SORTING WITH PREDICTIONS

XINGJIAN BAI (MIT)
CHRISTIAN COESTER (OXFORD)

Motivation: Example

Consider a real-world comparison-based sorting problem:

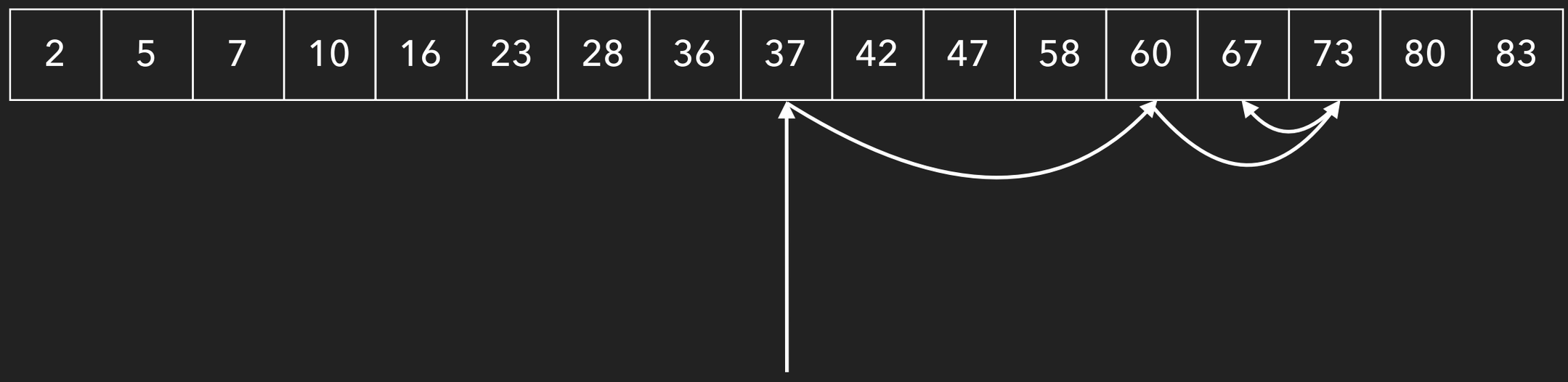
- Ranking 100 molecular structures for a potential vaccine

Current Approaches:

- (Pair-wise) Clinical trials:
  - Requires ~700 trails
  - Provides provably correct ranking
- Biomedical ML models:
  - Very cheap to obtain predictions
  - May produce erroneous rankings

- But we can leverage both tools to achieve efficiency and accuracy!
- Target: ~150 trails, but still get provably correct results

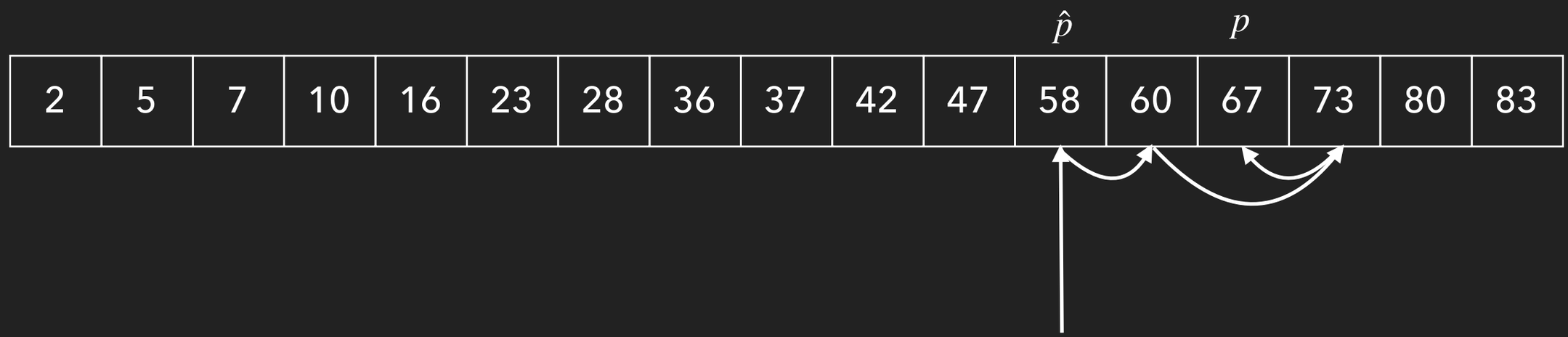
Example: Binary Search [Lykouris,Vassilvitskii 18] [Kraska et al. 18]



Does 67 appear in array?

- Binary search: Time $O(\log n)$

Example: Binary Search [Lykouris,Vassilvitskii 18] [Kraska et al. 18]



Does 67 appear in array?

- Binary search: Time $O(\log n)$
- Given prediction $\hat{p}$ of position p
  - Time $O(\log \eta)$, where $\eta = |\hat{p} - p|$

Sorting with Predictions [Bai,Coester 23]

Task: Sort $a_1, a_2, \ldots, a_n$ wrt. $<$

Setting 1: Receive prediction of positions in sorted list

Setting 2: Access to quick-and-dirty comparisons

Sorting with Positional Predictions

Input: $a_1, a_2, \ldots, a_n$

prediction $\hat{p}(i)$ of a_i's position in sorted list

Similar: Adaptive Sorting

We consider element-wise error ⇝ fine-grained guarantees

Notation: $p(i) =$ true position of a_i in sorted list

$$\eta_i = |\hat{p}(i) - p(i)|$$

Theorem: ∃ algorithm that sorts in time $O\left(\sum_{i=1}^{n} \log(\eta_i + 2)\right)$

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	82		208	281		364		510			621	711		813		914
	67					398		491			649					894
	90					385					625					

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	82		208	281		364		510			621	711		813		914
	67					398		491			649					894
						385					625					

90

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	82		208	281		364		510			621	711		813		914
	67					398		491			649					894
						385					625					

90

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	82		208	281		364		510			621	711		813		914
						398		491			649					894
						385					625					

67	90

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	82		208	281		364		510			621	711		813		914
						398		491			649					894
						385					625					

67	90

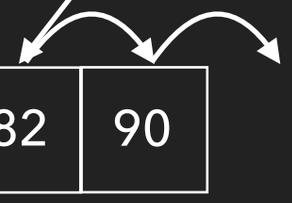
First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
			208	281		364		510			621	711		813		914
						398		491			649					894
						385					625					



67	82	90

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
				281		364		510			621	711		813		914
						398		491			649					894
						385					625					

67	82	90	208

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
						364		510			621	711		813		914
						398		491			649					894
						385					625					

67	82	90	208	281

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
						364		510			621	711		813		914
						398		491			649					894
											625					

