LECTURE 6: PRIORITY QUEUES & BINARY HEAPS

Harold Soh
harold@comp.nus.edu.sg

ADMINISTRATIVE ISSUES

Poll Everywhere

https://bit.ly/2LvG9bq

Discussion/Tutorial group sheets are on Piazza.

Which part of the DG/Tutorial was most challenging?

A. Nothing. It was all good. ☺
B. The sorting invariants! Yuck!
C. The recurrence relations / asymptotic analysis.
D. Definitely Guess-the-Number.
E. OMG! All so difficult! Halp! ☹
F. What's a "discussion group"?

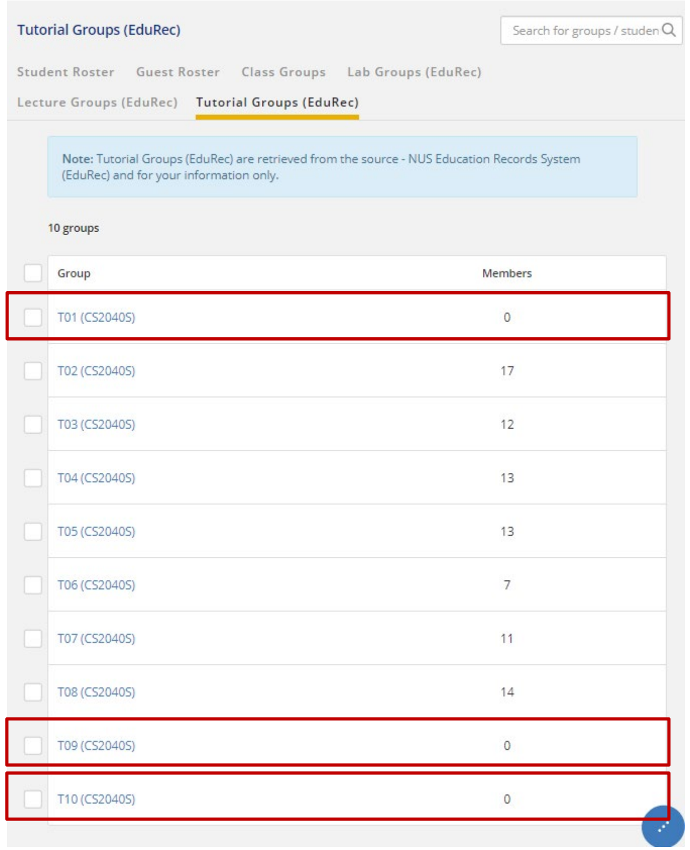
TUTORIALS AND LABS/DISCUSSION GROUPS

New allocations on Luminus.

Effective immediately.

Cancelled Groups:

- Labs: B01, B04
- Tutorials: T01, T09, T10



Tutorial Groups (EduRec)

Search for groups / studen

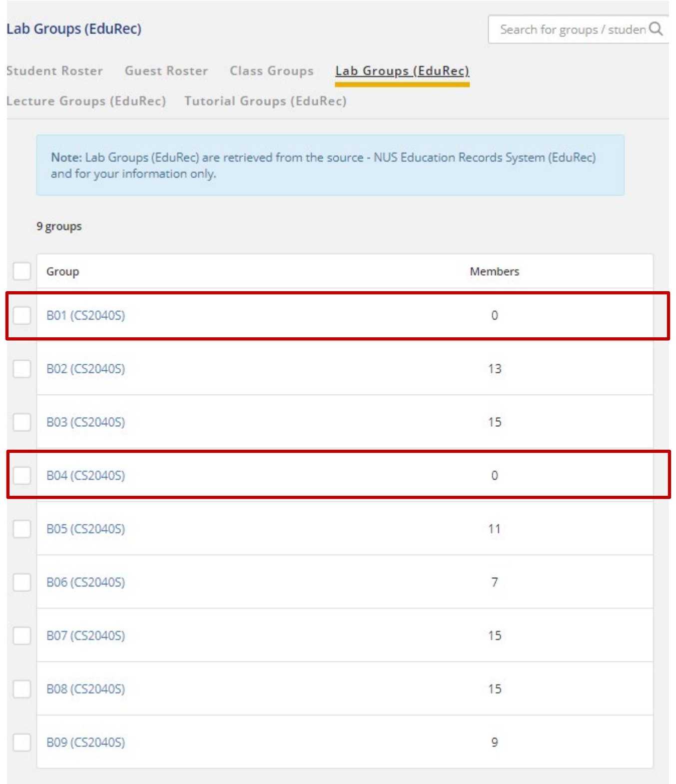
Student Roster Guest Roster Class Groups Lab Groups (EduRec)

Lecture Groups (EduRec) Tutorial Groups (EduRec)

Note: Tutorial Groups (EduRec) are retrieved from the source - NUS Education Records System (EduRec) and for your information only.

10 groups

Group	Members
T01 (CS2040S)	0
T02 (CS2040S)	17
T03 (CS2040S)	12
T04 (CS2040S)	13
T05 (CS2040S)	13
T06 (CS2040S)	7
T07 (CS2040S)	11
T08 (CS2040S)	14
T09 (CS2040S)	0
T10 (CS2040S)	0



Lab Groups (EduRec)

Search for groups / studen

Student Roster Guest Roster Class Groups Lab Groups (EduRec)

Lecture Groups (EduRec) Tutorial Groups (EduRec)

Note: Lab Groups (EduRec) are retrieved from the source - NUS Education Records System (EduRec) and for your information only.

9 groups

Group	Members
B01 (CS2040S)	0
B02 (CS2040S)	13
B03 (CS2040S)	15
B04 (CS2040S)	0
B05 (CS2040S)	11
B06 (CS2040S)	7
B07 (CS2040S)	15
B08 (CS2040S)	15
B09 (CS2040S)	9

ADMINISTRATIVE ISSUES

Quiz 1 is tomorrow

- Wednesday (4th Sept) during Lecture
- Both MCQ and Open-ended
- 3 Multipart questions.
- Open-book quiz
- No magnifying glass.
- No electronic equipment allowed.

QUIZ ADVICE

Don’t panic!

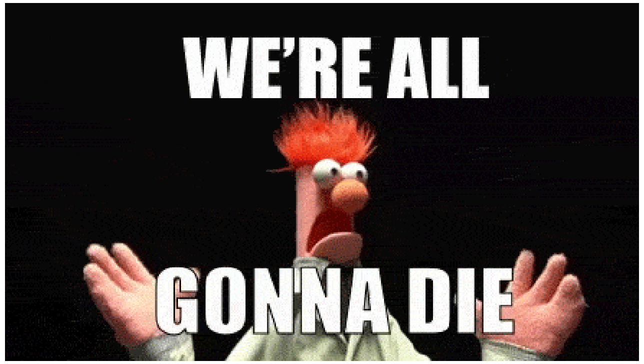
Go through Visualgo and the lecture notes.

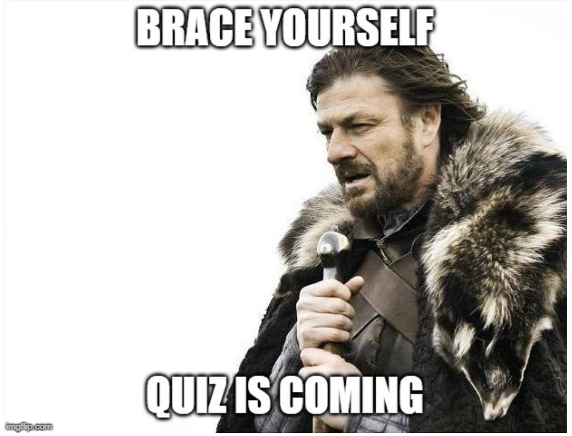
Sleep well the night before.

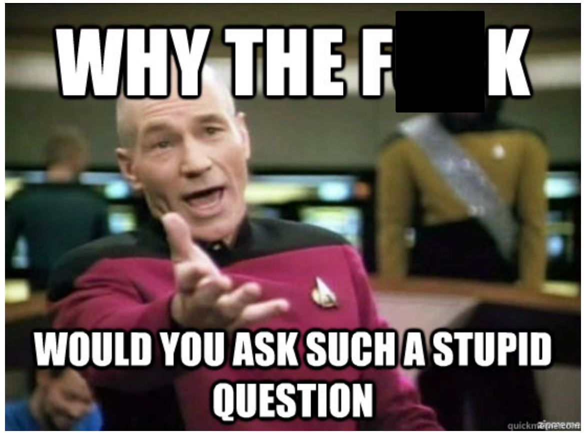
Sometimes, questions just *look* hard.

Don’t spend too long on any one question.

If you are unsure about something, raise your hand and ask.







1. Understand the Problem

- Understand what the problem is.
- Read through it carefully.

2. Analyze the Problem

- Analyze the problem:
  - What ***type*** of problem is it?
  - Have I seen something like it before?

3. Figure out a Solution.

- Is the solution apparent?
  - Draw upon what you have learnt.
- If not, use one of the strategies taught before, e.g.,
  - Derive the most straightforward **correct** solution (baseline).

4. Analyze the Solution

- What are the performance characteristics of the solution?

5. Describe the solution.

- Write the down neatly and clearly.
- Explain the method and also the performance characteristics.

6 Improve the solution

- What operations are expensive? Can I use a data structure to improve these operations?
- Can I trade memory for time?
- Is there structure (ordered, semi-ordered)?
- Divide and Conquer?
- Etc.

ADVICE:

Understanding + Analyzing the problem is usually 50-90% of the battle.

The solution may ***not*** be obvious at first.

It may take ***some hard thinking*** to “see” it.

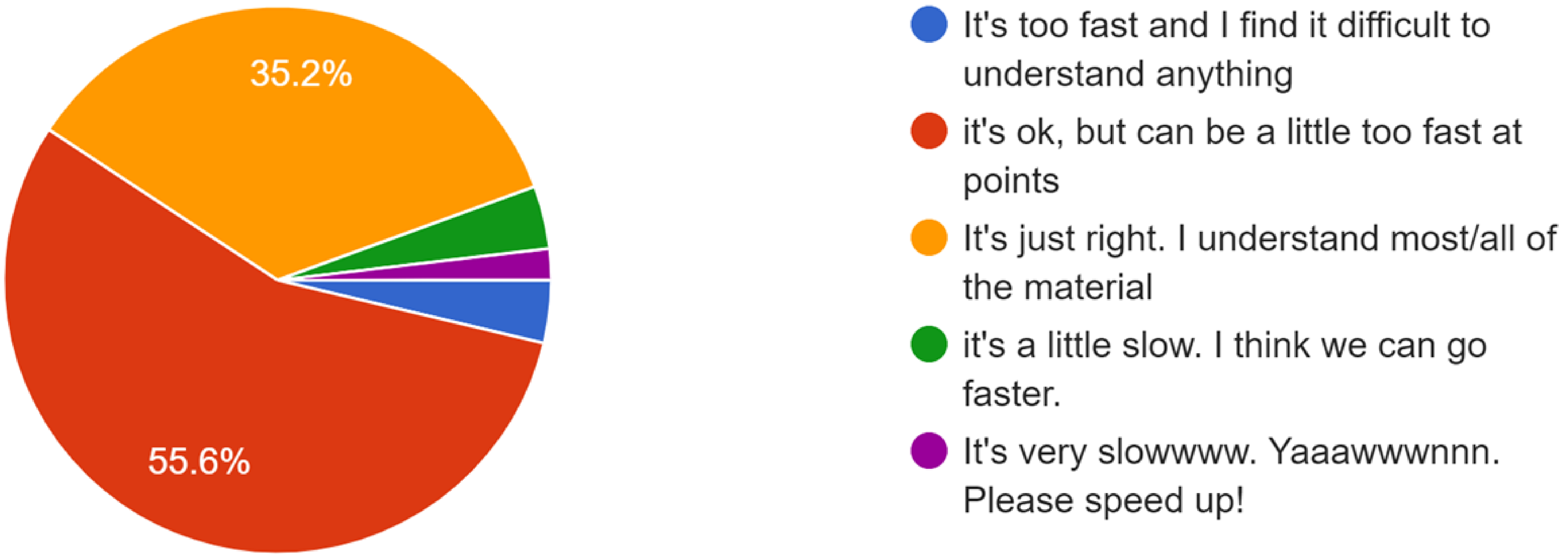
The ***best solution*** is ***often not obvious*** at first.

You have the tools to do all 6 steps.

LECTURE FEEDBACK

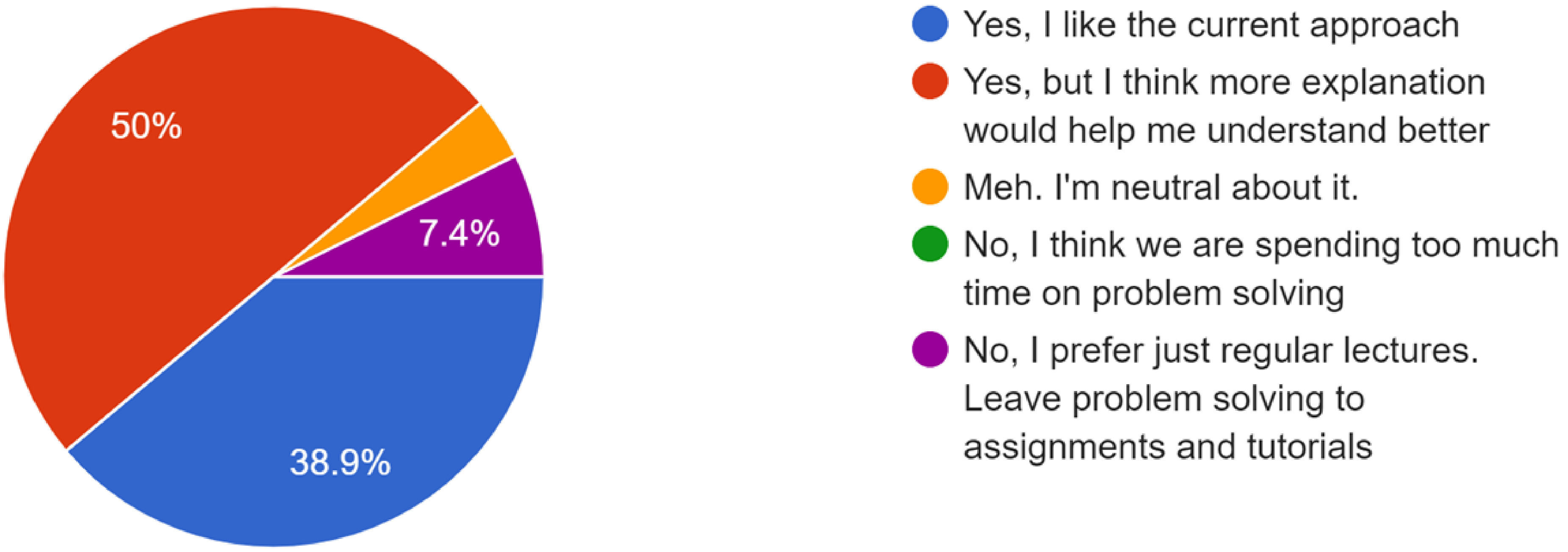
How do you find the speed of the lectures?

54 responses



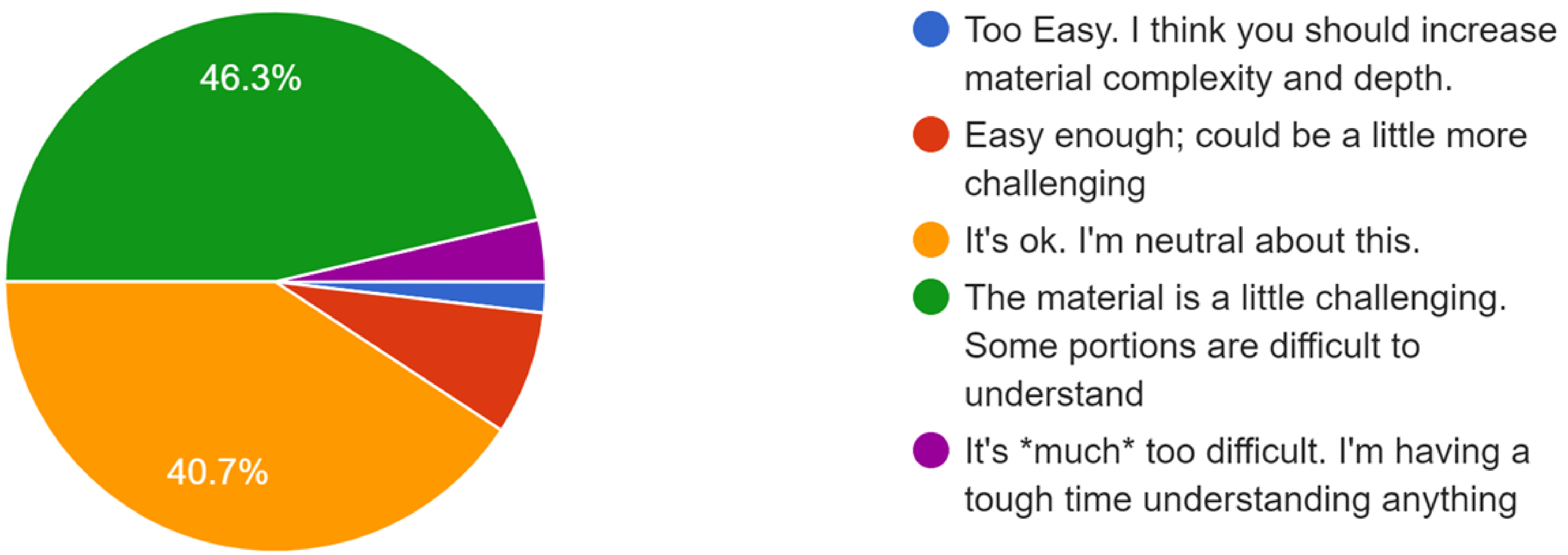
We are focussing on problem solving in class. Do you like the current style of teaching? (v.s. regular lectures)

54 responses



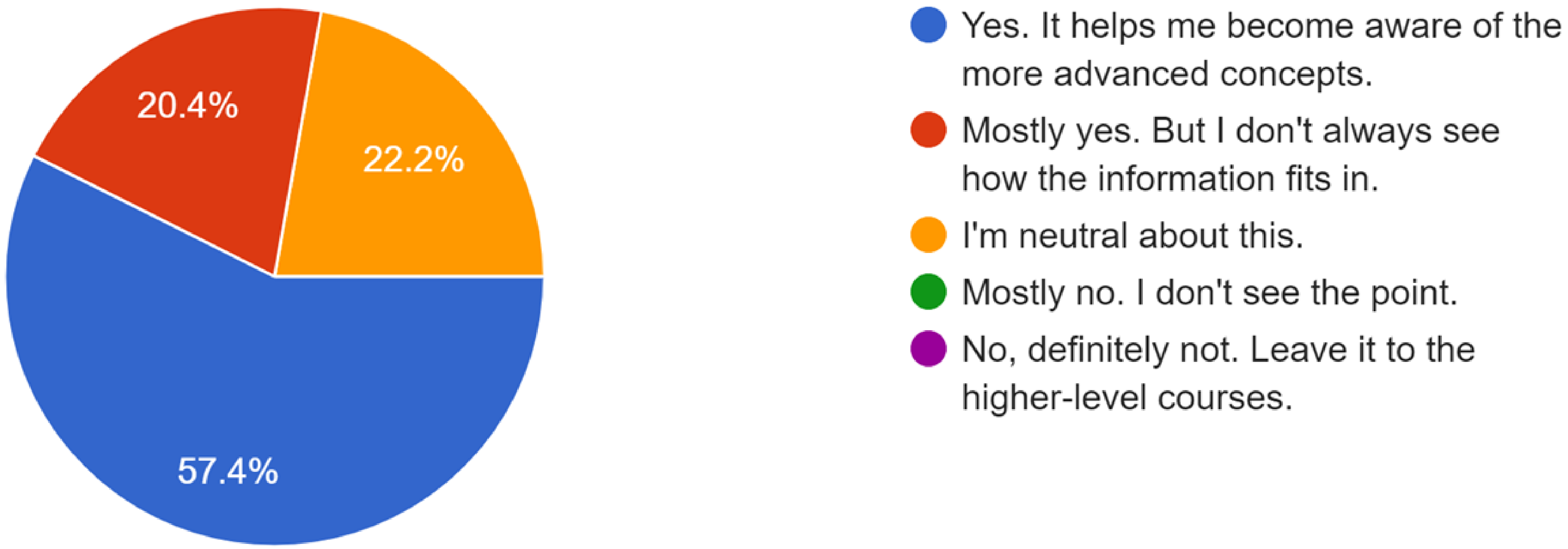
How difficult is the material?

