We want to insert $n$ elements into a BST so that they will be stored in sorted order.

InOrder walk: 1 2 3 5 6 7 8
Binary Search Trees - Built Randomly

We want to insert n elements into a BST so that they will be stored in sorted order.

InOrder walk: 1 2 3 5 6 7 8

Insertion: nothing fancy. Just read elements and insert into current tree.
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5
Given array of elements: 3 1 8 2 6 7 5

This is a sorting algorithm.
Given the very simple BST-sort/construction algorithm

- how long can it take to build the BST?
- what shape will it have? How balanced? What depth?
Given the very simple BST-sort/construction algorithm
- how long can it take to build the BST?
- what shape will it have?  How balanced? What depth?

Depends on input sequence

1 2 3 4 5 → 1

Can be bad: $O(n)$ depth
$O(n^2)$ time
$\Omega(?)$
Given the very simple BST-sort/construction algorithm
- how long can it take to build the BST?
- what shape will it have? How balanced? What depth?

Depends on input sequence

1 2 3 4 5 → 1

Can be bad: $O(n)$ depth
$O(n^2)$ time
$\Omega(n\log n)$ worst-case time: sorting lower bound
Even for a balanced tree,

$\Theta(n) \approx \frac{n}{2}$ nodes have height $= \Theta(\log n)$

so it must take $\Omega(n \log n)$ time to build.
Even for a balanced tree,
\[ \Theta(n) \approx \frac{n}{2} \text{ nodes have height } = \Theta(\log n) \]
so it must take \( \Omega(n \log n) \) time to build.

Any algorithm producing any tree shape : \( \Omega(n \log n) \) time
Even for a balanced tree,
\( \Theta(n) \approx \frac{n}{2} \) nodes have height = \( \Theta(\log n) \)
so it must take \( \Omega(n\log n) \) time to build.

Any algorithm producing any tree shape : \( \Omega(n\log n) \) time

So... if lucky, \( \Theta(n\log n) \) time
if unlucky, \( O(n^2) \) time
Even for a balanced tree,
\( \Theta(n) \approx \frac{n}{2} \) nodes have height = \( \Theta(\log n) \)
so it must take \( \Omega(n \log n) \) time to build.

Any algorithm producing any tree shape : \( \Omega(n \log n) \) time

So ... if lucky, \( \Theta(n \log n) \) time
if unlucky, \( O(n^2) \) time

Sounds like...
Even for a balanced tree, $\Theta(n) \approx \frac{n}{2}$ nodes have height $= \Theta(\log n)$ so it must take $\Omega(n\log n)$ time to build.

Any algorithm producing any tree shape $: \Omega(n\log n)$ time

So ... if lucky, $\Theta(n\log n)$ time

if unlucky, $O(n^2)$ time

Sounds like... quicksort
Stable quicksort

1 8 2 6 7 5

Use first elt to partition → 1 2 3 8 6 7 5
Stable quicksort

- Use first elt to partition
- Repeat on each side
Stable quicksort

1 2 3 8 6 7 5

- Use first elt to partition
- Repeat on each side
- 3rd round
Stable quicksort

1 8 2 6 7 5

- Use first elt to partition
- Repeat on each side

3rd round

4th round
Stable quicksort

1 8 2 6 7 5

• use first elt to partition
  • repeat on each side
  • 3rd round

• 4th round

Same tree as BST
Stable quicksort

1 8 2 6 7 5

- Use first elt to partition
- Repeat on each side

quicksort round 1: compare all elts to 3
BST sort: 3 = root; eventually all elts pass through.

3rd round

4th round

Same tree as BST
Stable quicksort

1 8 2 6 7 5

- Use first elt to partition
- Repeat on each side

quicksort round 1: compare all els to 3
BST sort: 3 = root; eventually all els pass through.
quicksort: partitions into 2 groups < 3 & > 3
each is independent

BST sort: same

Same tree as BST
Stable quicksort

1 8 2 6 7 5

- Use first elt to partition
- Repeat on each side

3rd round

4th round

Same tree as BST

Quicksort round 1: compare all els to 3
BST sort: 3 = root; eventually all els pass through.

Quicksort: partitions into 2 groups
<③ & >③
each is independent

BST sort: same
exactly same comparisons
but in different order