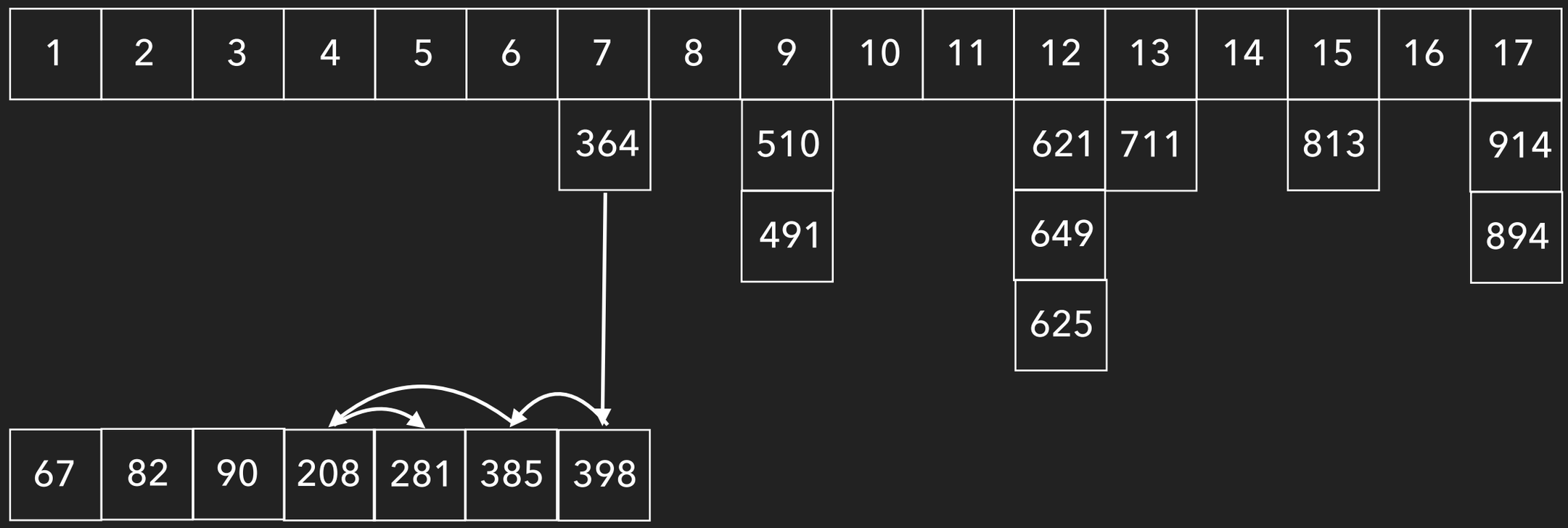
67	82	90	208	281	385

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12



First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
						364		510			621	711		813		914
								491			649					894
											625					

67	82	90	208	281

385	398

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

a_i	510	82	208	813	67	491	621	364	914	398	649	281	711	385	90	894	625
$\hat{p}(i)$	9	2	4	15	2	9	12	7	17	7	12	5	13	7	2	17	12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
								510			621	711		813		914
								491			649					894
											625					

67	82	90	208	281	364	385	398

First algorithm:

1. Bucket sort according to $\hat{p}$

2. From left to right: Insert into sorted list
   Use binary search with predictions to find insert position

WLOG: $\hat{p}(1) \leq \hat{p}(2) \leq \ldots \leq \hat{p}(n)$

#comparisons $\leq \ldots$

$$\leq \ldots$$

$$\leq O\left(\sum_{i=1}^{n} \log(\eta_i + 2)\right)$$

But shifting subarrays slow

Better: Replace array by BBST to get time $O\left(\sum_i \log(\eta_i + 2)\right)$

DoubleHoover Sort [Bai,Coester 23]

Theorem: Can sort with $O\left(\sum_i \log(\tilde{\eta}_i + 2)\right)$ comparisons, where

$$\tilde{\eta}_i := \min\Big\{\#\{j\colon a_j < a_i, \hat{p}(j) \geq \hat{p}(i)\},\\ \#\{j\colon a_j > a_i, \hat{p}(j) \leq \hat{p}(i)\}\Big\}$$

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

a_i	69	28	82	67	49	71	64	38	9	81

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

a_i	69	28	82	67	49	71	64	38	9	81

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Round 1

a_i	69	28	82	67	49	71	64	38	9	81

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

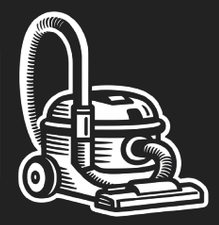
Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Round 1

9	81

69	82



a_i		28		67	49	71	64	38		

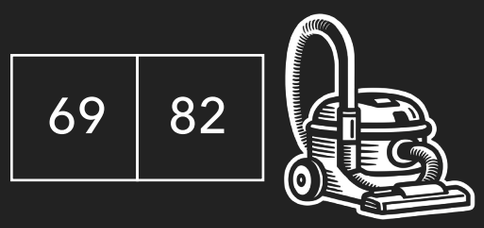
DoubleHoover Sort

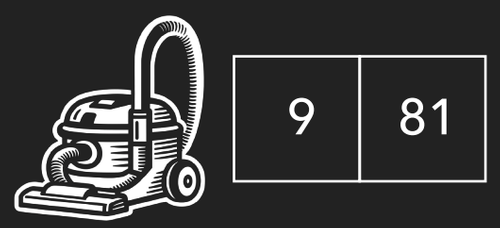
Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Round 2





a_i		28		67	49	71	64	38		

DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Round 2

9	38	81

28	69	71	82

a_i				67	49		64			

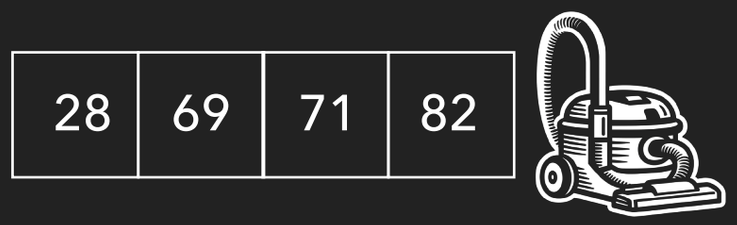
DoubleHoover Sort

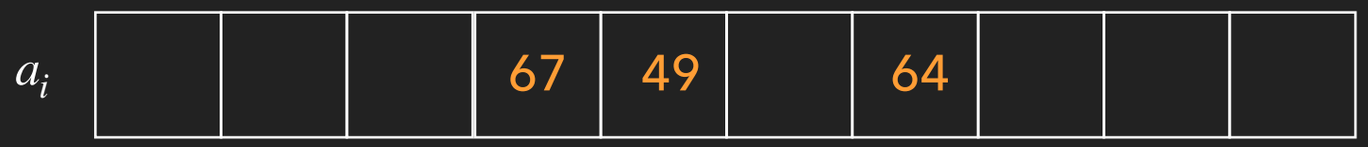
Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Round 3





DoubleHoover Sort

Idea: Bucket Sort the items w.r.t. $\hat{p}(i)$

Two "hoovers", L and R, scan through the array repeatedly in $\log(n)$ rounds

In round i, each hoover sucks in items that cost i comparisons to be inserted.

Finally, combine items in both hoovers.