54 responses



Do you think the "Extra" material is helpful?

54 responses



LECTURE FEEDBACK: MAIN COMMENTS

Will test webcasts out. :P

Webcasts:

"please enable WEBCASTS!...."

"Having webcast will be better, currently it is hard to revise lecture material with just the lecture slides."

"There can be webcast for lectures, to allow for better revision after the lectures."

"I would personally prefer webcast so that I can refer to it again if I have doubts. Maybe delay webcast broadcast by 2 to 3 sessions so as to prevent students from skipping lectures?"

Many many more…

EXTRA MATERIALS

There's already a section. We'll post more stuff!

"…the "extra" materials are really interesting but a little bit difficult to understand and just a tiny bit more explanation would go a really long way… the depth covered in class is a little lacking and rushed… But perhaps that is your goal, to pique our interest, but to have us not completely understand so we can look it up ourselves? If that is the case, you've succeeded but at the same time left me feeling a little cheated :("

"Maybe there can also be an extra extra section for more readings/links to resources that are interesting/go more in depth"

DIFFICULTY

- **Labs will often revise the weeks (or previous week's material)**
- **Problem sets have easy and challenge problems**
  - **See Tips and Hints on piazza.**
- **Tutorials are meant to train on-the-spot thinking.**

"the lab is really really really very hard.... the stuff taught in the lecture and the lab feel very different. The algo that the lab question is trying to test is not very obvious..."

"I find the lectures okay, but the assignment I think is way too difficult."

"Hopefully, tutorials can be released before the actual tutorial because the pace of tutorial is too fast for me and I needed more time to think about approaching the qns etc."

POLLING TIME

Will try to adjust polling time.

“I don’t really like the poll in lectures as i feel it wastes quite a bit of time”

“i think the breaks for answering are a bit long sometimes.”

“I think we could possibly save some time if we cut down the "waiting time" for closing the polls.”

If you have any other feedback, please share it with me or the Tas.

QUESTIONS?

LEARNING OUTCOMES

By the end of this session, students should be able to:

- Describe the **priority queue ADT** and its operations
- Describe the **binary heap data structure** and explain how it works
- Analyze the **performance of the binary heap data** structure
- Describe the **heapsort algorithm** and explain how it works
- Analyze the **performance of heapsort**

PROBLEM: PRIORITIZING PATIENTS AT THE HOSPITAL

Each patient assigned a priority score.

Doctors need to see the highest priority patients first!

DESIGN AN ABSTRACT DATA TYPE FOR THIS PROBLEM

What operations would we need to support?

THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

Poll Everywhere

https://bit.ly/2LvG9bq

WHAT DATA STRUCTURE?

Operations:

- `insert(x)`
- `max()`
- `extractMax()`
- `size()`
- `buildHeap(A)`

What is the worst-case time complexity of `insert` followed by `extractMax` if we use a singly linked list?

A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(n^2)$
E. I predict there will be complexity questions in the quiz.

WHAT DATA STRUCTURE?

Poll Everywhere
https://bit.ly/2LvG9bq

Operations:

- `insert(x)`
- `max()`
- `extractMax()`
- `size()`
- `buildHeap(A)`

Have to scan through the array to find the max (or the next maximum)
or inserts have to maintain sorted order

What is the worst-case time complexity of `insert` followed by `extractMax` if we use a singly linked list?

A. $O(1)$
B. $\boldsymbol{O(n)}$
C. $O(\log n)$
D. $O(n^2)$
E. I predict there will be complexity questions in the quiz.

Poll Everywhere

https://bit.ly/2LvG9bq

WHAT DATA STRUCTURE?

Operations:

- `insert(x)`
- `max()`
- `extractMax()`
- `size()`
- `buildHeap(A)`

What is the worst-case time complexity of `insert` followed by `extractMax` if ***we use an array***?

A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(n^2)$
E. I'm sure there will be complexity questions in the quiz.

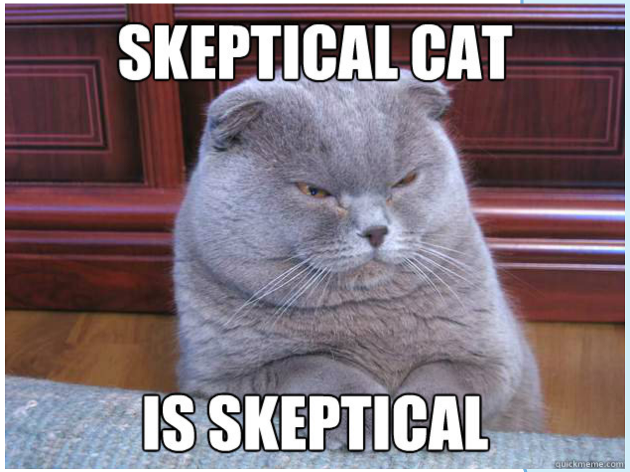
Poll Everywhere

https://bit.ly/2LvG9bq

WHAT DATA STRUCTURE?

Operations:

- `insert(x)`
- `max()`
- `extractMax()`
- `size()`
- `buildHeap(A)`



What is the worst-case time complexity of `insert` followed by `extractMax` if ***<u>we use an array</u>***?

A. $O(1)$

B. $O(n)$

C. $\boldsymbol{O(\log n)}$

D. $O(n^2)$

E. I'm sure there will be complexity questions in the quiz.

EXPLOITING STRUCTURE

Unsorted array

30	70	12	50	6

Insert: O(1)
Max: O(n)
Search: O(n)

Sorted array

6	12	30	50	70

Insert: O(n)
Max: O(1)
Search: O(logn)

Unordered **Ordered**

EXPLOITING STRUCTURE

The more “ordered” your structure:

- the more prior information you can exploit to speed up certain operations.

But: you will likely have to pay to maintain this order.

- some operations (e.g., that change the data) will become more expensive

EXPLOITING STRUCTURE

Unsorted array

30	70	12	50	6

Insert: $O(1)$
Max: $O(n)$
Search: $O(n)$

Sorted array

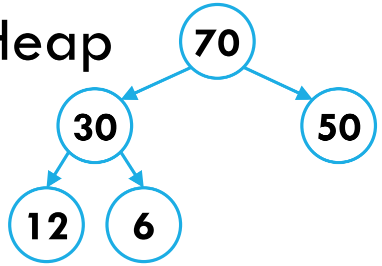
6	12	30	50	70

Insert: $O(n)$
Max: $O(1)$
Search: $O(\log n)$

Unordered **Ordered**

EXPLOITING STRUCTURE

Binary Heap



Unsorted array

30	70	12	50	6

Insert: O(1)
Max: O(n)
Search: O(n)

70	30	50	12	6

Insert: O(logn)
Max: O(1)
Search: O(n)

Sorted array

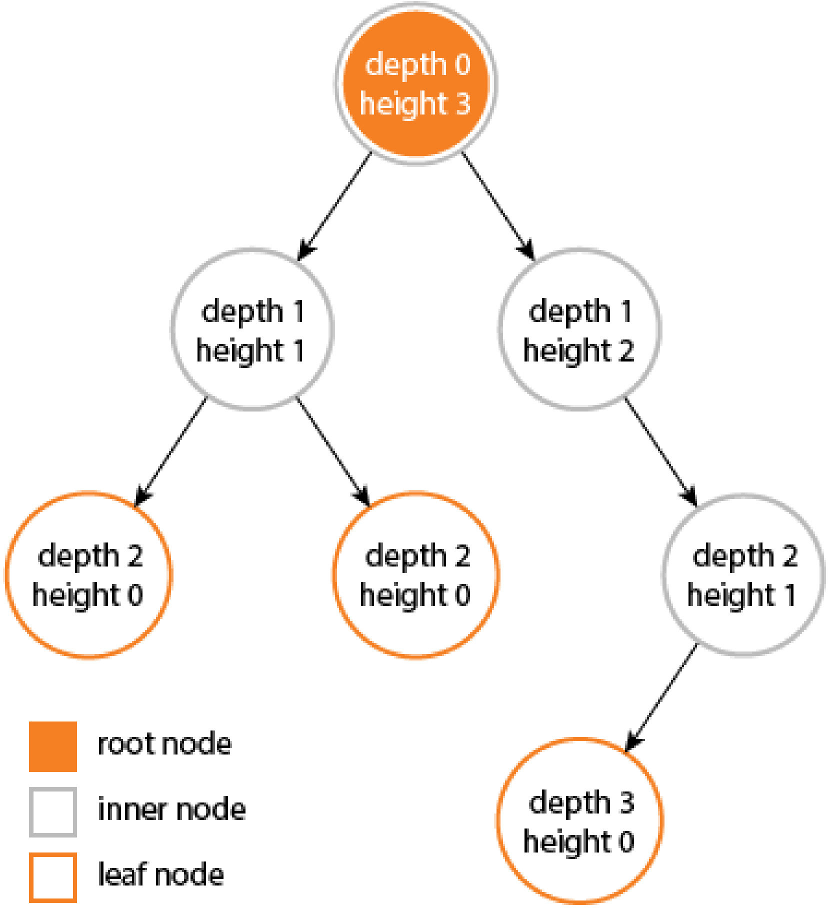
6	12	30	50	70

Insert: O(n)
Max: O(1)
Search: O(logn)

Unordered **Ordered**

BINARY TREE

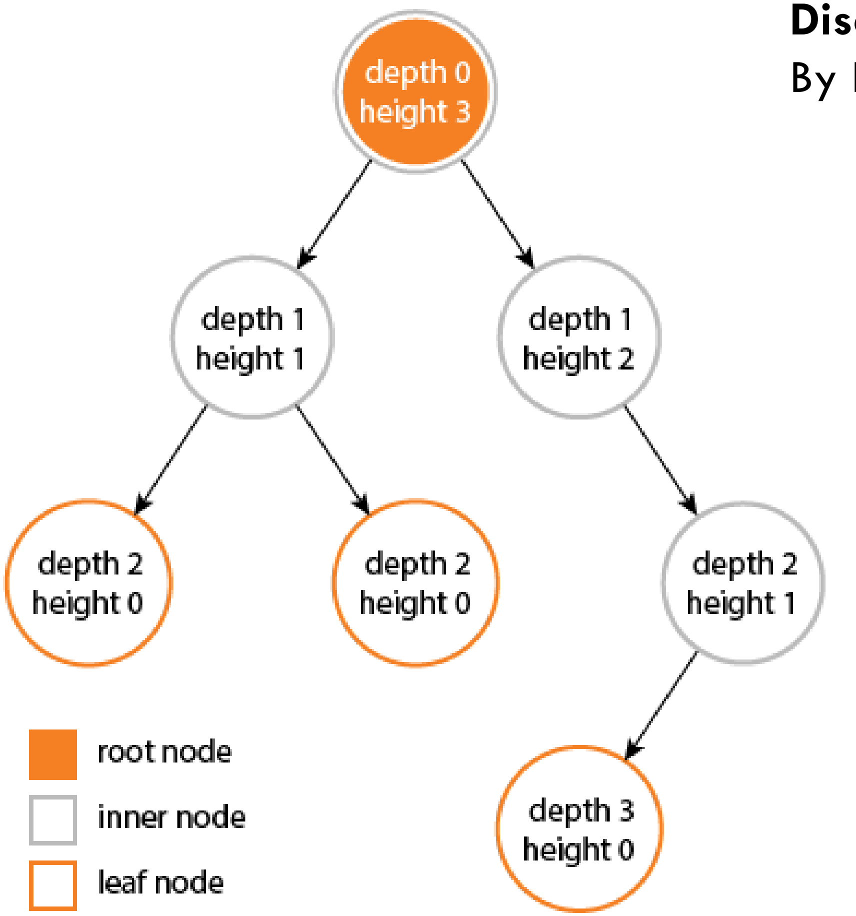
A BINARY TREE



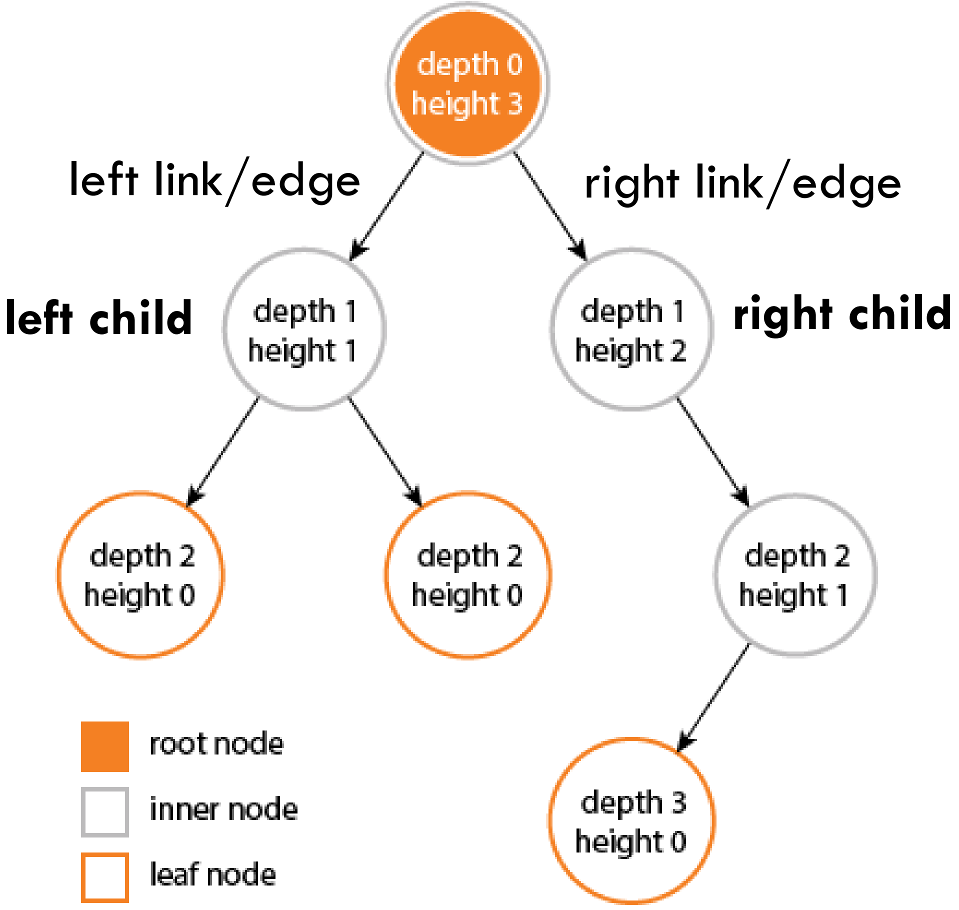
A BINARY TREE

> **Remark.** A picture like this is called a *tree*. (This is not a formal definition; that will follow later.) If you want to know why the tree is growing upside down, ask the computer scientists who introduced this convention. (The conventional wisdom is that they never went out of the room, and so they never saw a real tree.)

