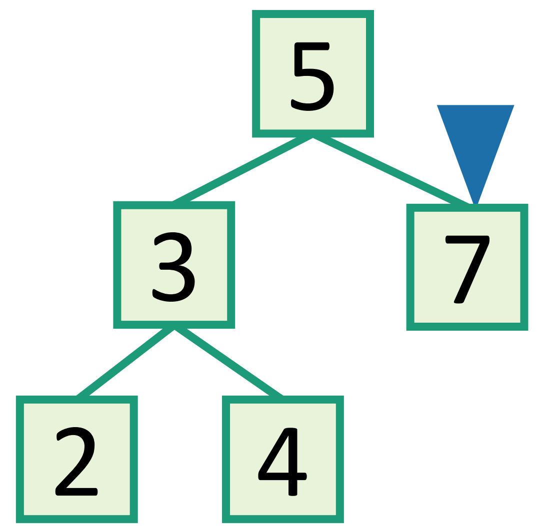


2 3 4 5

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
    - if x!= NIL:
        - inOrderTraversal(x.left)
        - print(x.key)
        - inOrderTraversal(x.right)

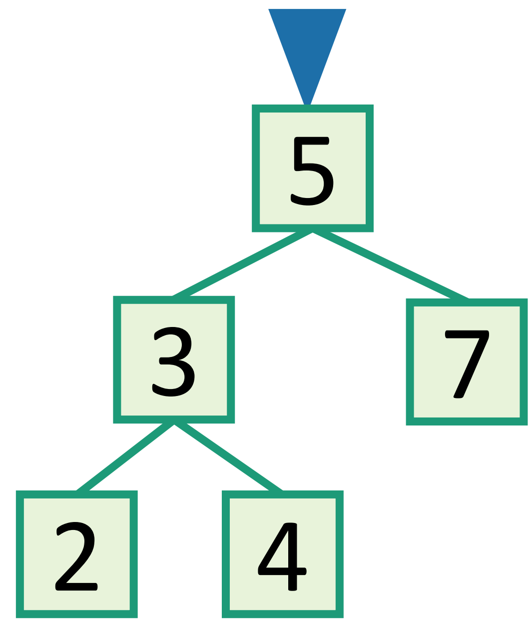


2 3 4 5 7

Aside: In-Order Traversal of BSTs

- Output all the elements in sorted order!

- inOrderTraversal(x):
  - if x!= NIL:
    - inOrderTraversal(x.left)
    - print(x.key)
    - inOrderTraversal(x.right)



- Runs in time O(n).

2 3 4 5 7 Sorted!

Back to the goal

Fast SEARCH/INSERT/DELETE

Can we do these?