Each a_i is sucked into a hoover before round $\log \tilde{\eta}_i$

Sorting with Dirty and Clean Comparisons

Input: $a_1, a_2, \ldots, a_n$

slow-and-clean comparator $<$

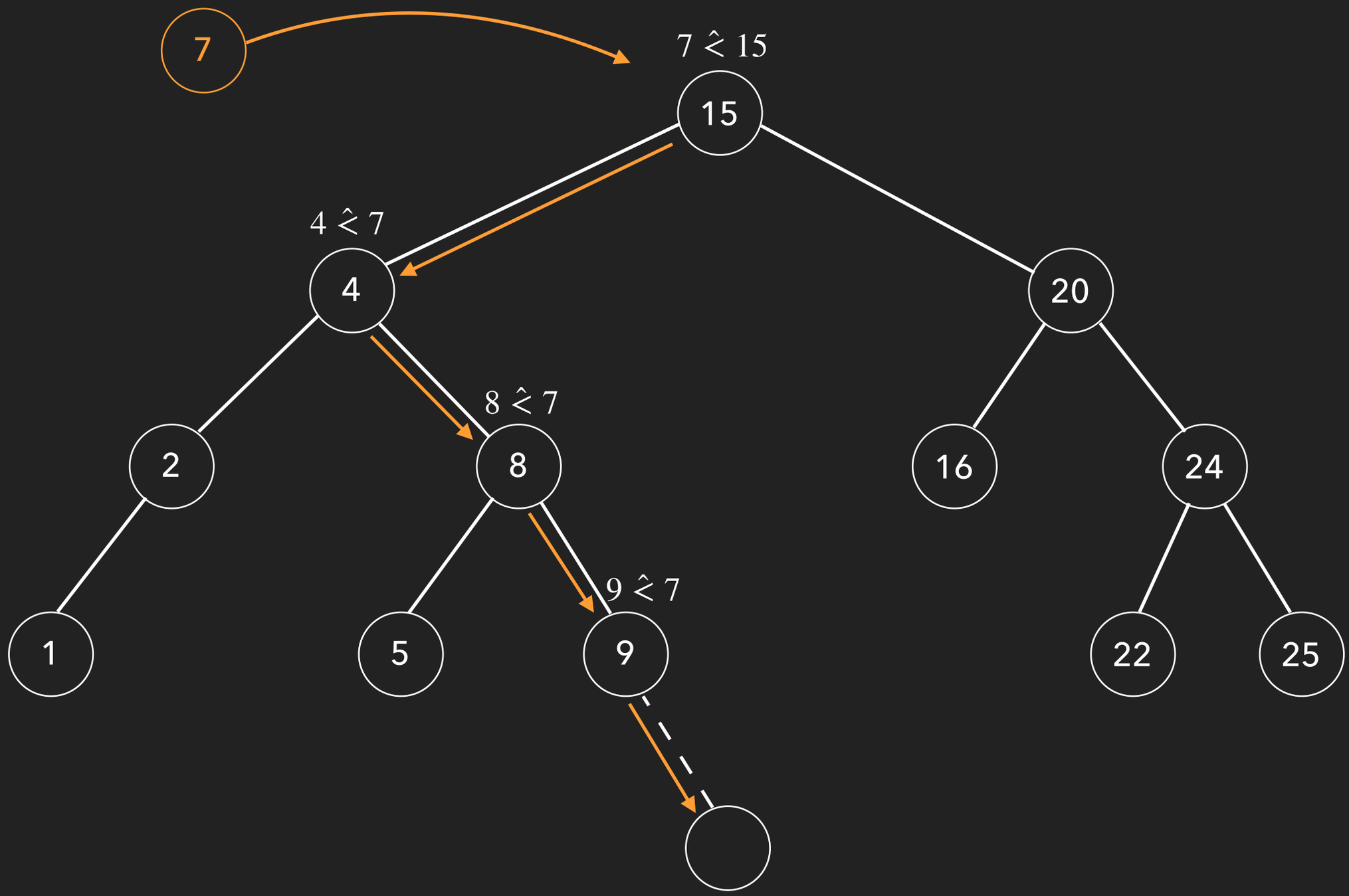
quick-and-dirty comparator $\hat{<}$

Error: $\eta_i := \#\{j\colon (a_j < a_i) \neq (a_j \mathrel{\hat{<}} a_i)\}$

Theorem: Can sort with $O(n \log n)$ dirty comparisons and $O\left(\sum_{i=1}^{n} \log(\eta_i + 2)\right)$ clean comparisons