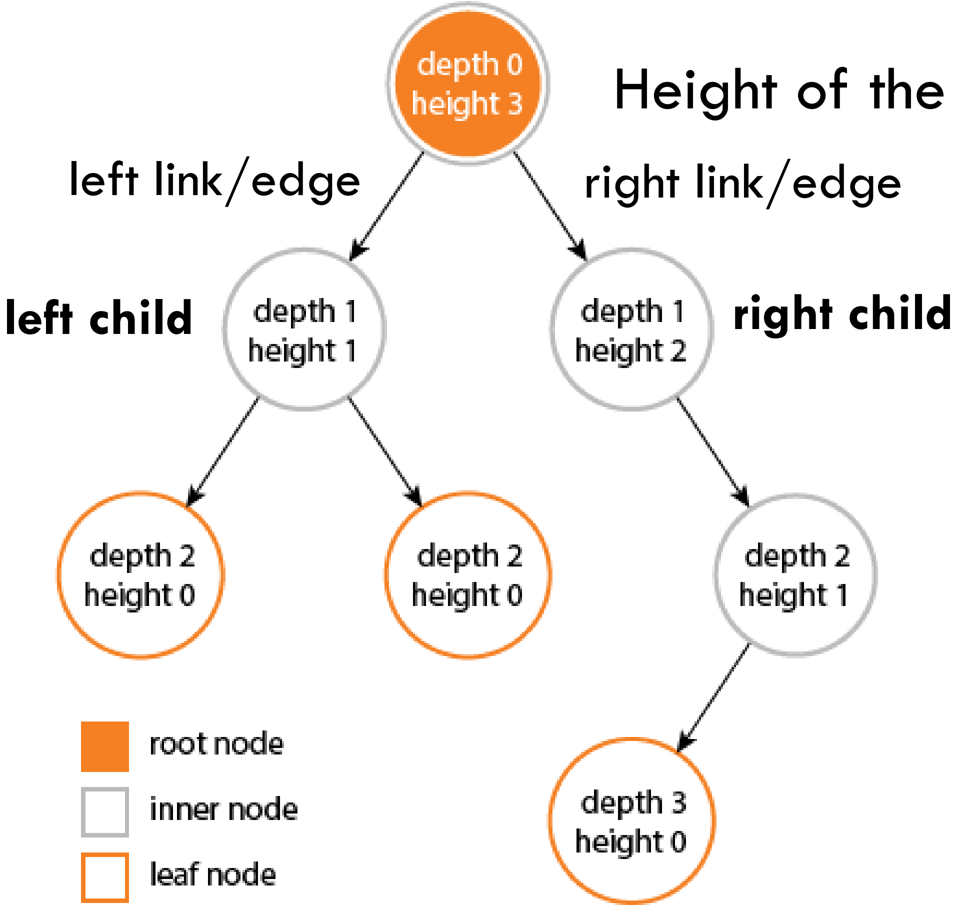
Discrete Mathematics: Elementary and Beyond
By László Lovász, József Pelikán, Katalin Vesztergombi



A BINARY TREE



A BINARY TREE

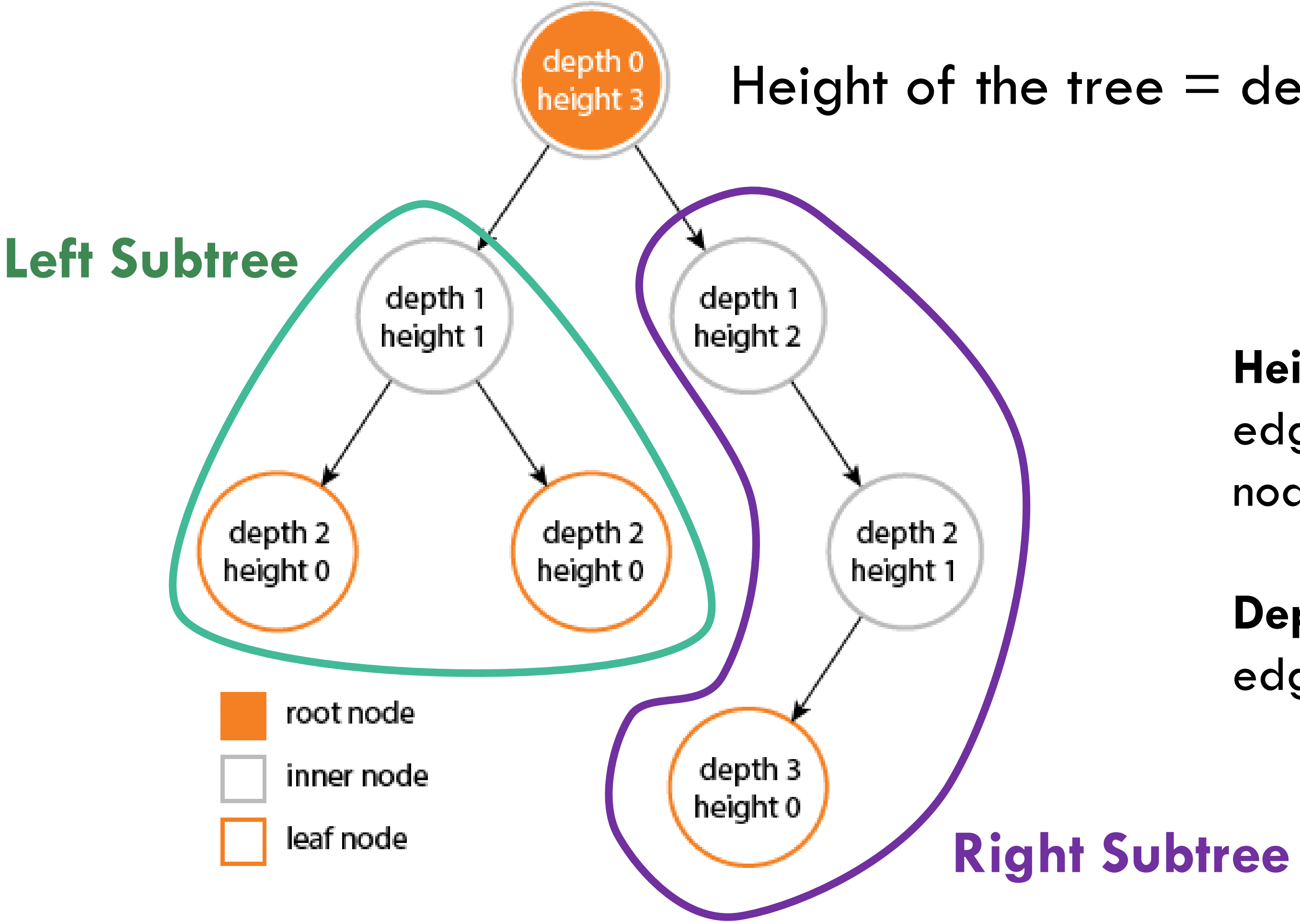


Height of the tree = depth of its deepest node

Height of a node: number of edges on the longest path from node to a leaf (via its children)

Depth of a node: number of edges to the root

A BINARY TREE

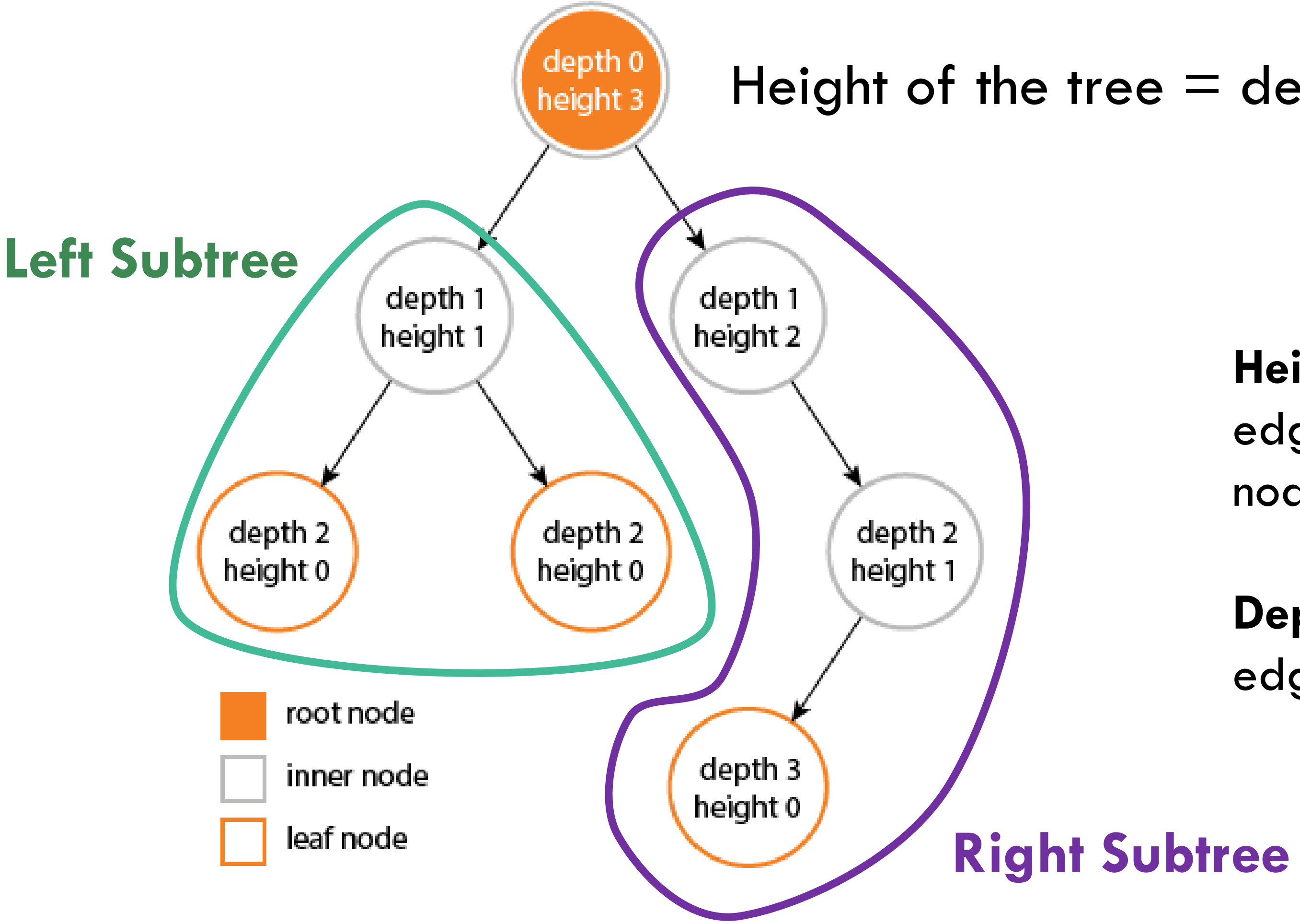


Height of the tree = depth of its deepest node

Height of a node: number of edges on the longest path from node to a leaf (via its children)

Depth of a node: number of edges to the root

A BINARY TREE

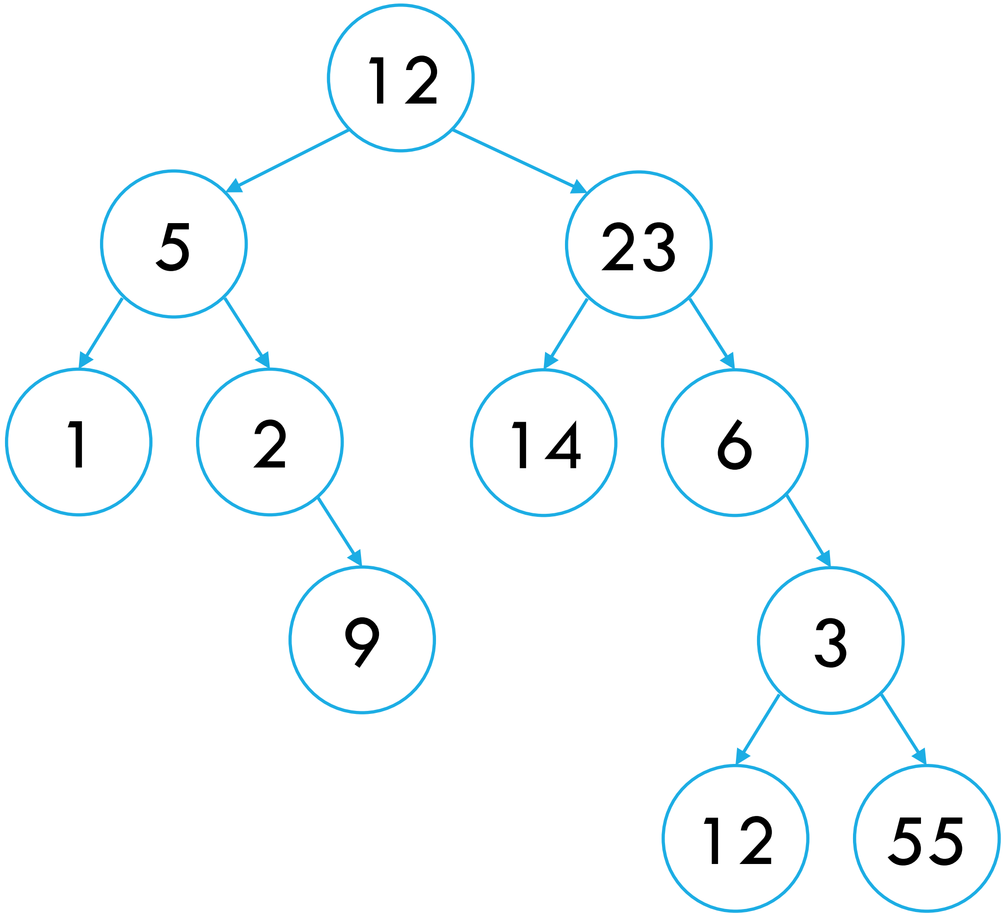


Height of the tree = depth of its deepest node

Height of a node: number of edges on the longest path from node to a leaf (via its children)

Depth of a node: number of edges to the root

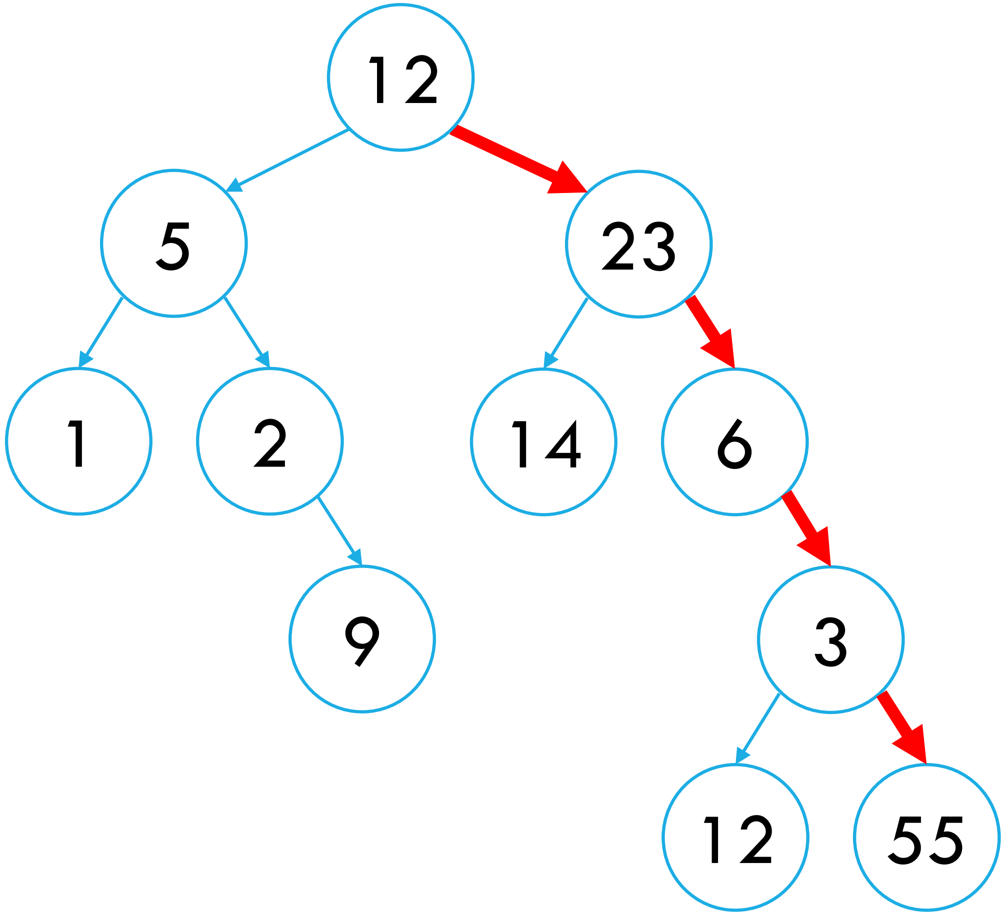
A BINARY TREE



What is the height of the tree?

A. 1
B. 3
C. 4
D. 5
E. Height.. depth.. why is the tree upside down?!

A BINARY TREE



What is the height of the tree?

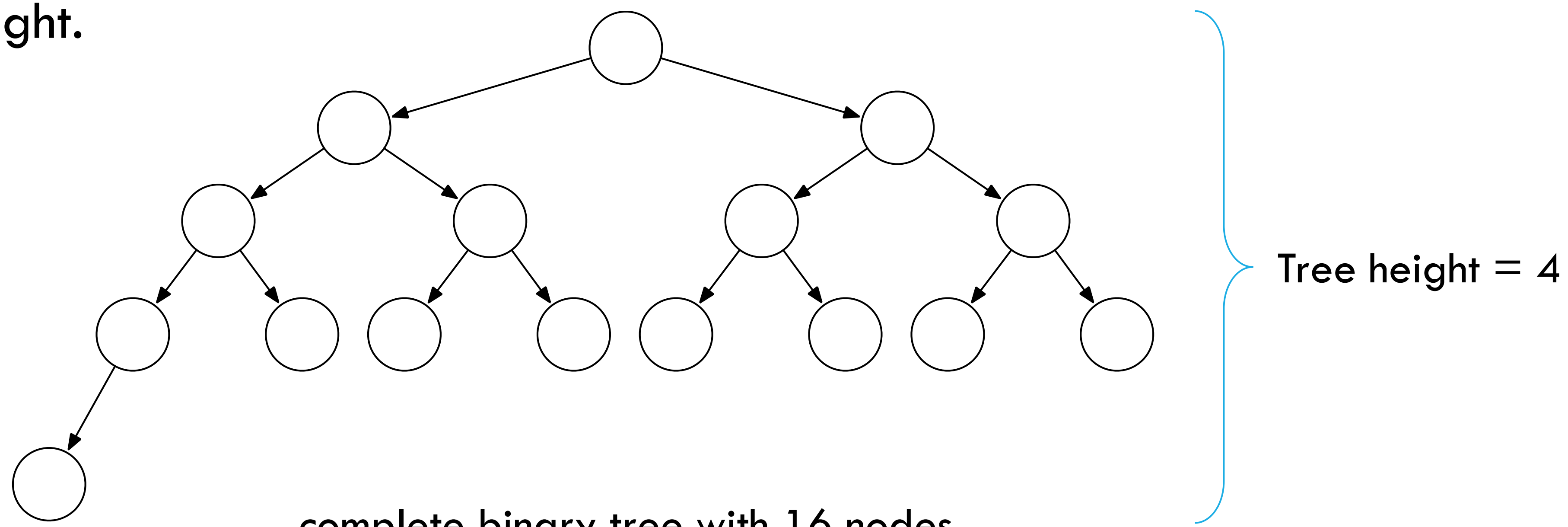
A. 1
B. 3
C. **4**
D. 5
E. Height.. depth.. why is the tree upside down?!