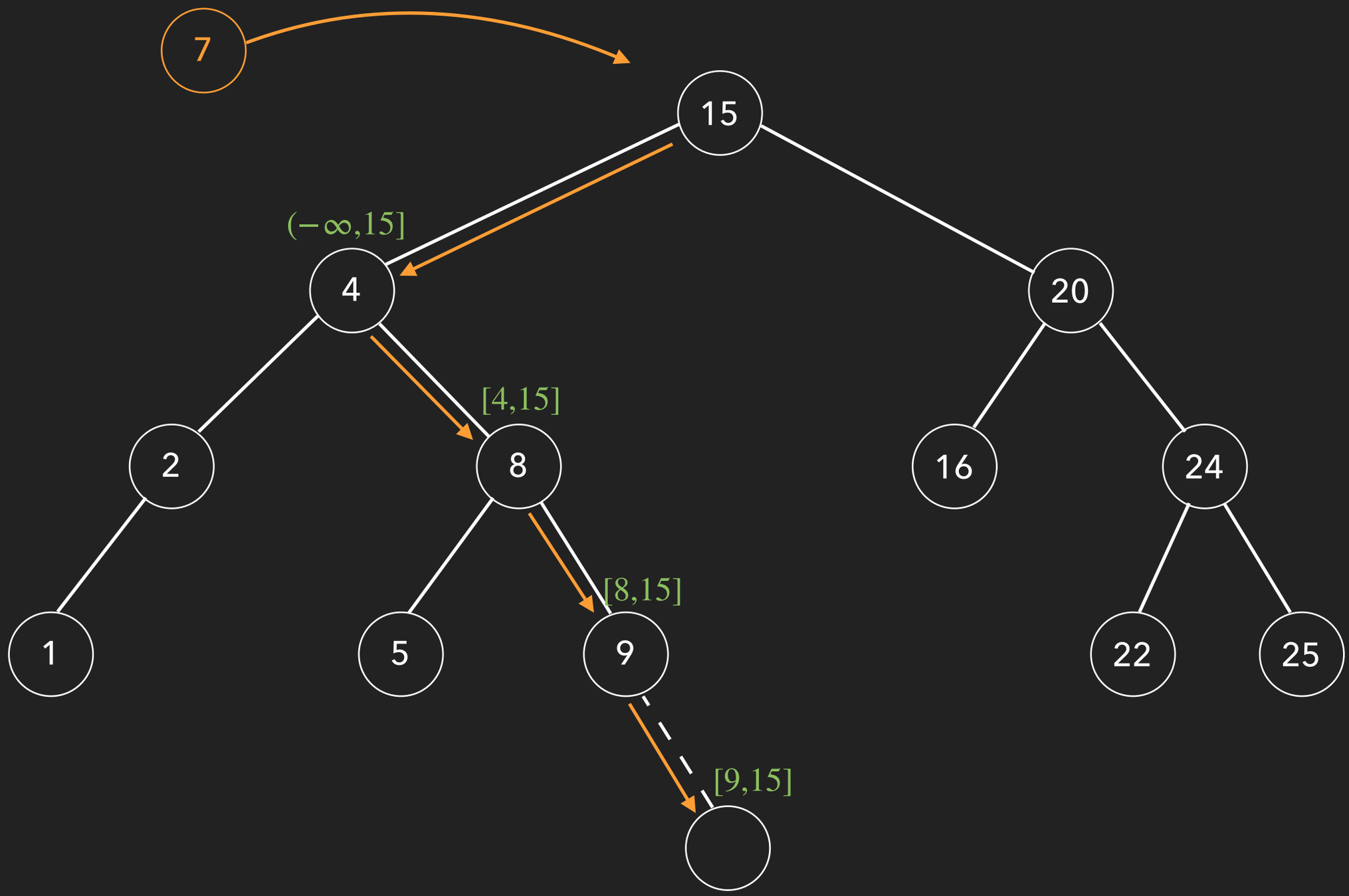
Idea: Build BST wrt. $<$

Guide insertions via $\hat{<}$ and $<$



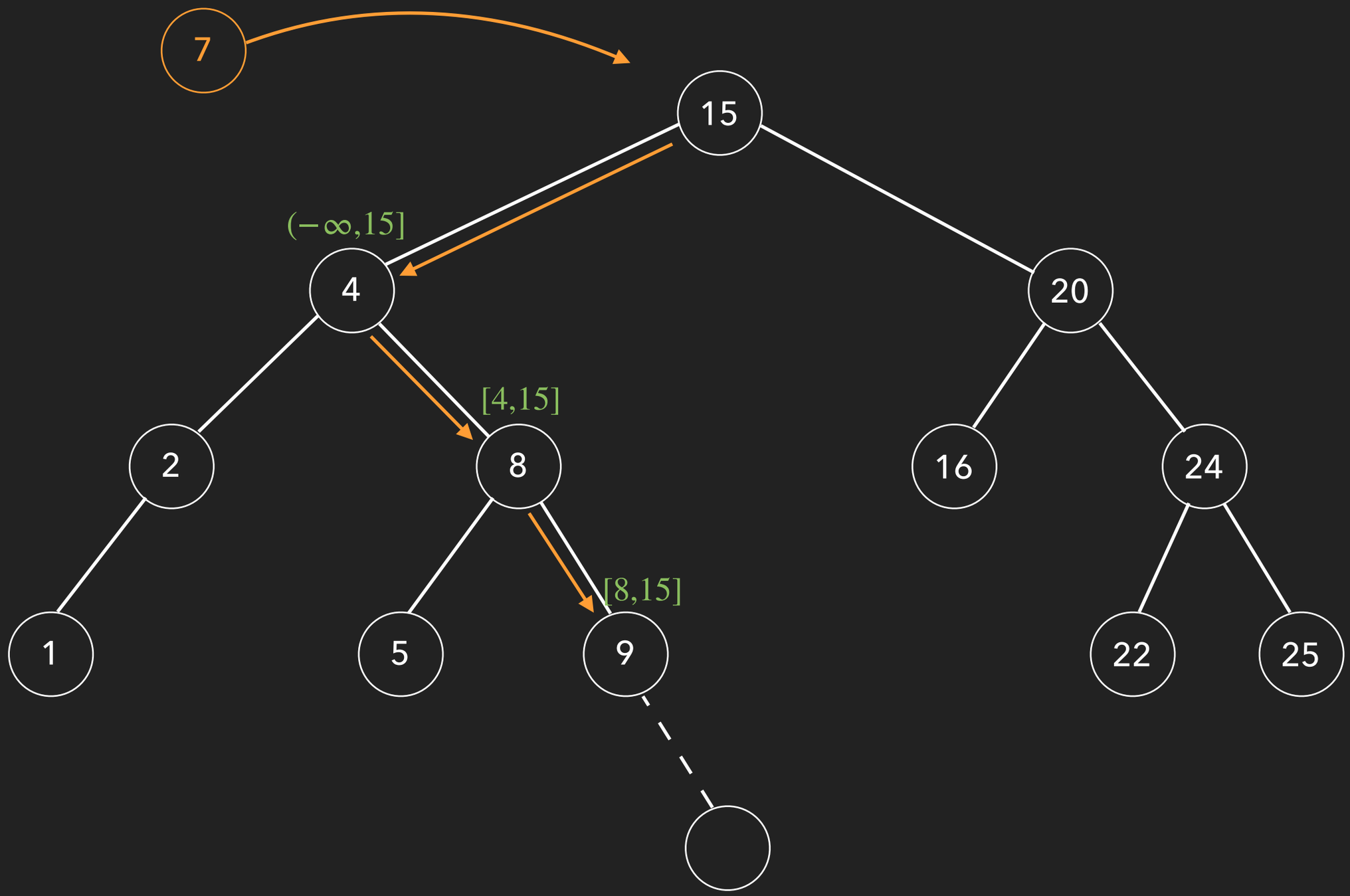
Idea: Build BST wrt. $<$

Guide insertions via $\hat{<}$ and $<$