COMPLETE BINARY TREE

Binary Tree: nodes with links to left and right binary trees (or empty)

Complete tree: Perfectly balanced, except for bottom level

- all levels completely filled except *possibly* last level where the keys are filled from left to right.

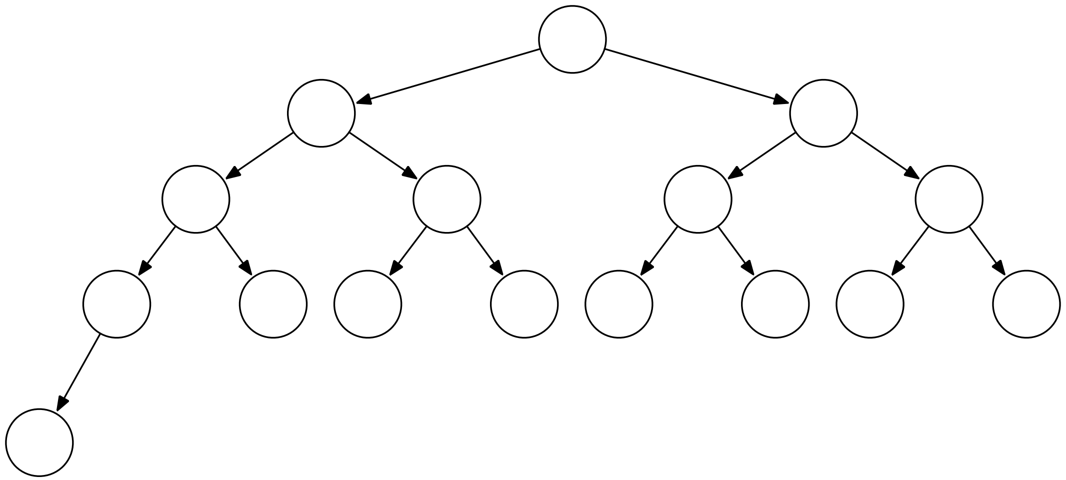


complete binary tree with 16 nodes

COMPLETE BINARY TREE

Poll Everywhere

https://bit.ly/2LvG9bq



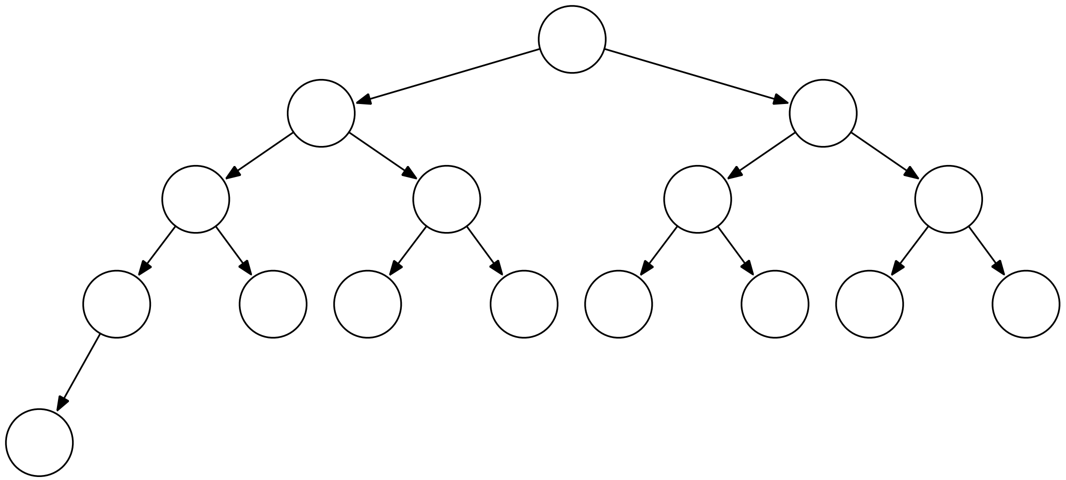
What is the height h of a complete binary tree with n nodes?

A. $h = n$
B. $h = 2^n$
C. $h = \log n$
D. $h = \lfloor \log n \rfloor$
E. None of the above

Poll Everywhere

https://bit.ly/2LvG9bq

COMPLETE BINARY TREE



What is the height *h* of a complete binary tree with n nodes?

A. $h = n$

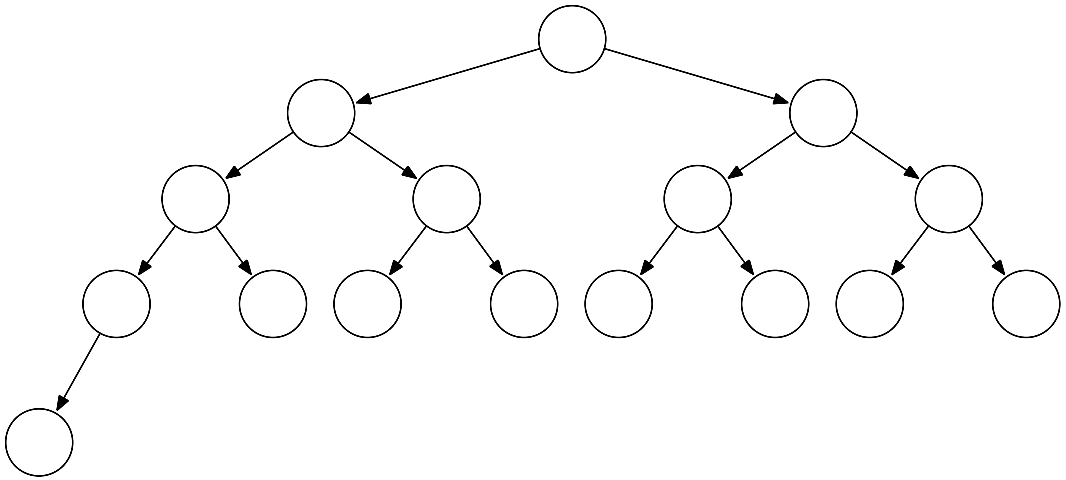
B. $h = 2^n$

C. $h = \log n$

D. $\boldsymbol{h = \lfloor \log n \rfloor}$

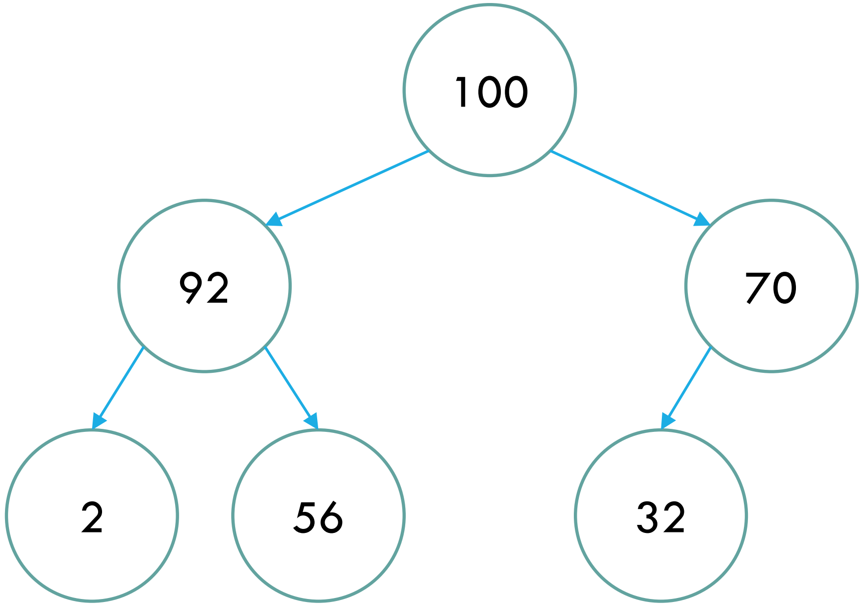
E. None of the above

HEIGHT OF A COMPLETE BINARY TREE



# Nodes	$\log n$	$h = \lfloor \log n \rfloor$
1	0.00	0
2	1.00	1
3	1.58	1
4	2.00	2
7	2.81	2
8	3.00	3
15	3.91	3
16	4.00	4

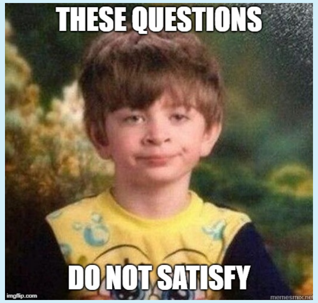
BINARY HEAP



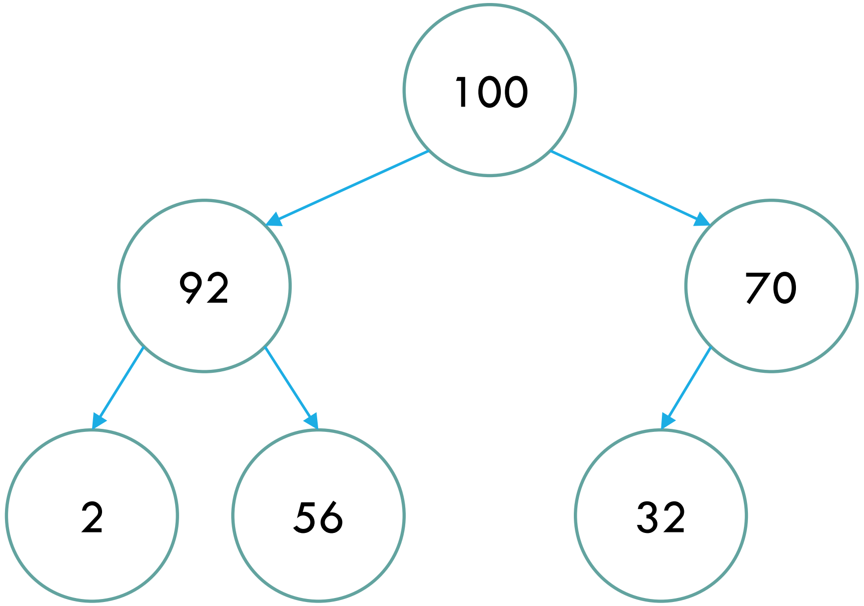
Max heap property:

- the key of each node is larger than or equal to the keys in its children

Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. No, there is a violation.
C. Maybe yes.. Maybe no...
D. 

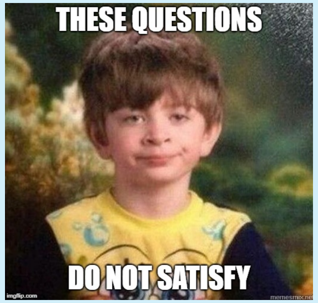
BINARY HEAP



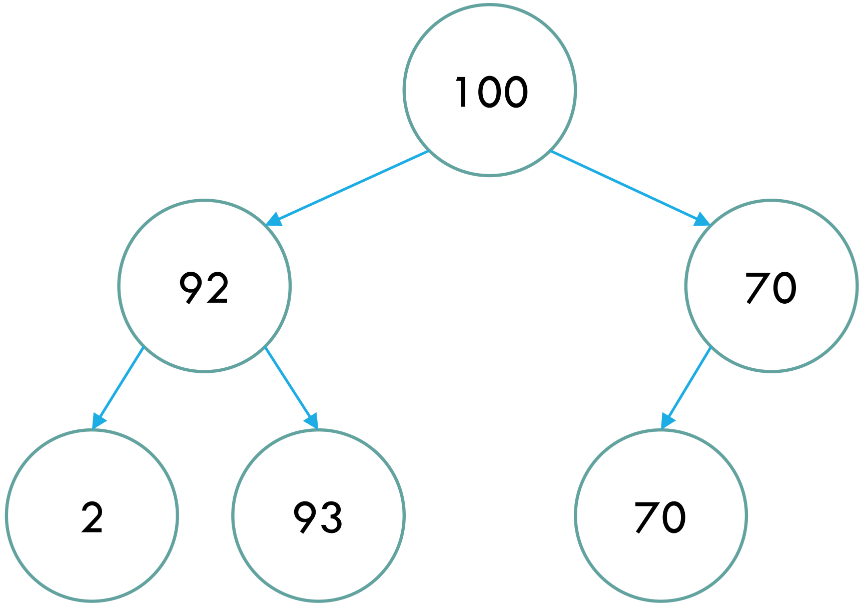
Max heap property:

- the key of each node is larger than or equal to the keys in its children

Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. No, there is a violation.
C. Maybe yes.. Maybe no...
D. 

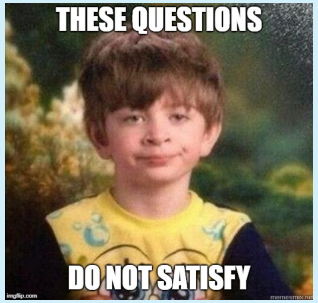
BINARY HEAP



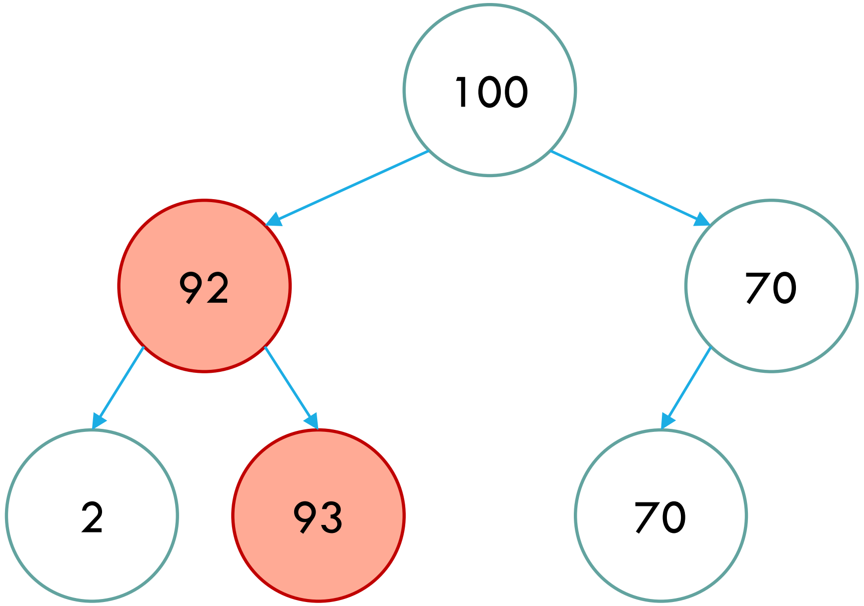
Max heap property:

- the key of each node is larger than or equal to the keys in its children

Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. No, there is a violation.
C. Maybe yes.. Maybe no...
D. 

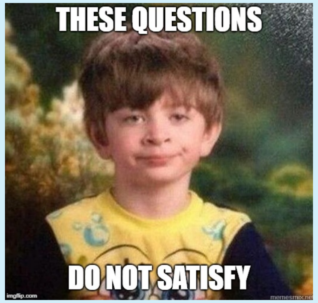
BINARY HEAP



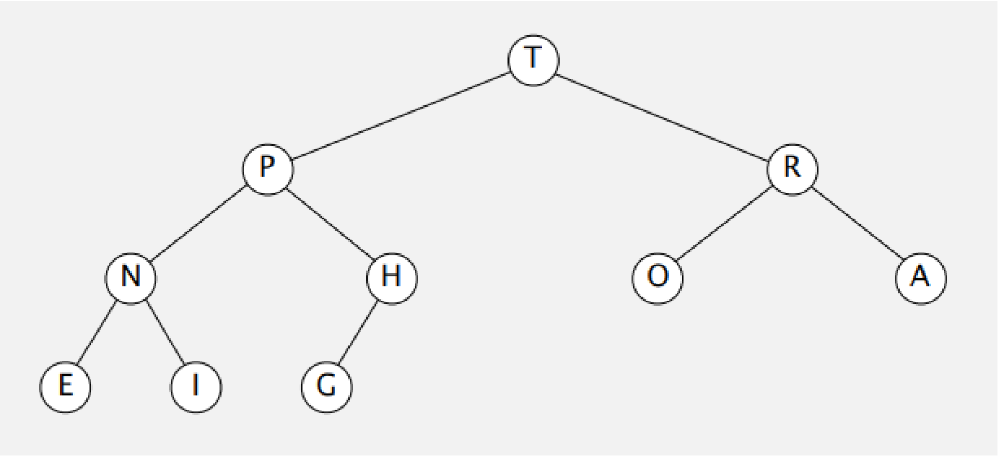
Max heap property:

- the key of each node is larger than or equal to the keys in its children

Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. **No, there is a violation.**
C. Maybe yes.. Maybe no...
D. 

BINARY HEAP



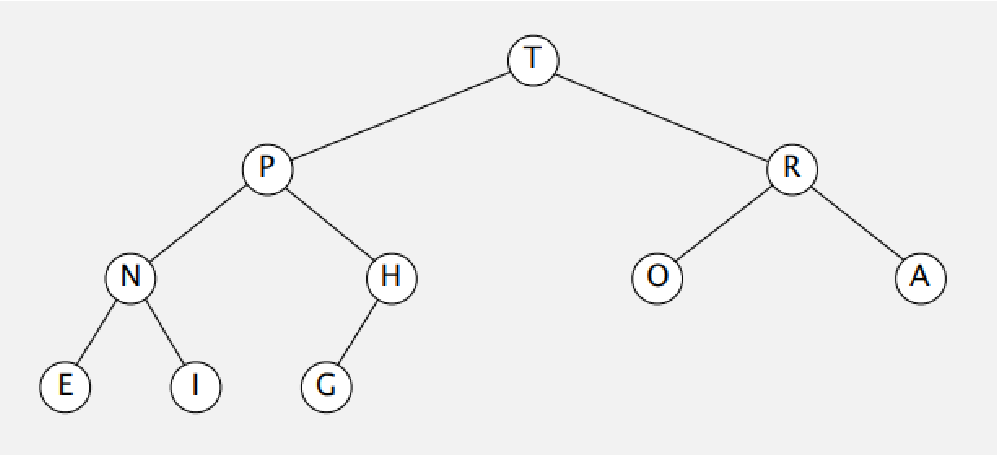
Max heap property:

- the key of each node is larger than or equal to the keys in its children

Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. No, there is a violation.
C. Not sure!
D. Where did the numbers go?
E. Depends on how we define our comparator.