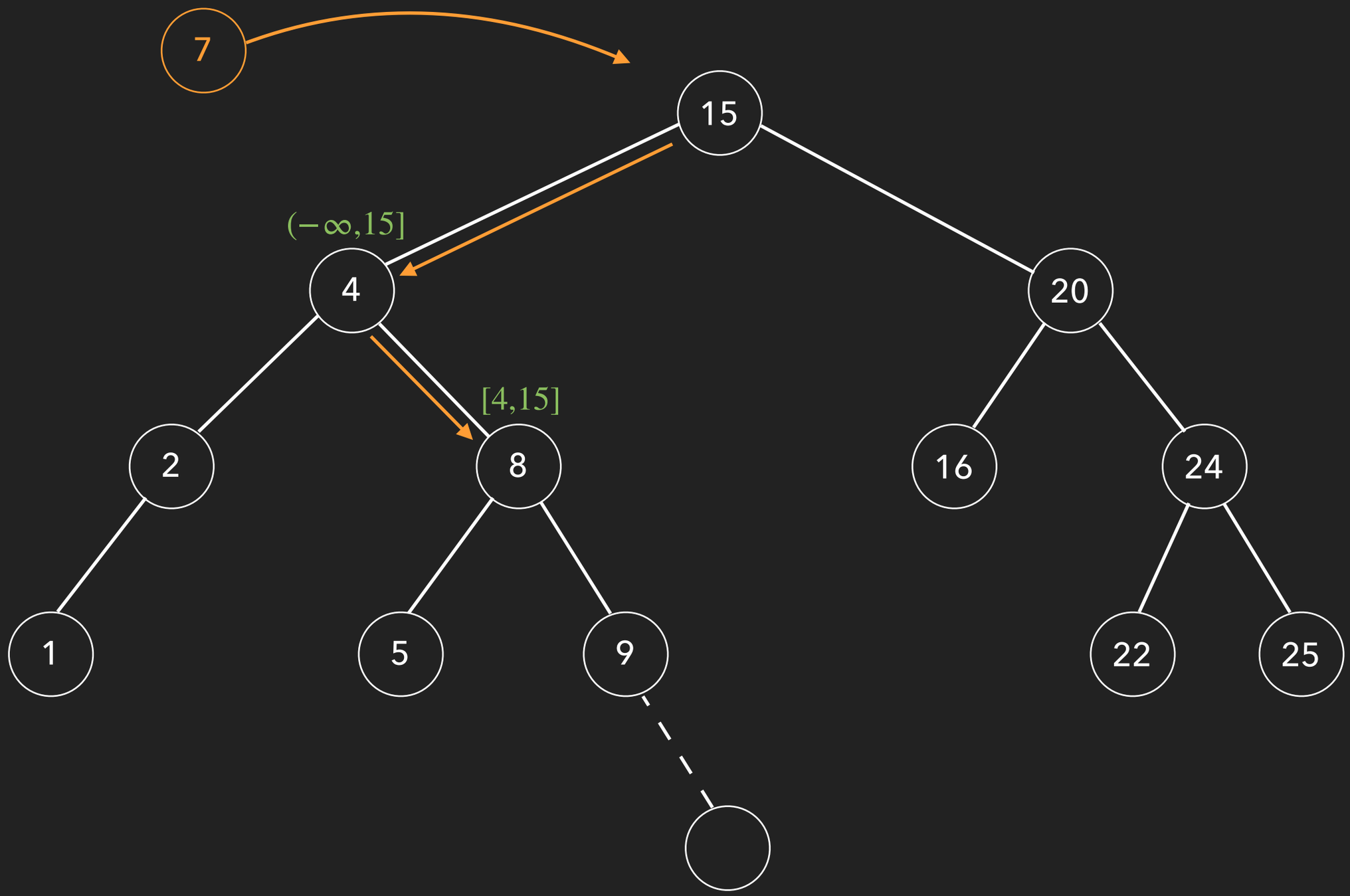
Idea: Build BST wrt. $<$

Guide insertions via $\hat{<}$ and $<$



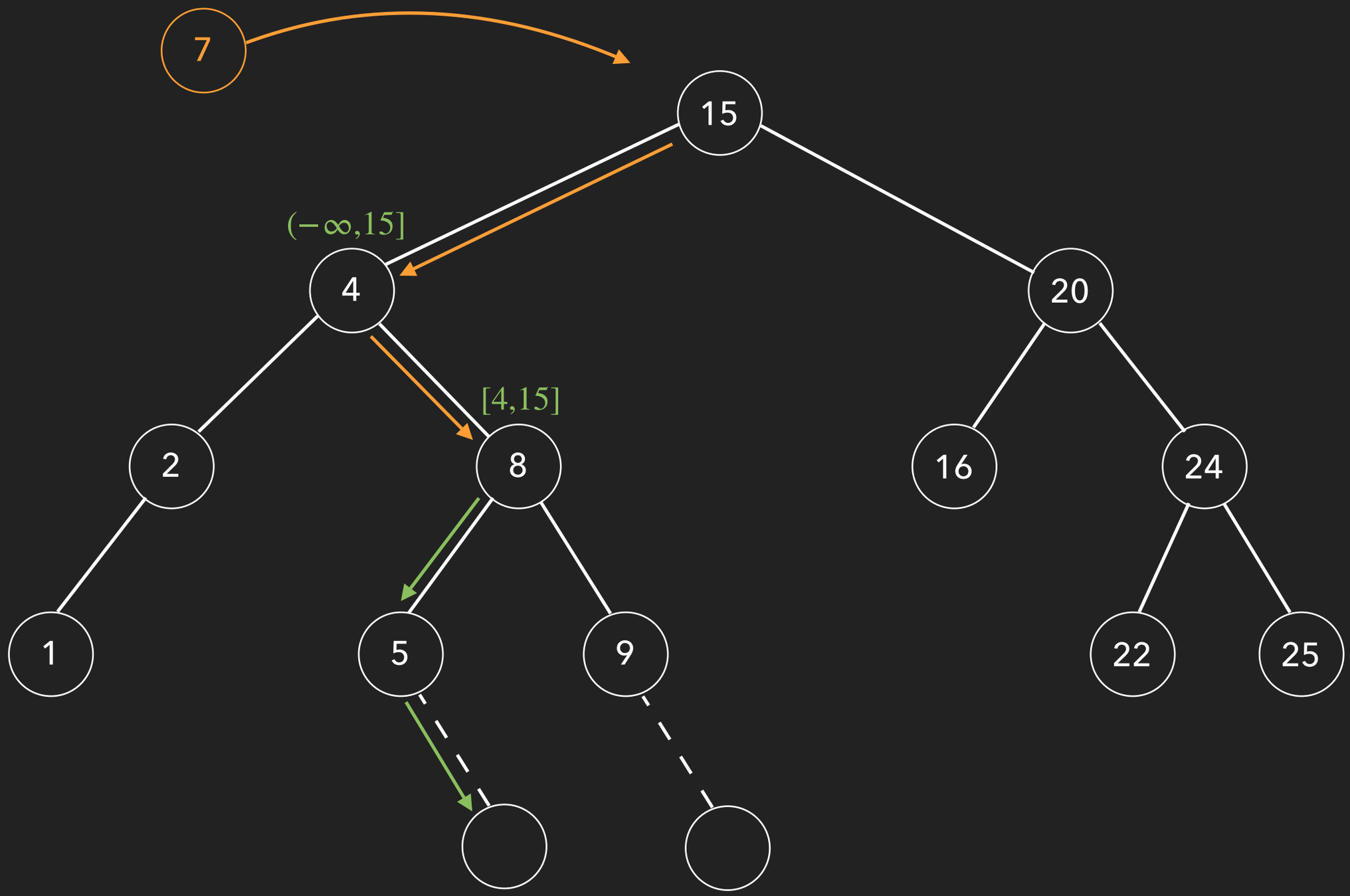
Idea: Build BST wrt. $<$

Guide insertions via $\hat{<}$ and $<$



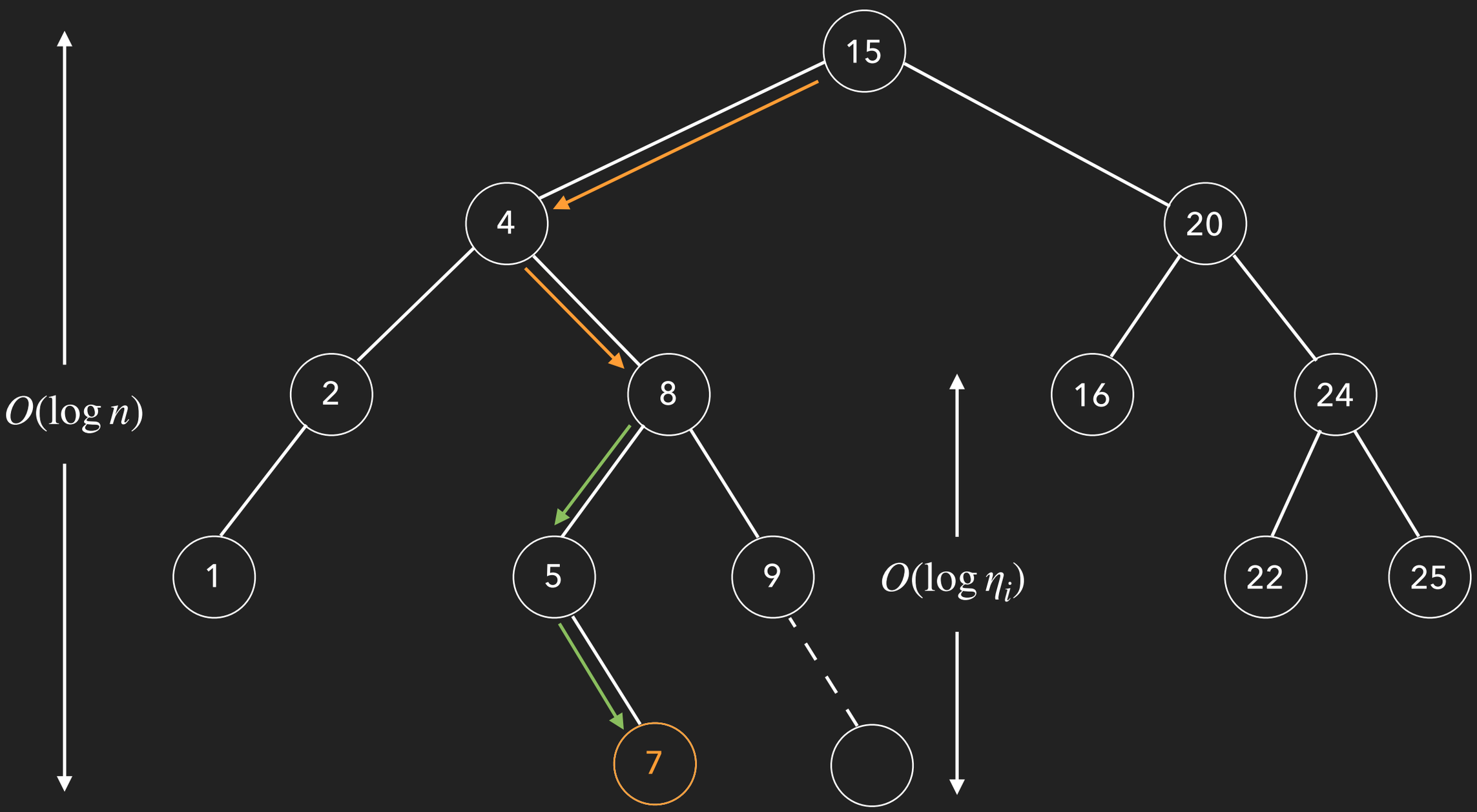