BINARY HEAP



Max heap property:

- the key of each node is larger than or equal to the keys in its children

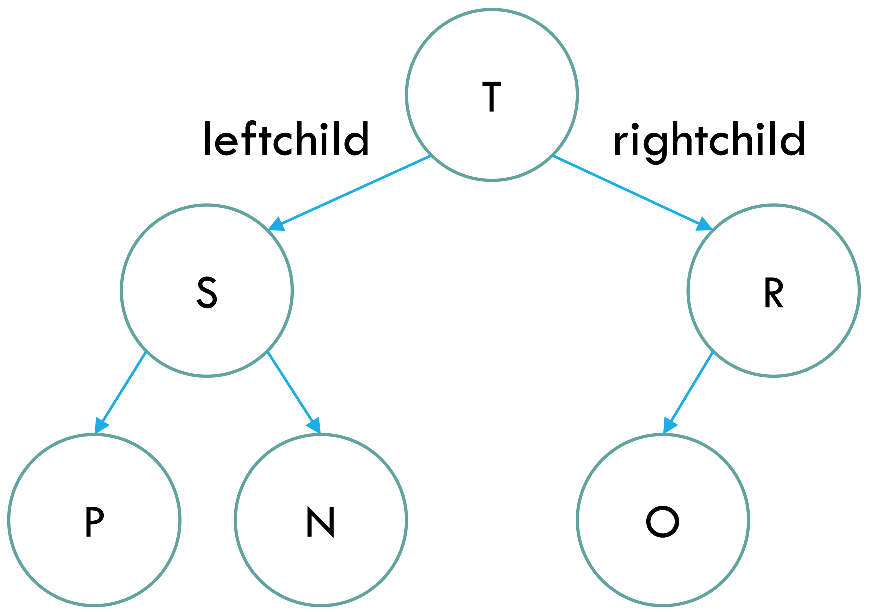
Does the binary tree on the left satisfy the max heap property?

A. Yes.
B. No, there is a violation.
C. Not sure!
D. Where did the numbers go?
E. Depends on how we define our comparator.

TREE REPRESENTATION

```
Class Node
    Object key
    Object data

    Node leftchild
    Node rightchild
    Node parent
```



ARRAY REPRESENTATION

Indices start at 1

Take nodes in level order

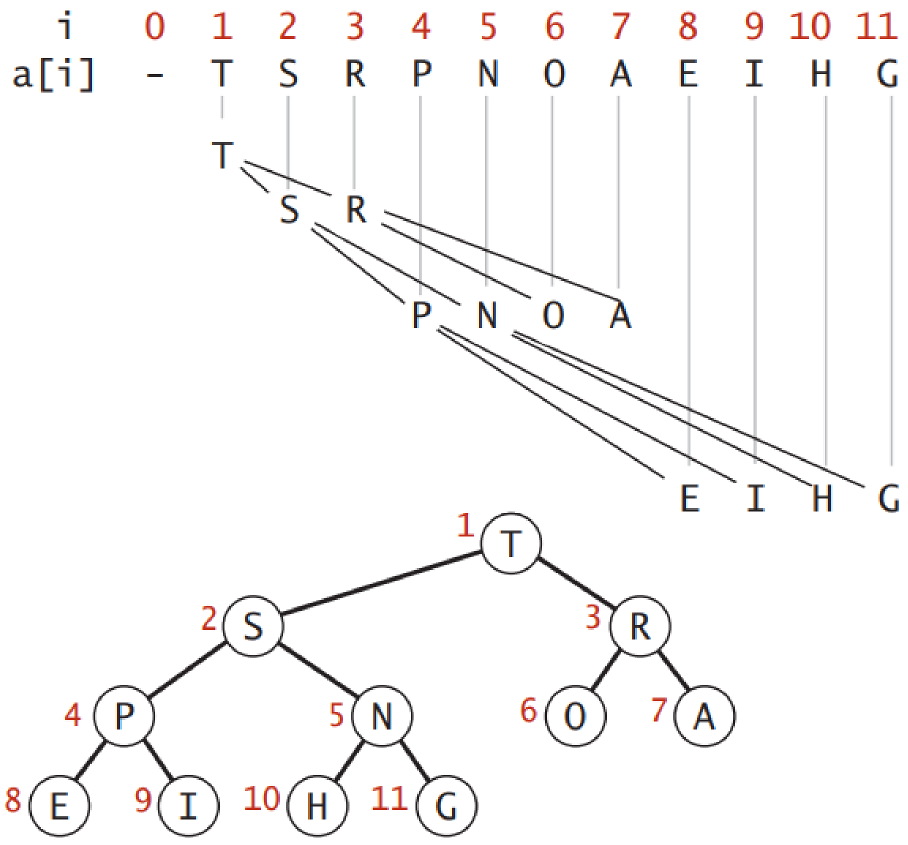
No links needed

Parent of node k is at ______

Children of node k are at:

Left Child: ______

Right Child: ______



Heap representations

ARRAY REPRESENTATION

Indices start at 1

Take nodes in level order

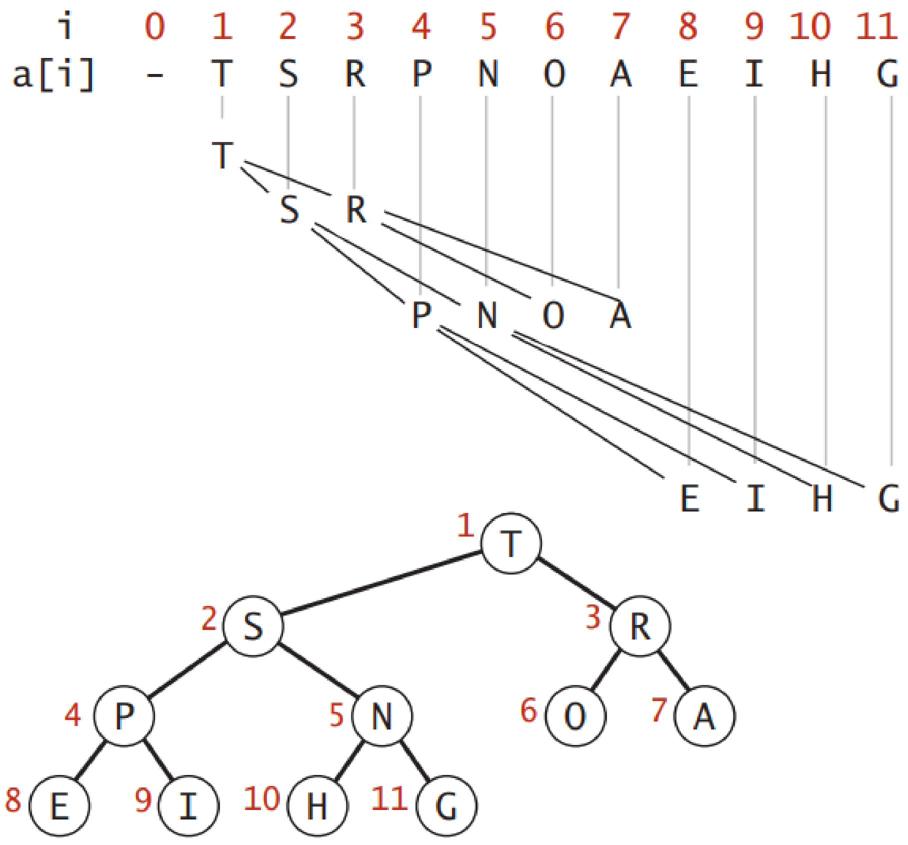
No links needed

Parent of node k is at $\boldsymbol{k/2}$

Children of node k are at:

Left Child: _______

Right Child: _______



Heap representations

ARRAY REPRESENTATION

Indices start at 1

Take nodes in level order

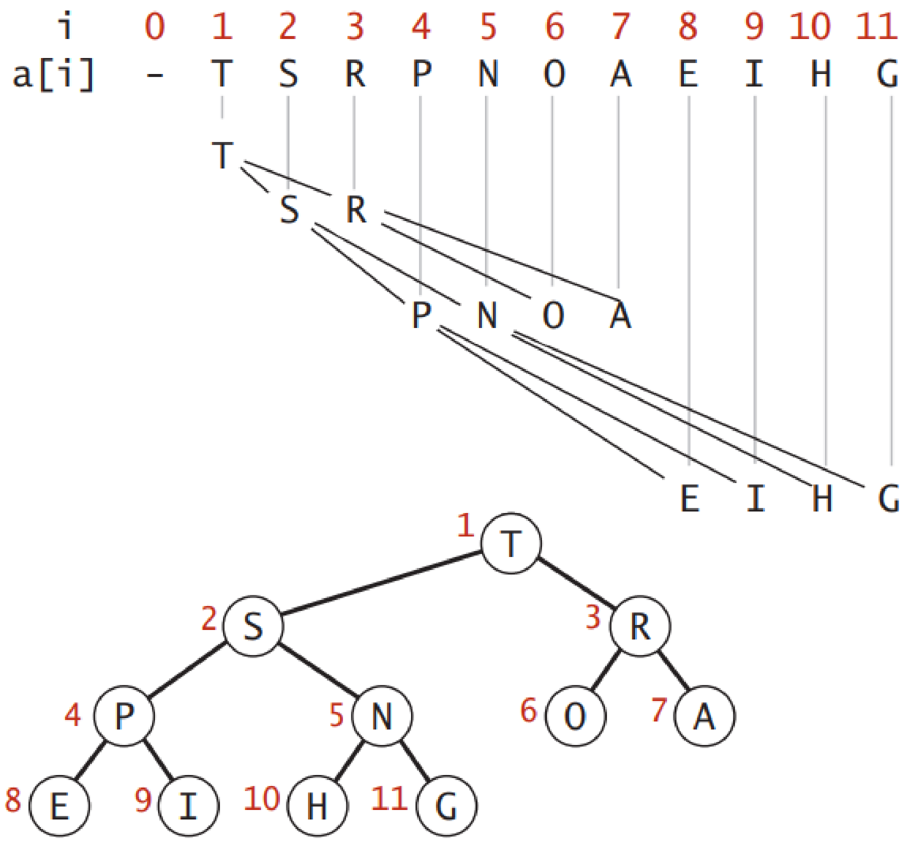
No links needed

Parent of node k is at $\boldsymbol{k/2}$

Children of node k are at:

Left Child: $\boldsymbol{2k}$

Right Child: $\boldsymbol{2k+1}$

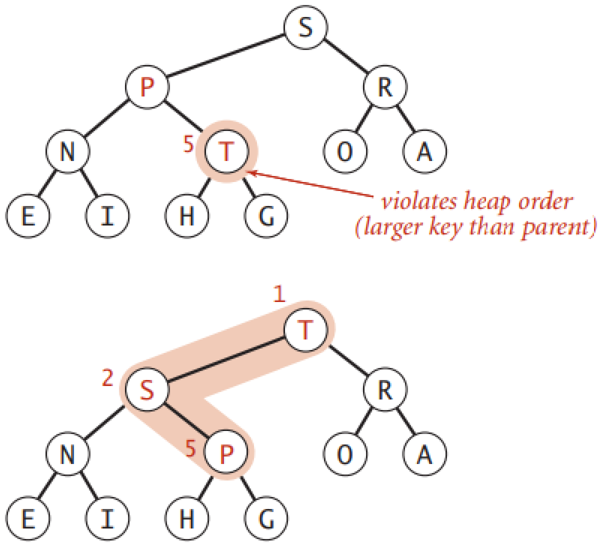


Heap representations

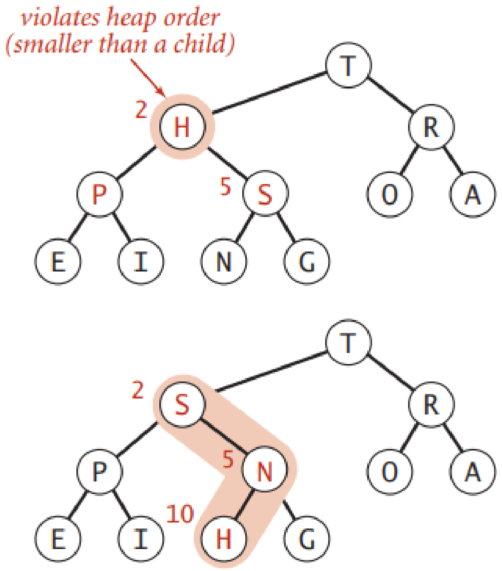
TWO IMPORTANT OPERATIONS: SWIM & SINK

We use these operations to correct binary trees to satisfy the max heap property

TWO IMPORTANT OPERATIONS: SWIM & SINK



Swim



Sink

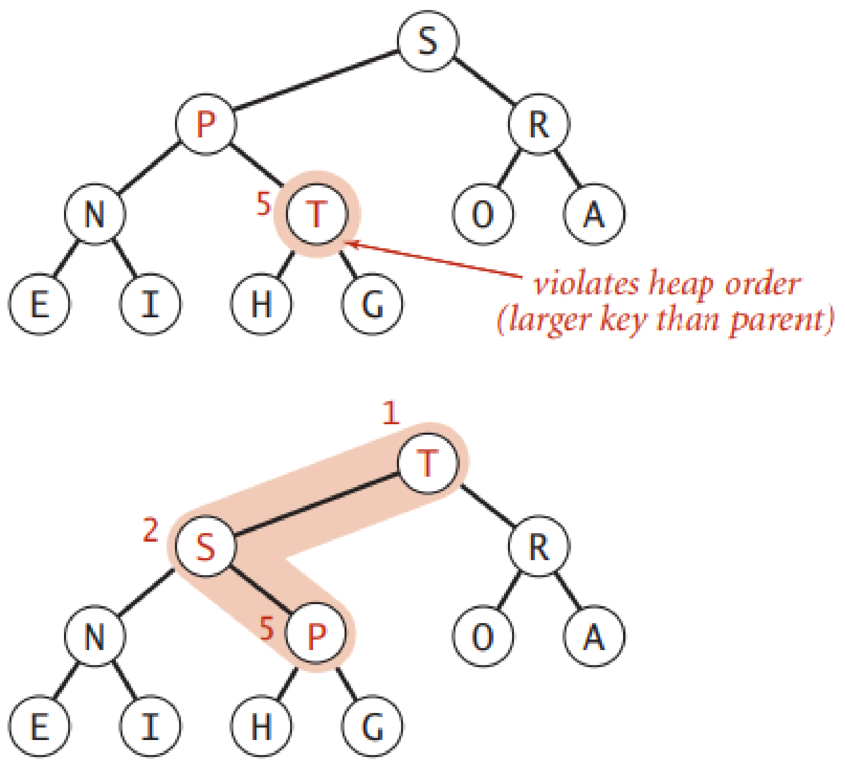
We use these operations to correct binary trees to satisfy the max heap property

SWIM or (SHIFTUP, BUBBLEUP, INCREASEKEY)

Problem: A key becomes larger than its parent's key.

Solution:

- Swap child with parent
- Repeat until heap order restored



SWIM or (SHIFTUP, BUBBLEUP, INCREASEKEY)

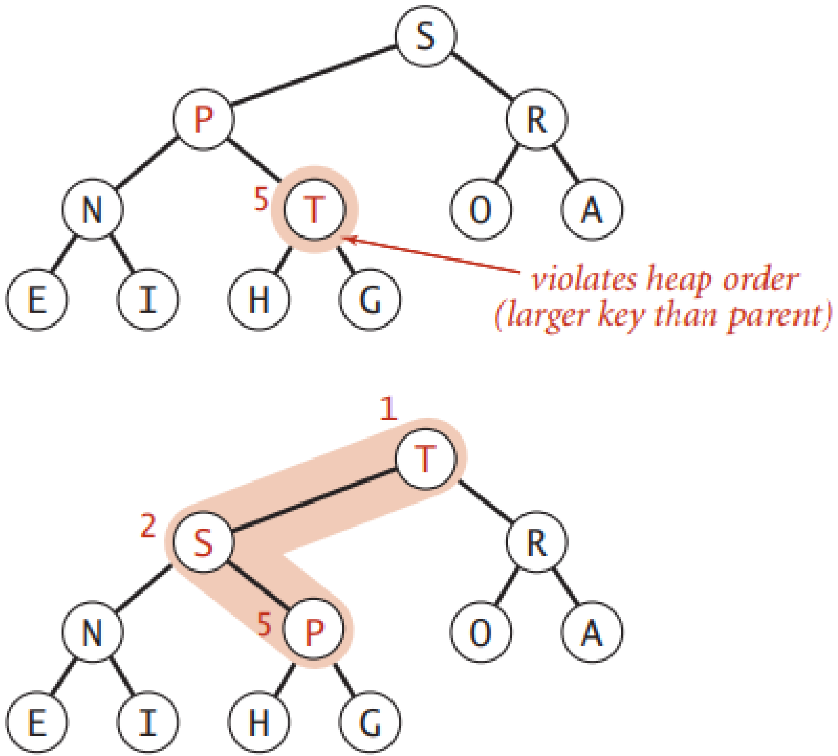
Problem: A key becomes larger than its parent's key.

Solution:

- Swap child with parent
- Repeat until heap order restored

Pseudocode for array implementation

```
function swim(A, k)
    while (k > 1) and A[k/2].key < A[k].key
        swap(A[k], A[k/2])
        k = k/2
```

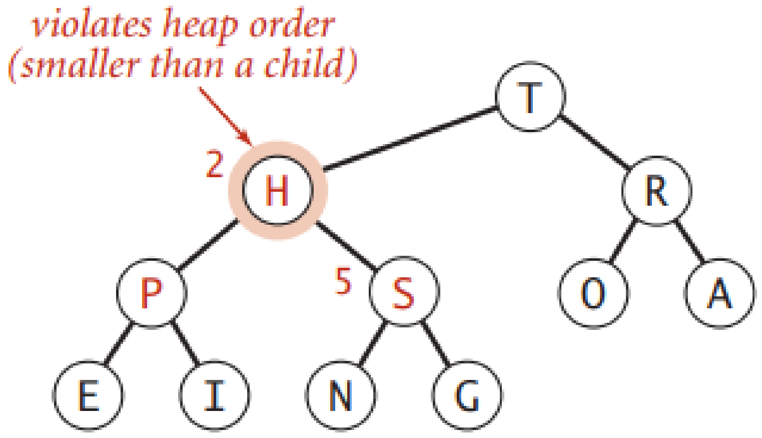


SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

?

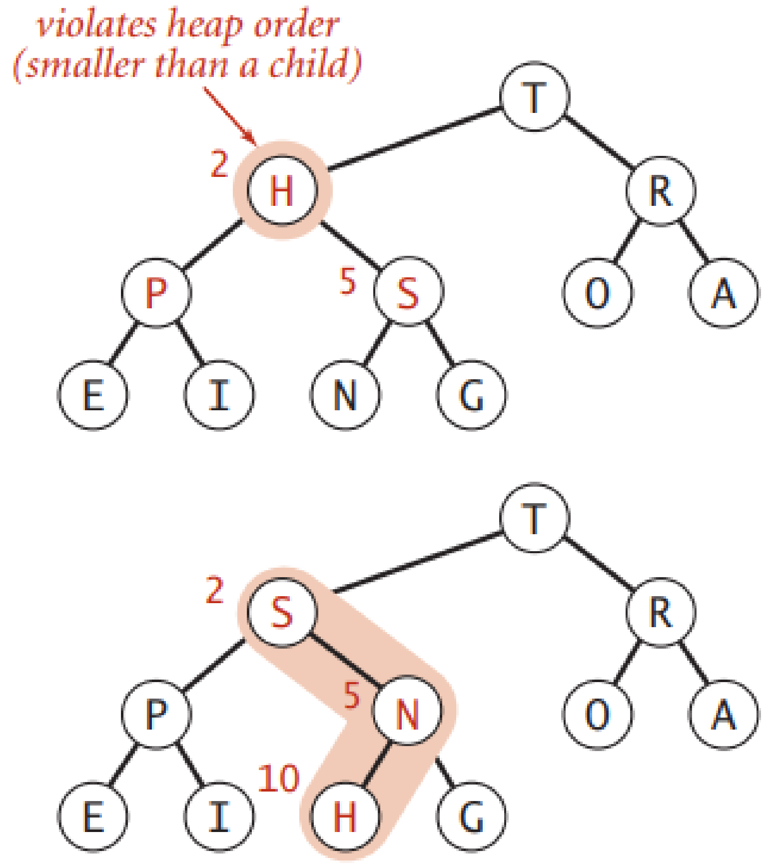


SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

- Swap larger child with parent
- Repeat until heap order restored

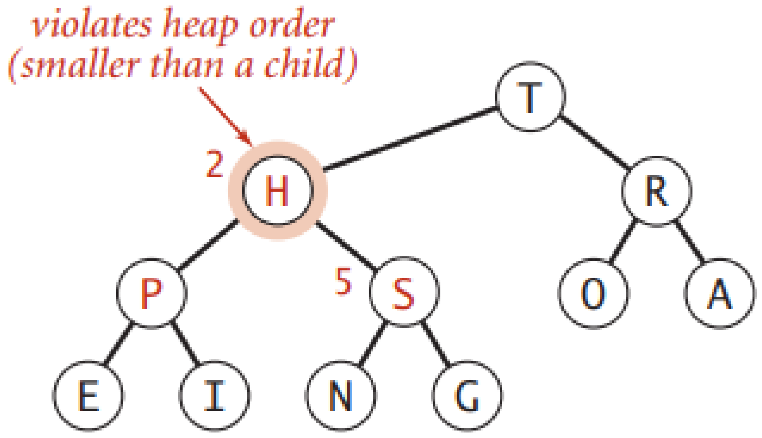


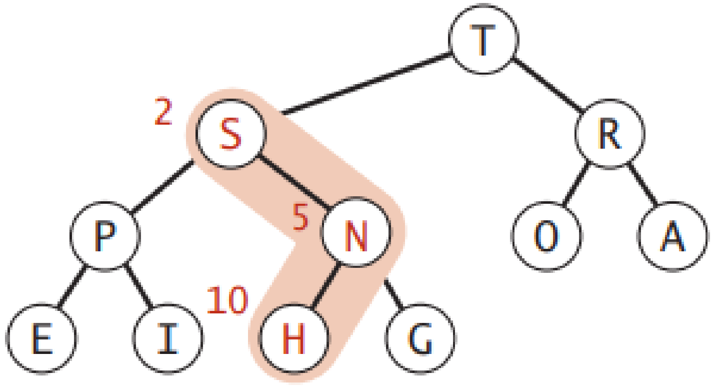
SINK or (SHIFTDOWN, BUBBLEDOWN, HEAPIFY)

Problem: A key becomes smaller than one (or both) of its children's keys.

Solution:

- Swap larger child with parent
- Repeat until heap order restored





Pseudocode for array implementation

```
function sink(A, k)
   while (2k <= n)
         j = 2k
         if (j<n and (A[j].key < A[j+1].key)) j++
         if not (A[k].key < A[j].key) break
         swap(A[k], A[j])
         k = j
```

THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

BINARY HEAP: INSERTION

Idea:

- Add at node at the “end” (first available leaf)
- Swim it up

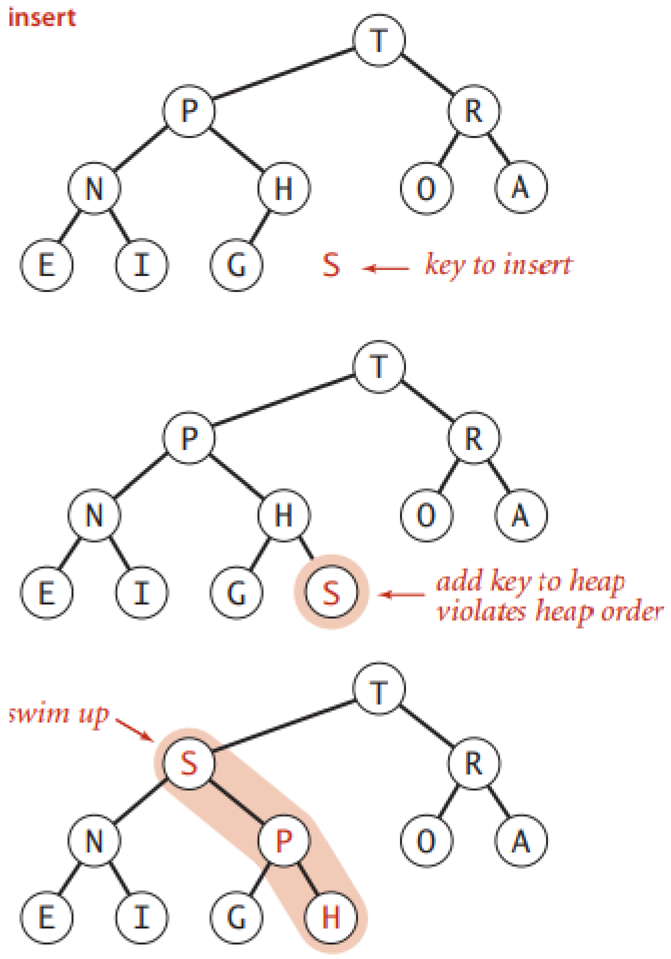
What is the cost of insertion?

A. $O(n)$

B. $O(1)$

C. $O(\log n)$

D. So, so confusing!



BINARY HEAP: INSERTION

Idea:

- Add at node at the “end” (first available leaf)
- Swim it up

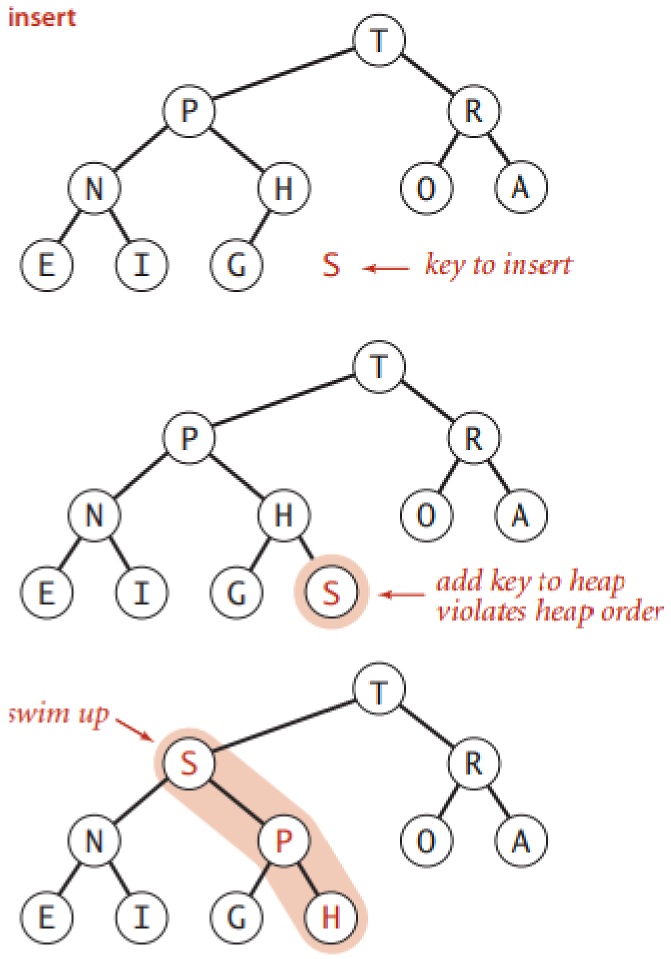
What is the cost of insertion?

A. $O(n)$

B. $O(1)$

C. $O(\log n)$

D. So, so confusing!



Why?

THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

Just return the root! $O(1)$

THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

BINARY HEAP: EXTRACTMAX

Idea:

- Remove the root
- Exchange last element with root
- Sink it down.

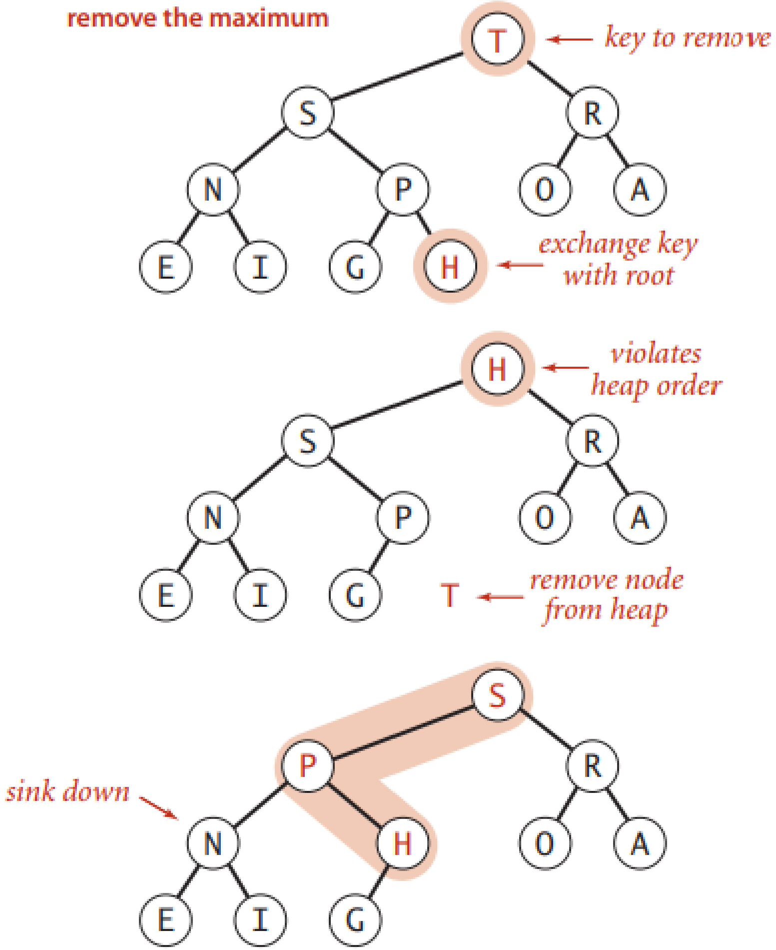
What is the cost of extractMax?

A. $O(n)$

B. $O(1)$

C. $O(\log n)$

D. I love these asymptotic analysis questions!



BINARY HEAP: EXTRACTMAX

Idea:

- Remove the root
- Exchange last element with root
- Sink it down.

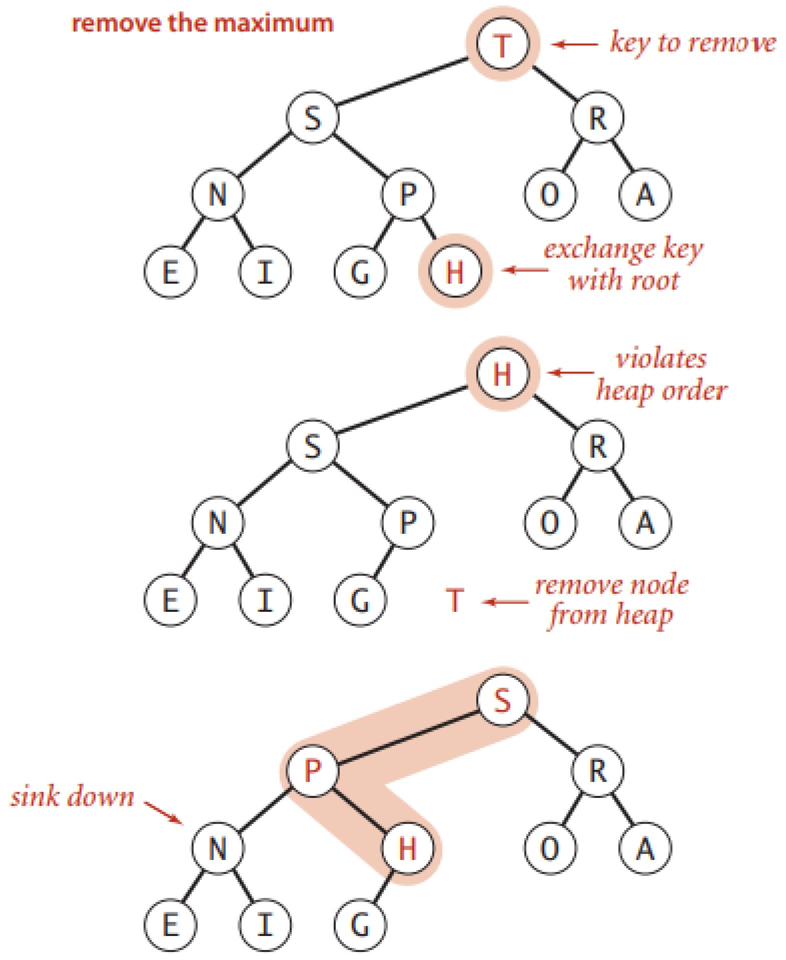
What is the cost of extractMax?

A. $O(n)$

B. $O(1)$

C. $O(\log n)$ **Why?**

D. I love these asymptotic analysis questions!



THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

Poll Everywhere

https://bit.ly/2LvG9bq

CREATE

Given an unsorted array A of elements, how long do we need to create a binary heap?

What is the time cost of `buildHeap`?

A. $O(n \log n)$
B. $O(1)$
C. $O(n)$
D. The answer has to be A.

CREATE

Poll Everywhere
https://bit.ly/2LvG9bq

Given an unsorted array A of elements, how long do we need to create a binary heap?

What is the time cost of `buildHeap`?

A. $O(n \log n)$
B. $O(1)$
C. $\boldsymbol{O(n)}$
D. The answer has to be A.

The idea:
Insert each element one by one.
Each insert takes $O(\log n)$ time.
Total for n elements is $O(n \log n)$

CLEVER CREATION IN $O(n)$ TIME

Invented by Robert Floyd in 1964

- invented invariants (among other things)
- we'll hear about him again in when we meet graphs!

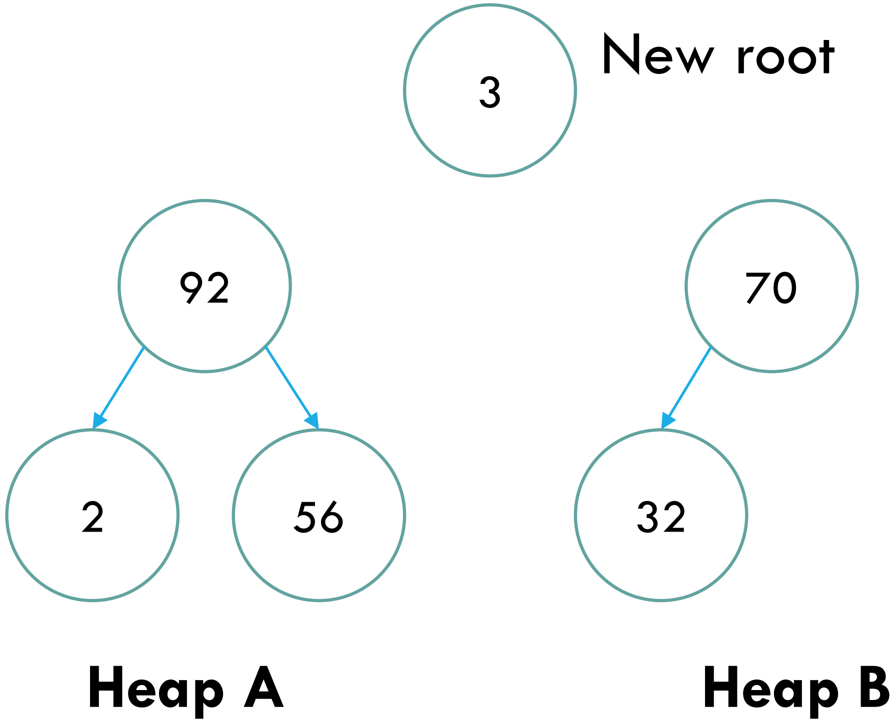
The idea:

- View the input array as a binary tree
- "Bottom up" fixing of the tree to satisfy MaxHeap property

https://en.wikipedia.org/wiki/Robert_W._Floyd

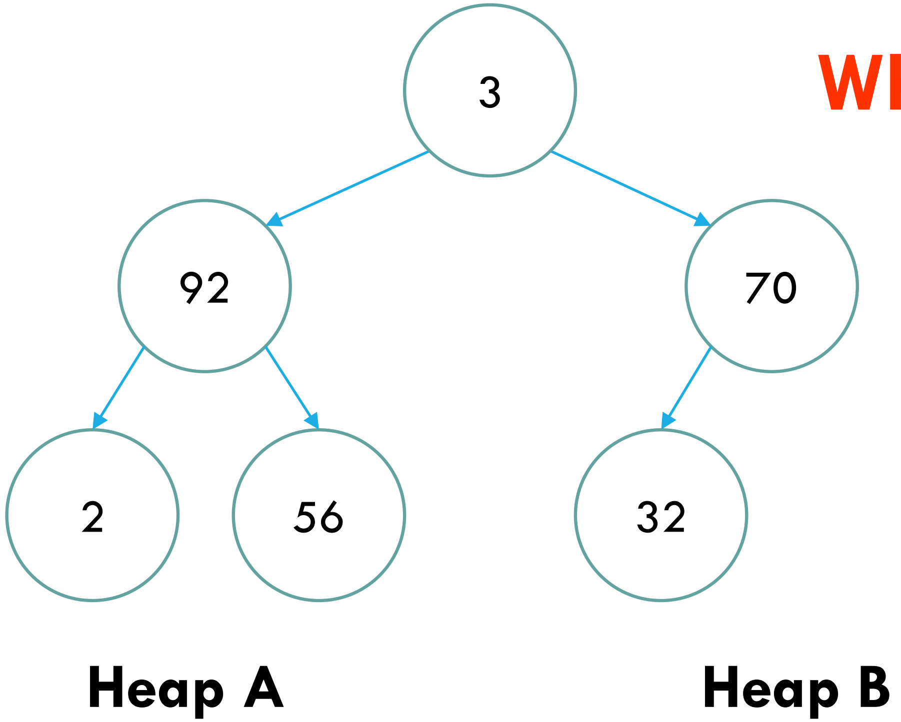
CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to "combine" with a new root node.



CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to "combine" with a new root node.



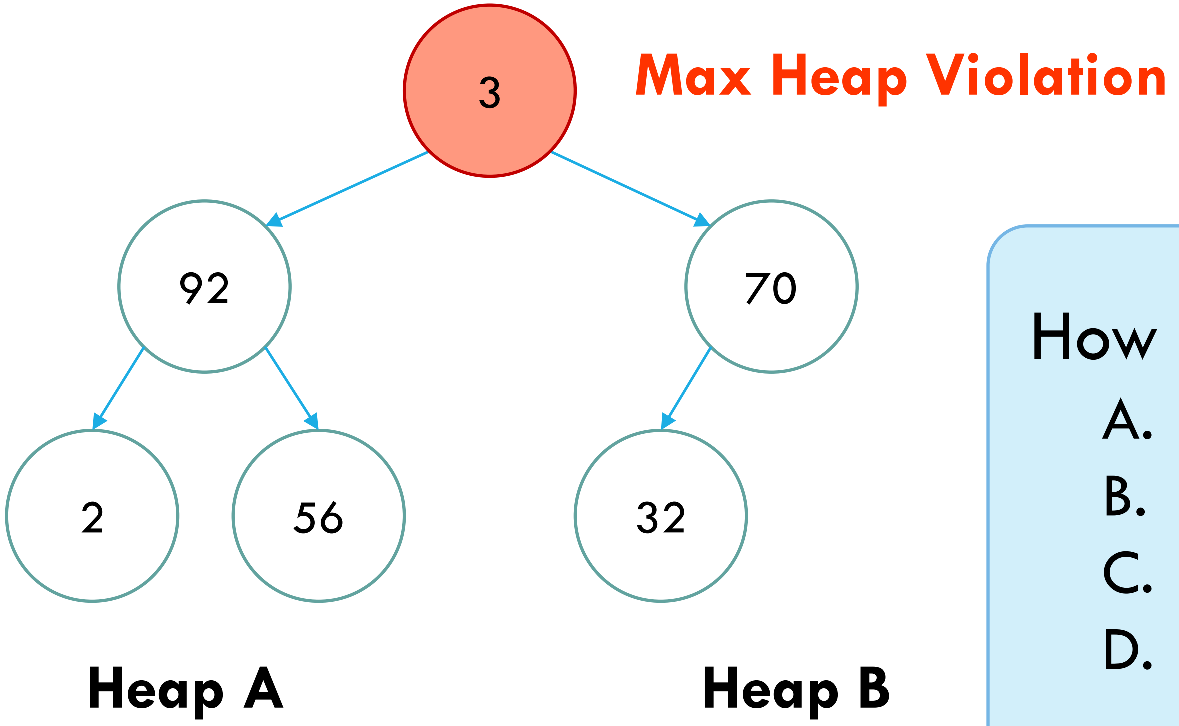
What is the problem?

Poll Everywhere

https://bit.ly/2LvG9bq

CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to "combine" with a new root node.



How can we fix this?

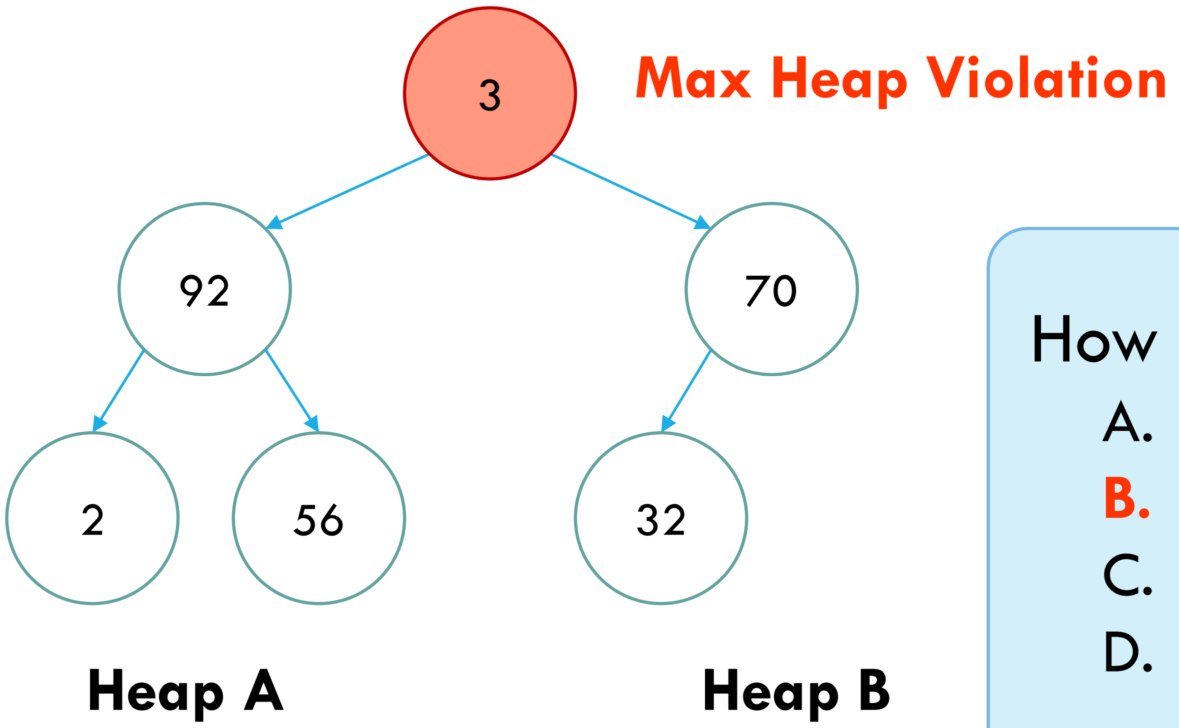
A. Swim!
B. Sink!
C. Sink and swim!
D. Naruto and I are not good swimmers.

Poll Everywhere

https://bit.ly/2LvG9bq

CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to “combine” with a new root node.



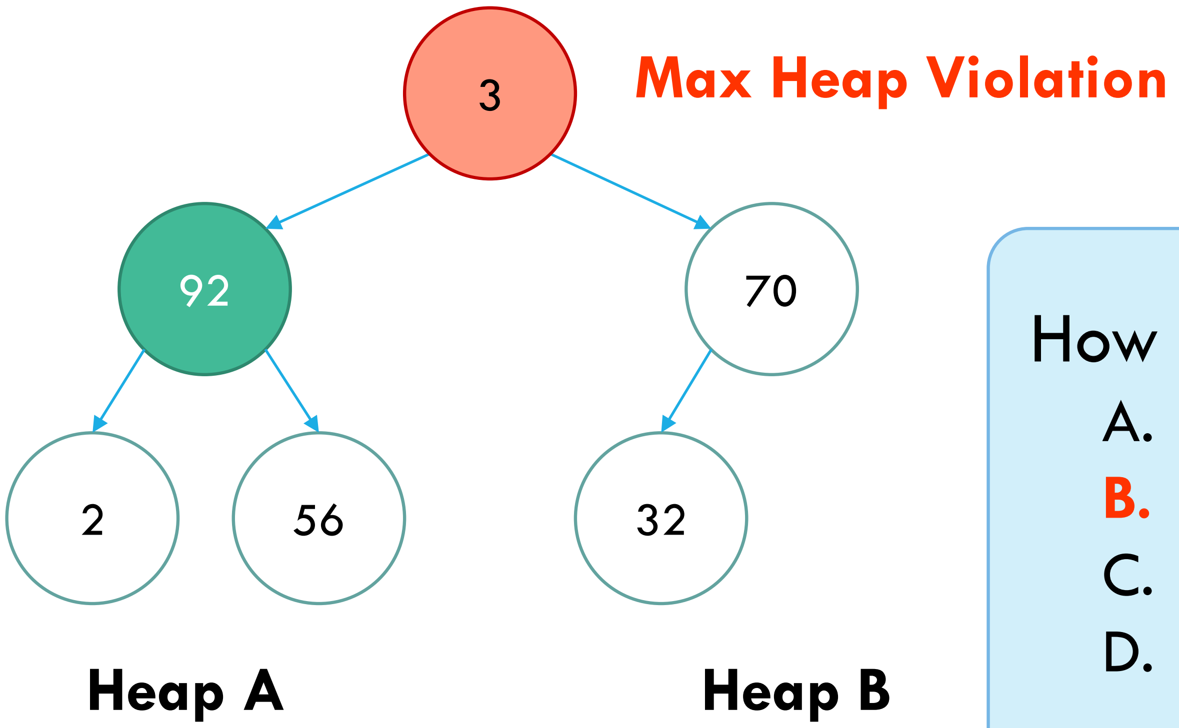
How can we fix this?

A. Swim!
B. **Sink!**
C. Sink and swim!
D. Naruto and I are not good swimmers.

Poll Everywhere
https://bit.ly/2LvG9bq

CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to "combine" with a new root node.



How can we fix this?

A. Swim!
B. Sink!
C. Sink and swim!
D. Naruto and I are not good swimmers.

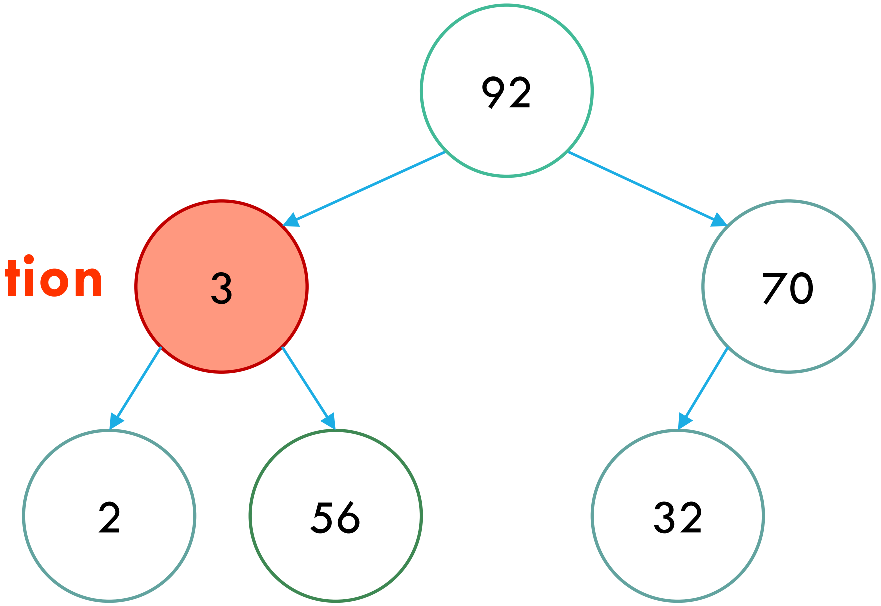
Poll Everywhere

https://bit.ly/2LvG9bq

CLEVER CREATION IN $O(n)$ TIME

Say we have two binary heaps we want to "combine" with a new root node.

Max Heap Violation



Heap A **Heap B**

How can we fix this?

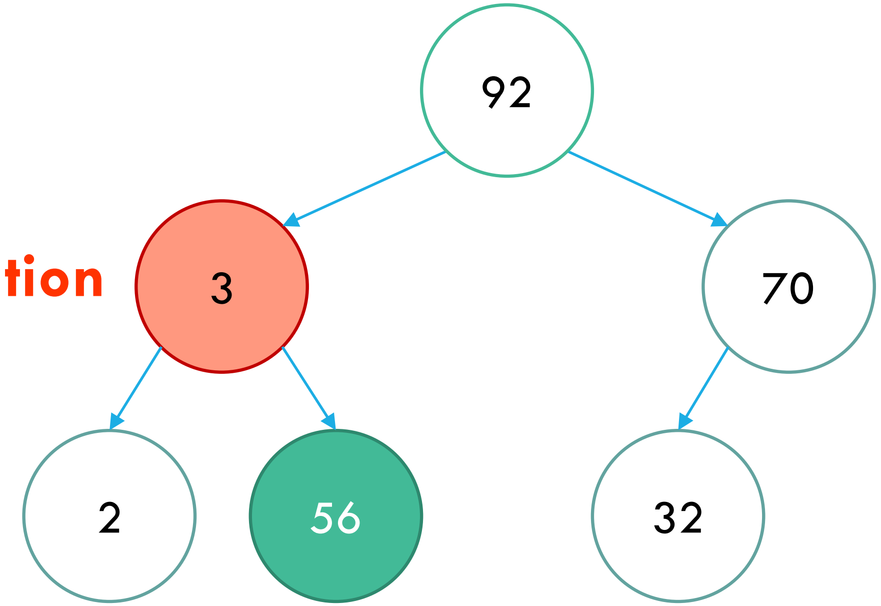
A. Swim!
B. **Sink!**
C. Sink and swim!
D. Naruto and I are not good swimmers.

CLEVER CREATION IN $O(n)$ TIME

Poll Everywhere
https://bit.ly/2LvG9bq

Say we have two binary heaps we want to "combine" with a new root node.

Max Heap Violation



Heap A **Heap B**

How can we fix this?

A. Swim!
B. **Sink!**
C. Sink and swim!
D. Naruto and I are not good swimmers.

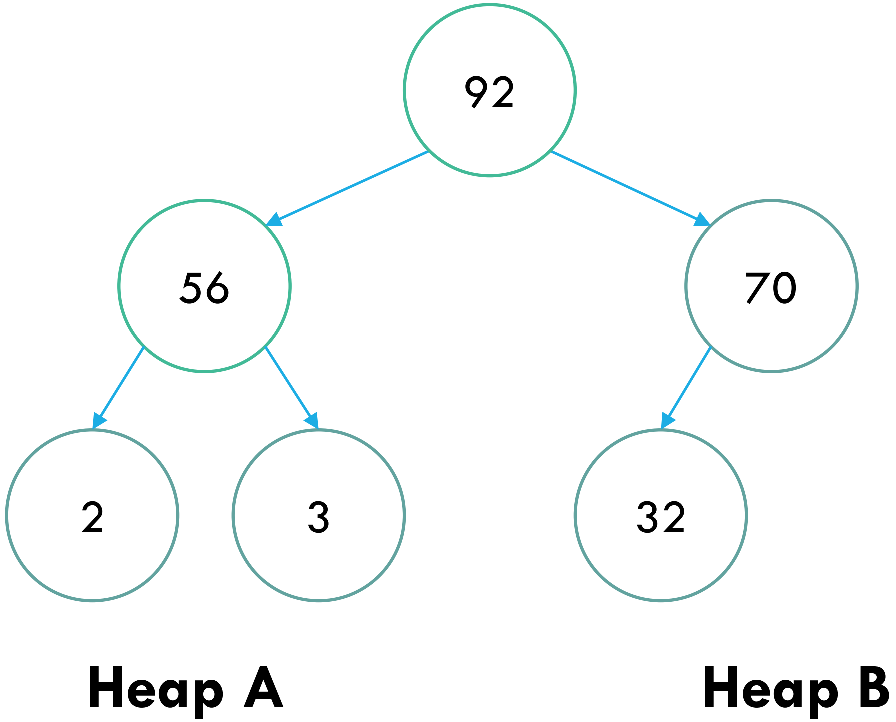
CLEVER CREATION IN $O(n)$ TIME

Poll Everywhere

https://bit.ly/2LvG9bq

Say we have two binary heaps we want to "combine" with a new root node.

Done! ☺



How can we fix this?