Idea: Build BST wrt. $<$

Guide insertions via $\hat{<}$ and $<$



Idea: Build BST wrt. $<$

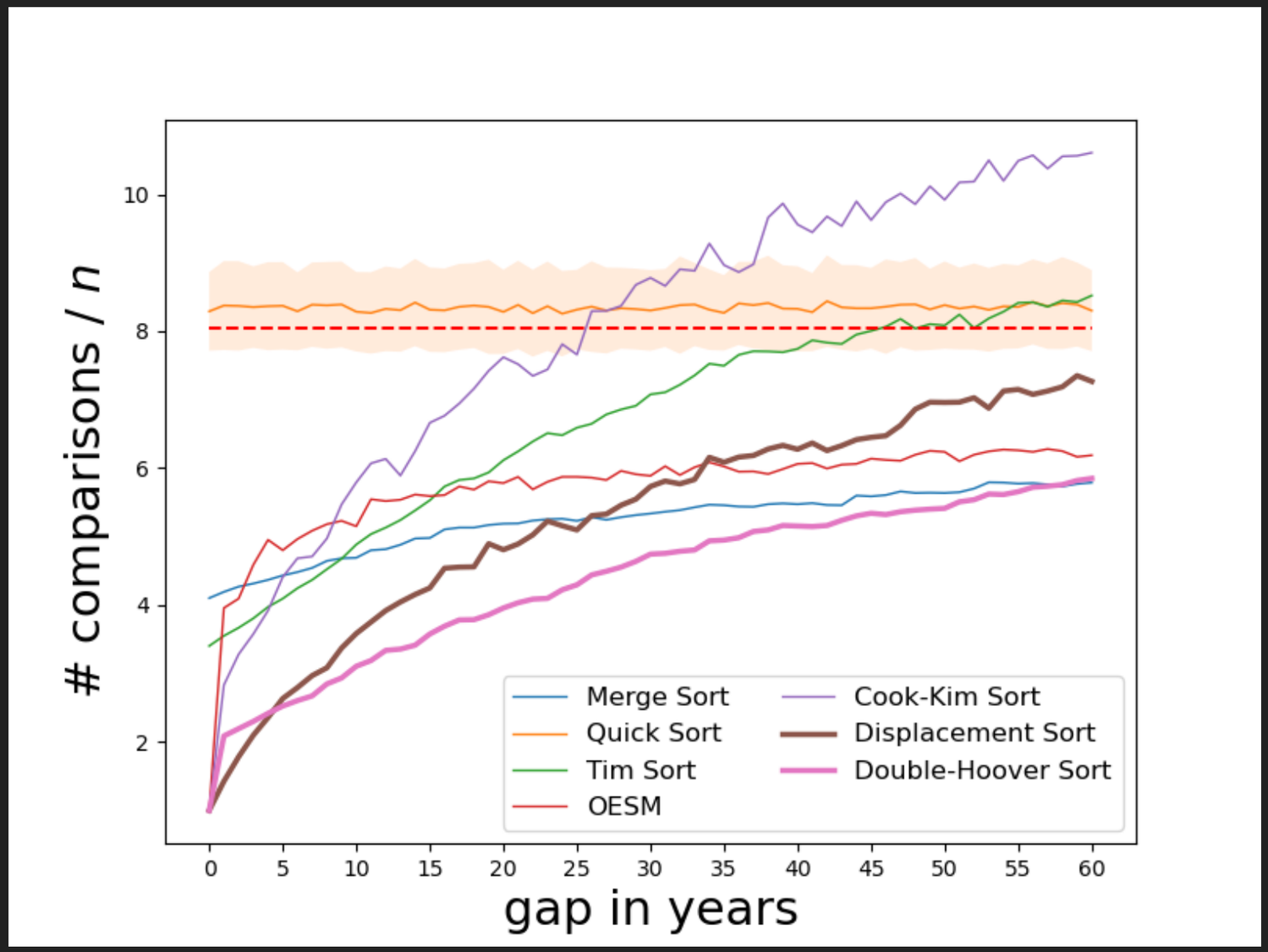
Guide insertions via $\hat{<}$ and $<$



Experiments

Sorting countries by population (n=261)

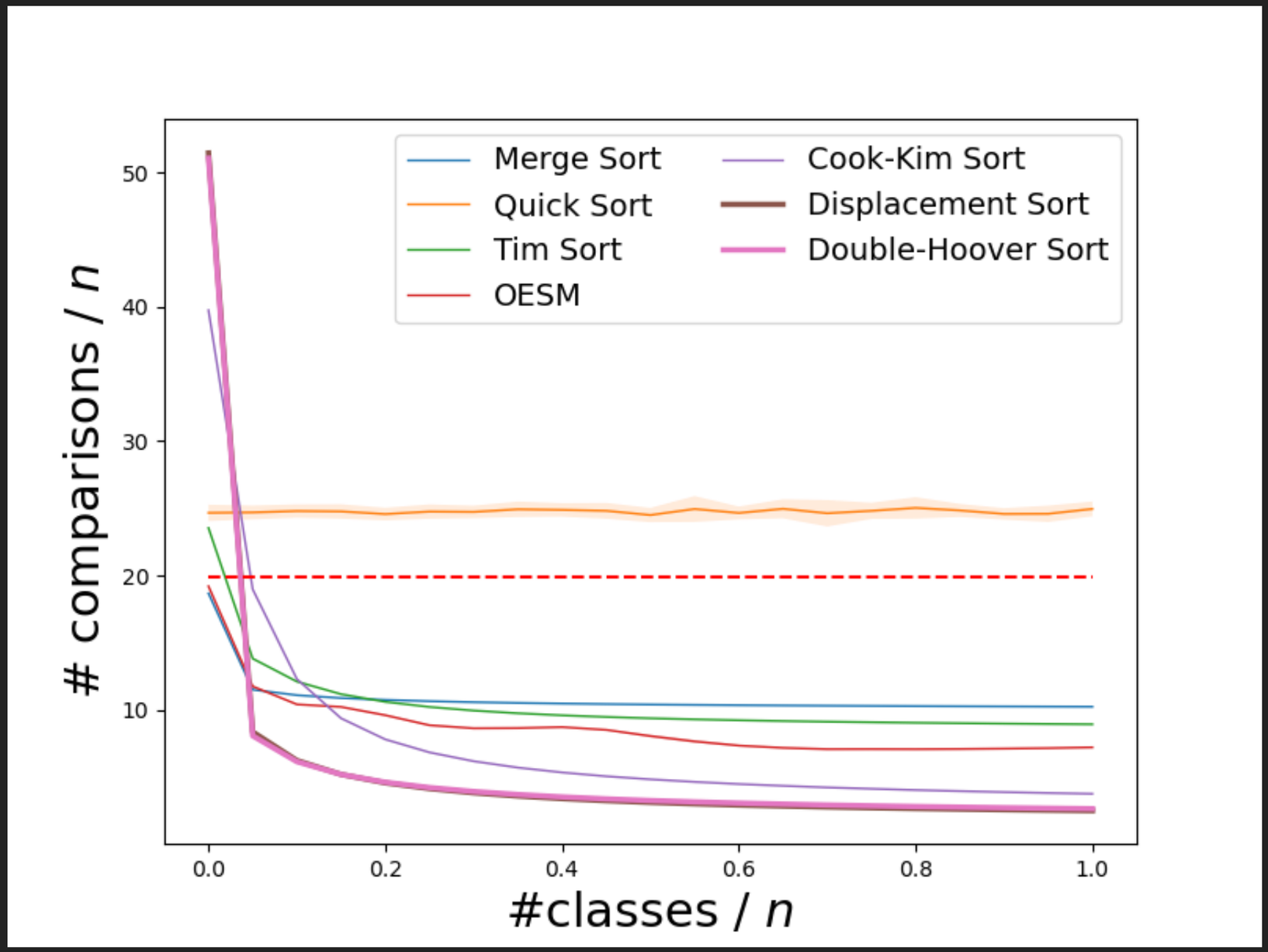
Predictions: ranking x years ago



Experiments

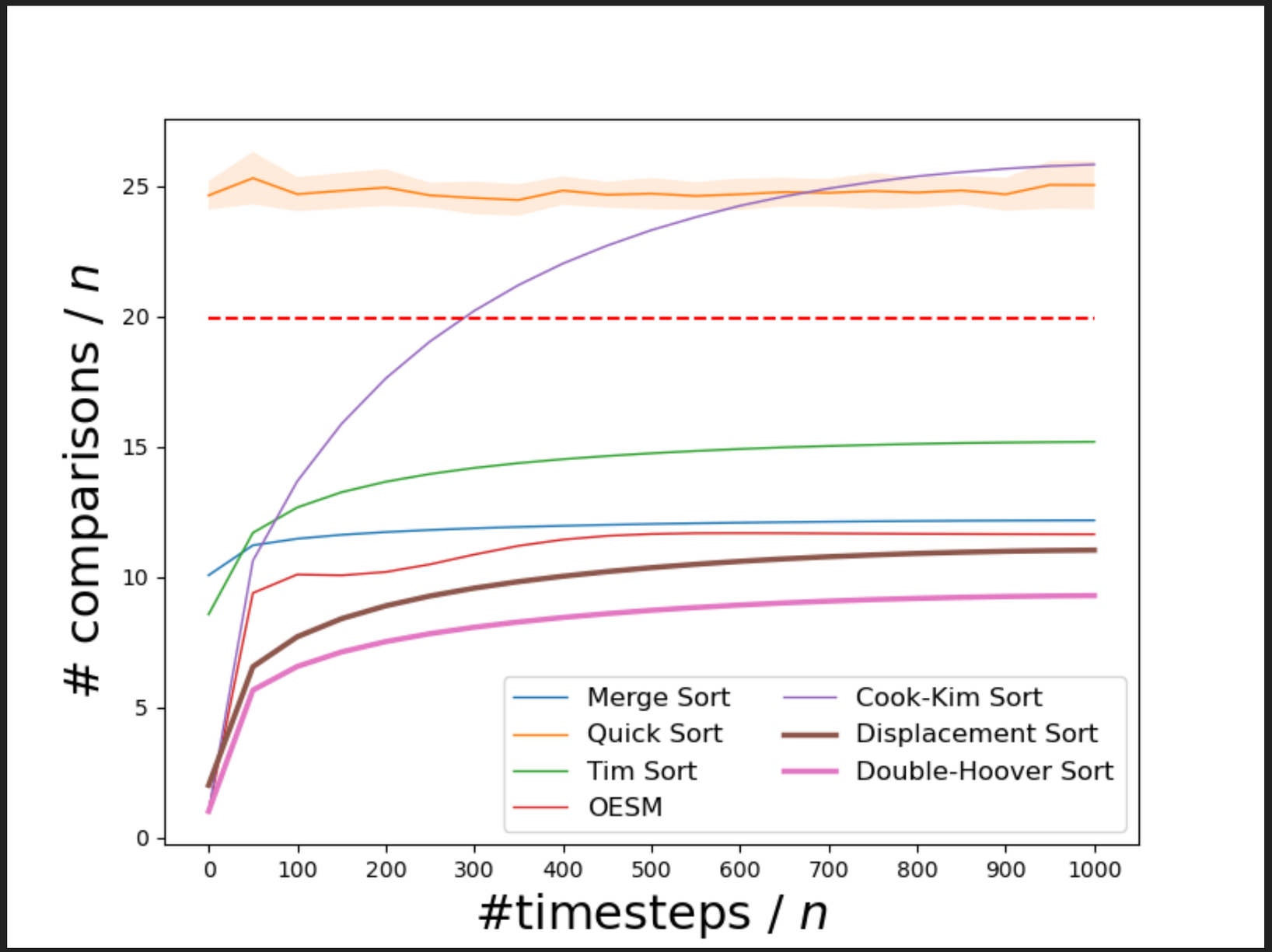
Classes of consecutive items (n=1,000,000)

Predictions: random position within class



Experiments

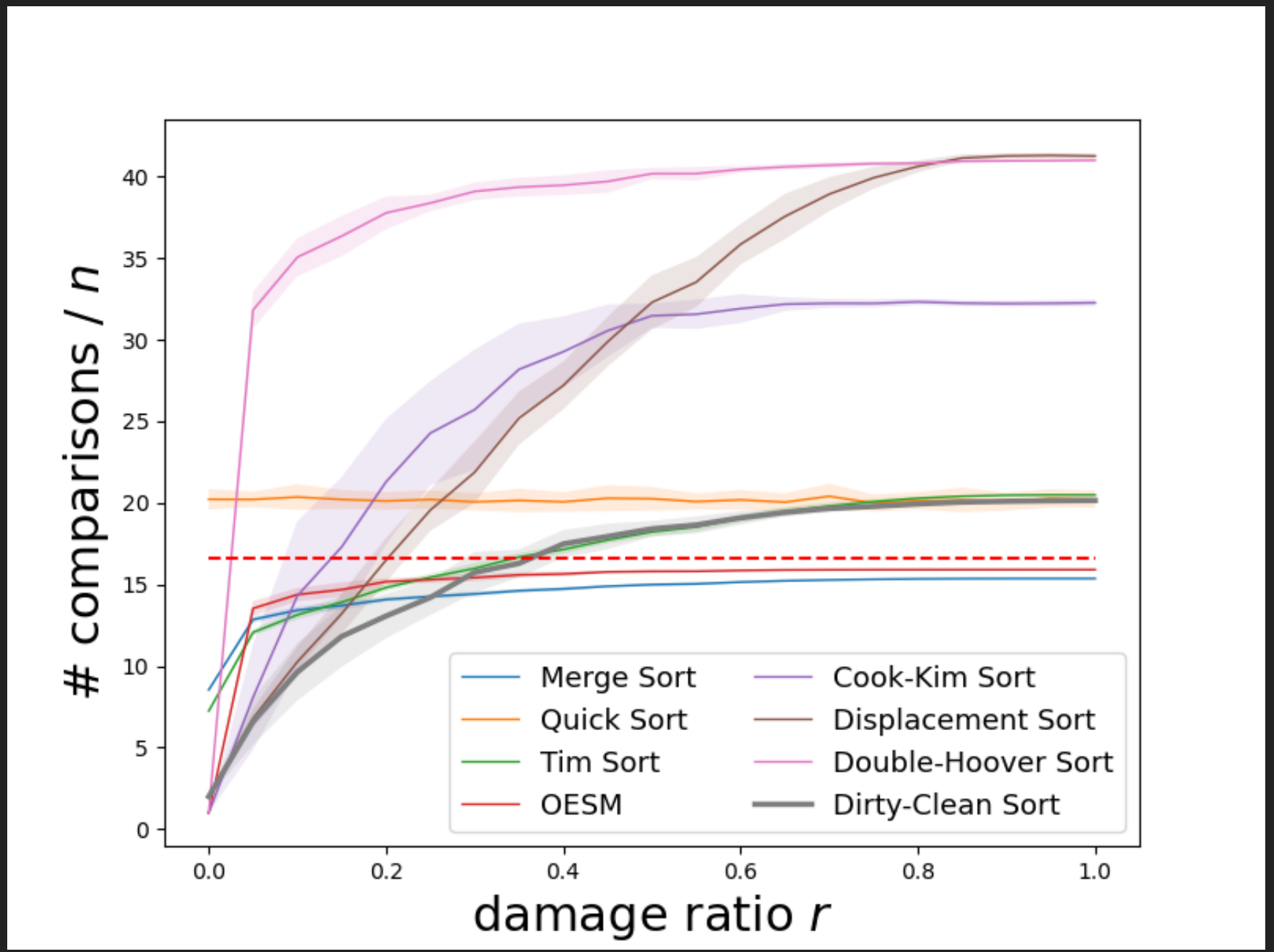
Repeatedly add ± 1 to $\hat{p}(i)$, for i random (n=1,000,000)



Experiments

Fraction r of items damaged (n=100,000)

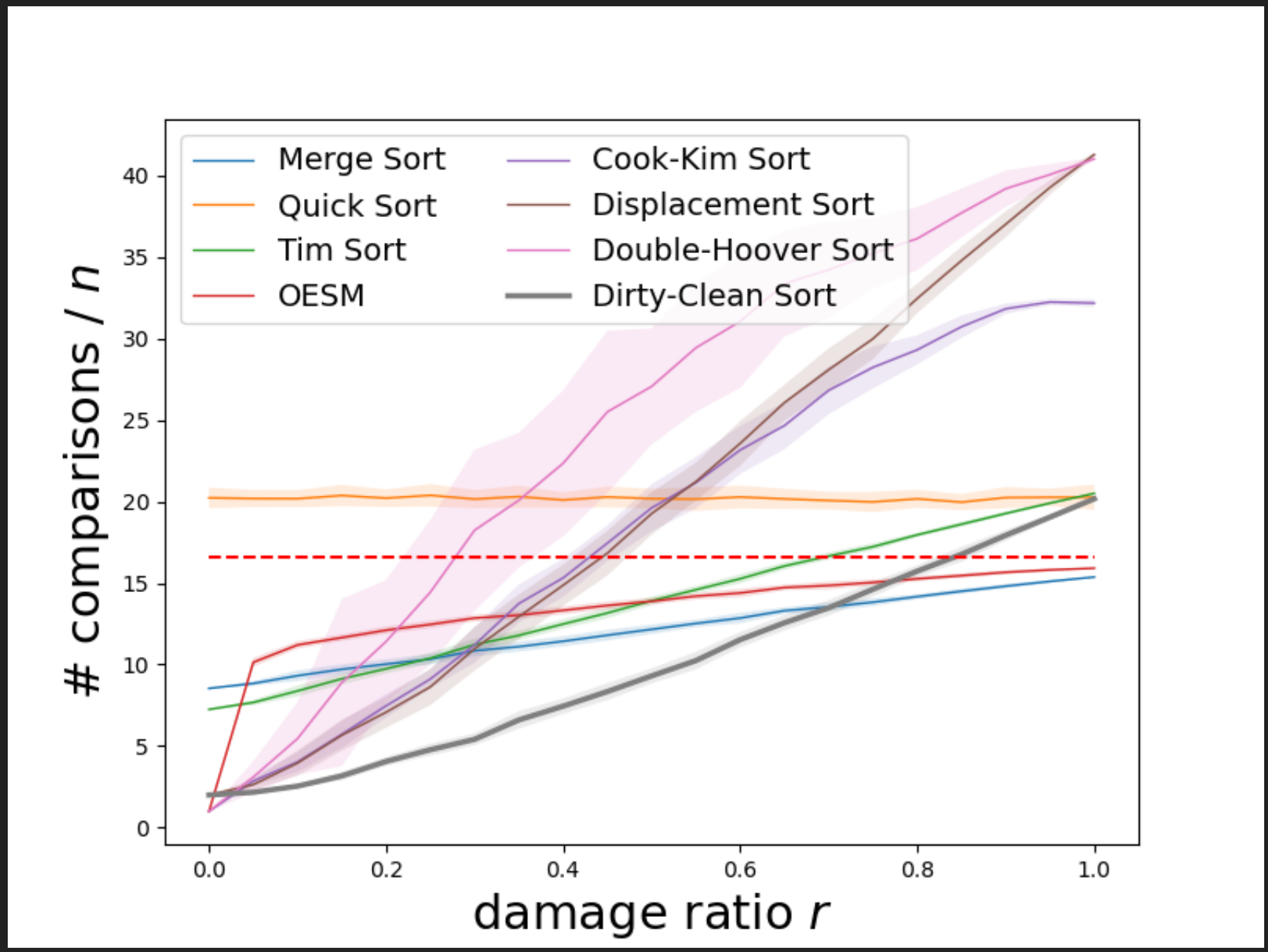
$\hat{<}$ random if an item damaged, otherwise correct



Experiments

Fraction r of items damaged (n=100,000)

$\hat{<}$ random if both items damaged, otherwise correct



Applications

- Testing drug efficiency
- Drawing conclusions from social experiments
- Ranking pages for their relevance to queries
- Any scenarios where quick-and-dirty comparisons are available!