A. Swim!

B. Sink!

C. Sink and swim!

D. Naruto and I are not good swimmers.

CLEVER CREATION IN $O(n)$ TIME

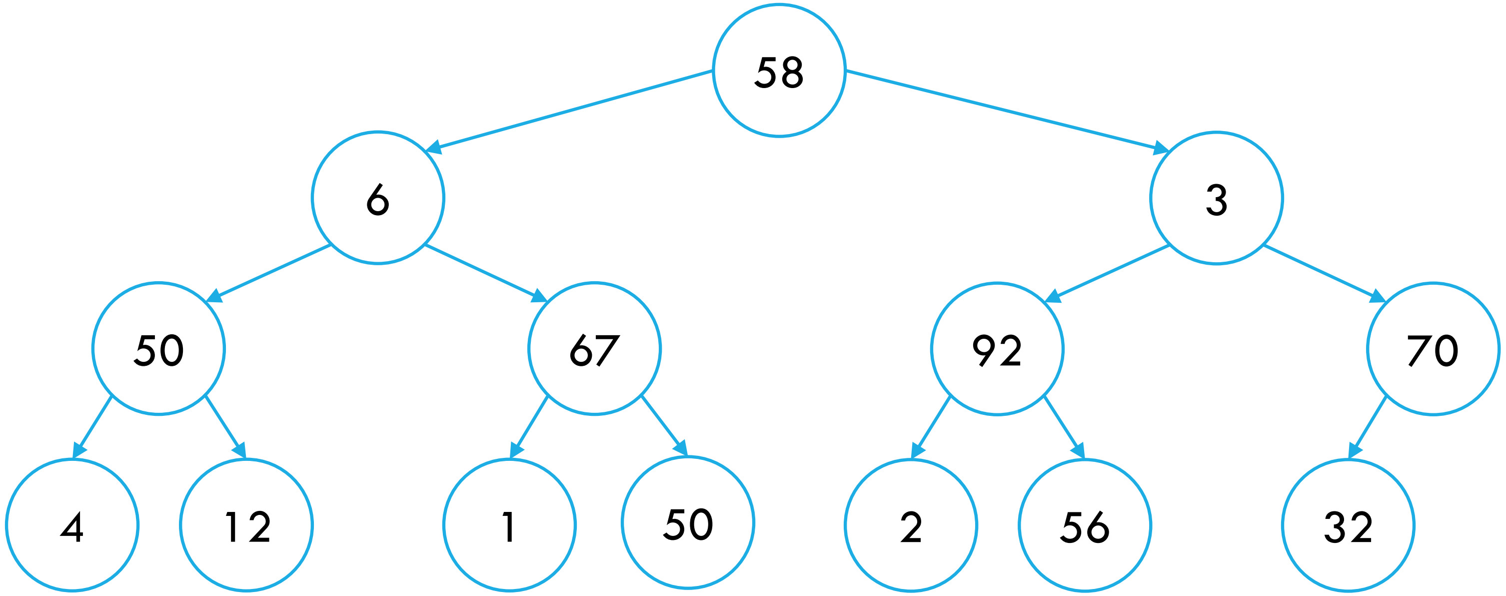
View the input array as a binary heap with violations

58	6	3	12	67	92	70	4	50	1	50	2	56	32

CLEVER CREATION IN $O(n)$ TIME

View the input array as a binary heap with violations

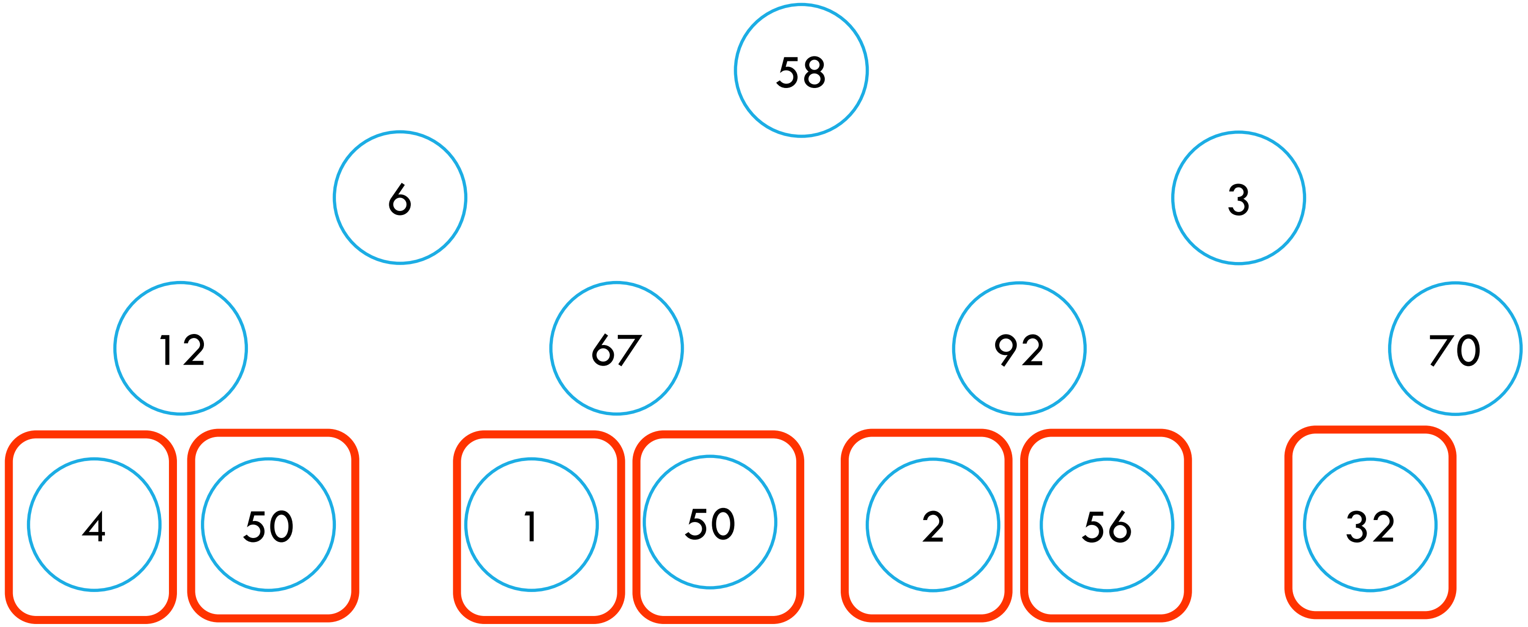
58	6	3	12	67	92	70	4	50	1	50	2	56	32



CLEVER CREATION IN $O(n)$ TIME

View the input array ***as binary heaps*** with violations

58	6	3	12	67	92	70	4	50	1	50	2	56	32

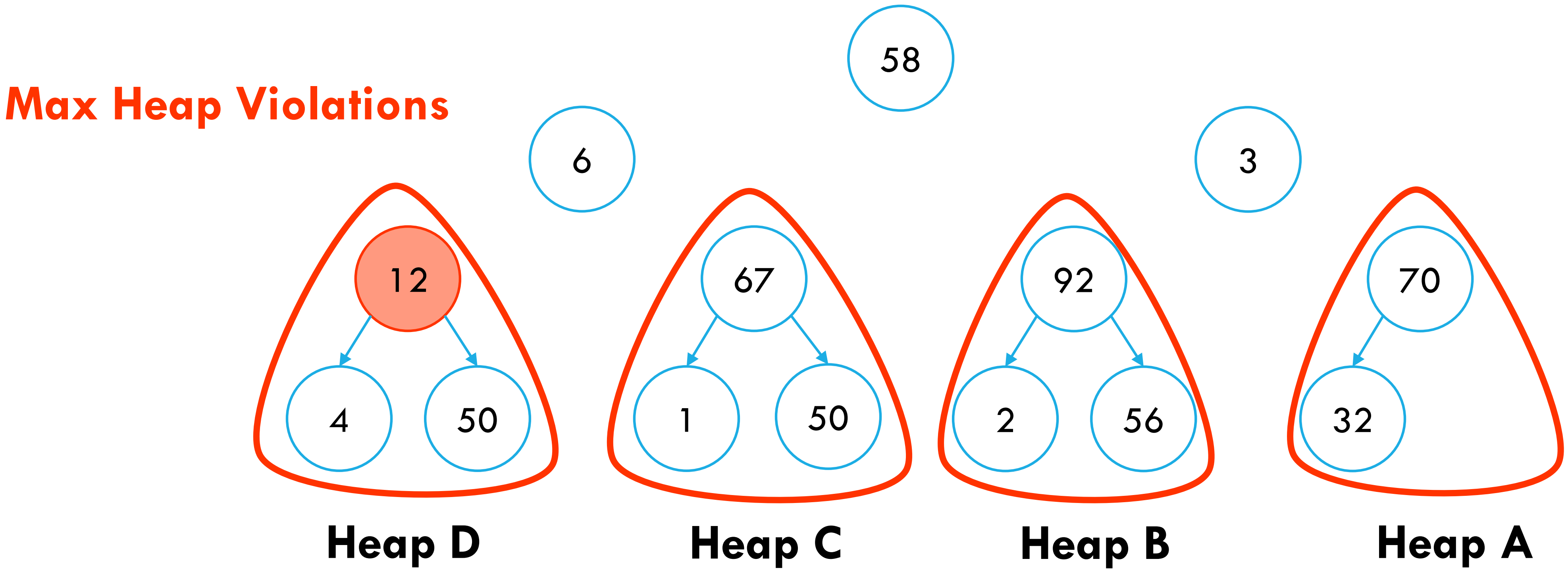


One Element Heaps

CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

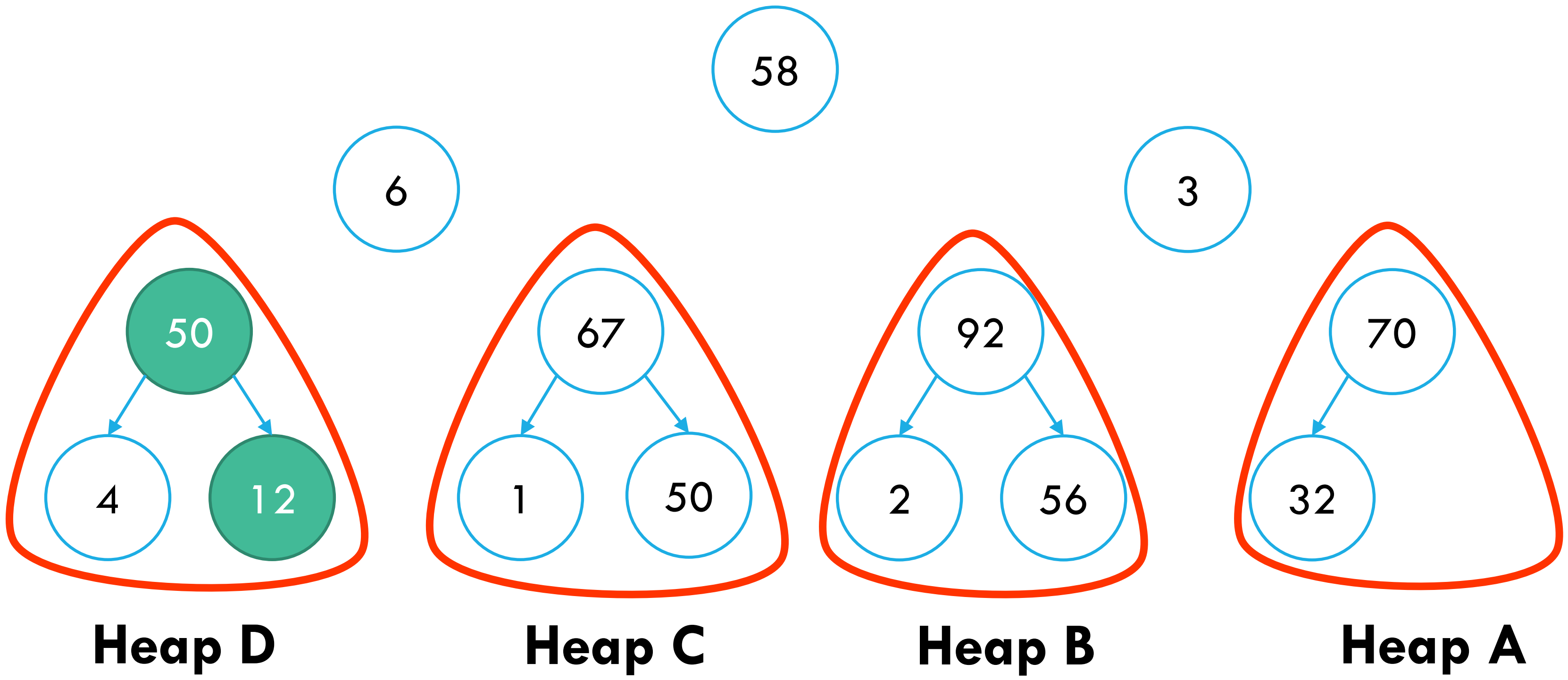
58	6	3	12	67	92	70	4	50	1	50	2	56	32



CLEVER CREATION IN O(n) TIME

View the input array as binary heaps with violations

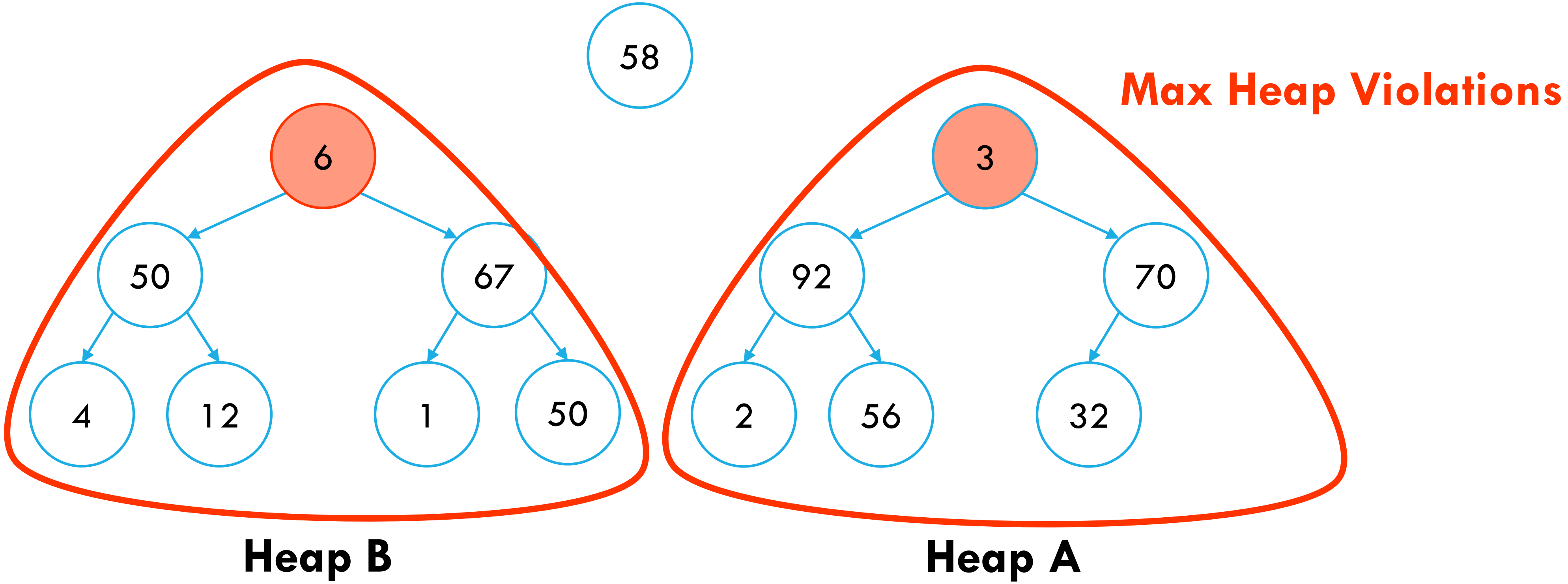
58	6	3	50	67	92	70	4	12	1	50	2	56	32



CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

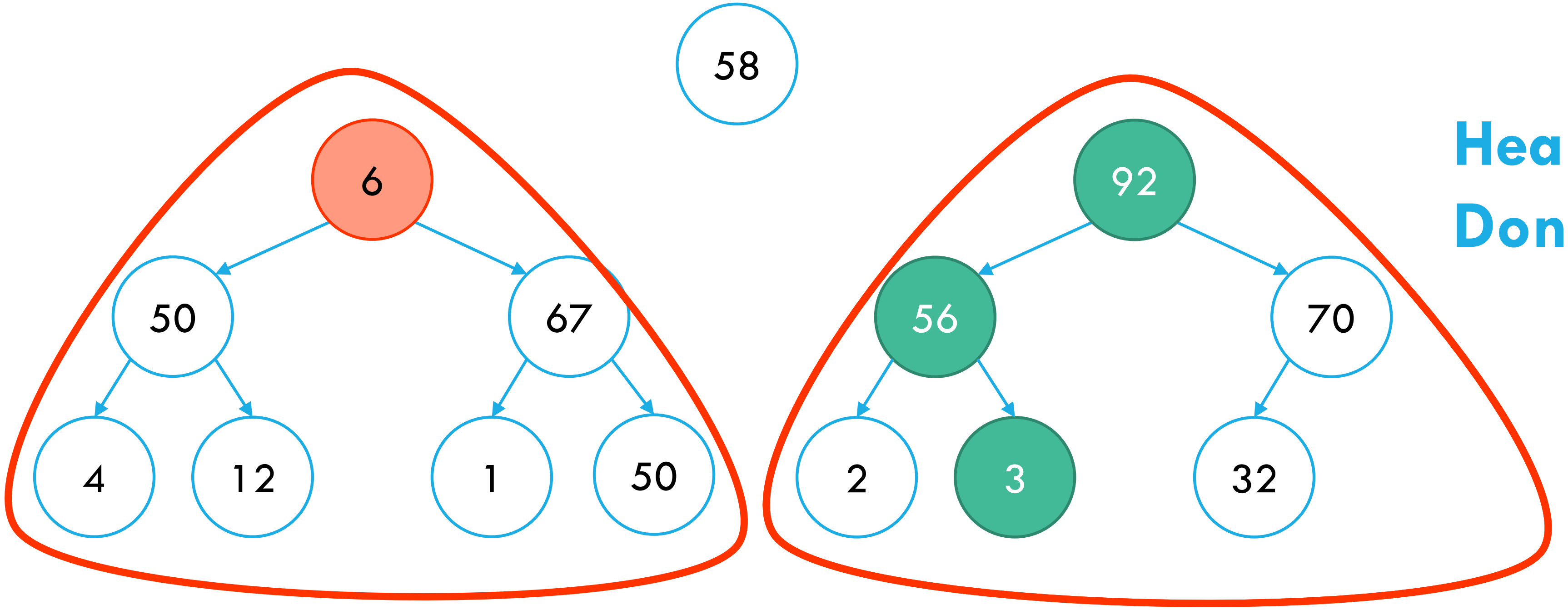
58	6	3	50	67	92	70	4	12	1	50	2	56	32



CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

58	6	92	50	67	56	70	4	12	1	50	2	3	32



Heap B

Heap A

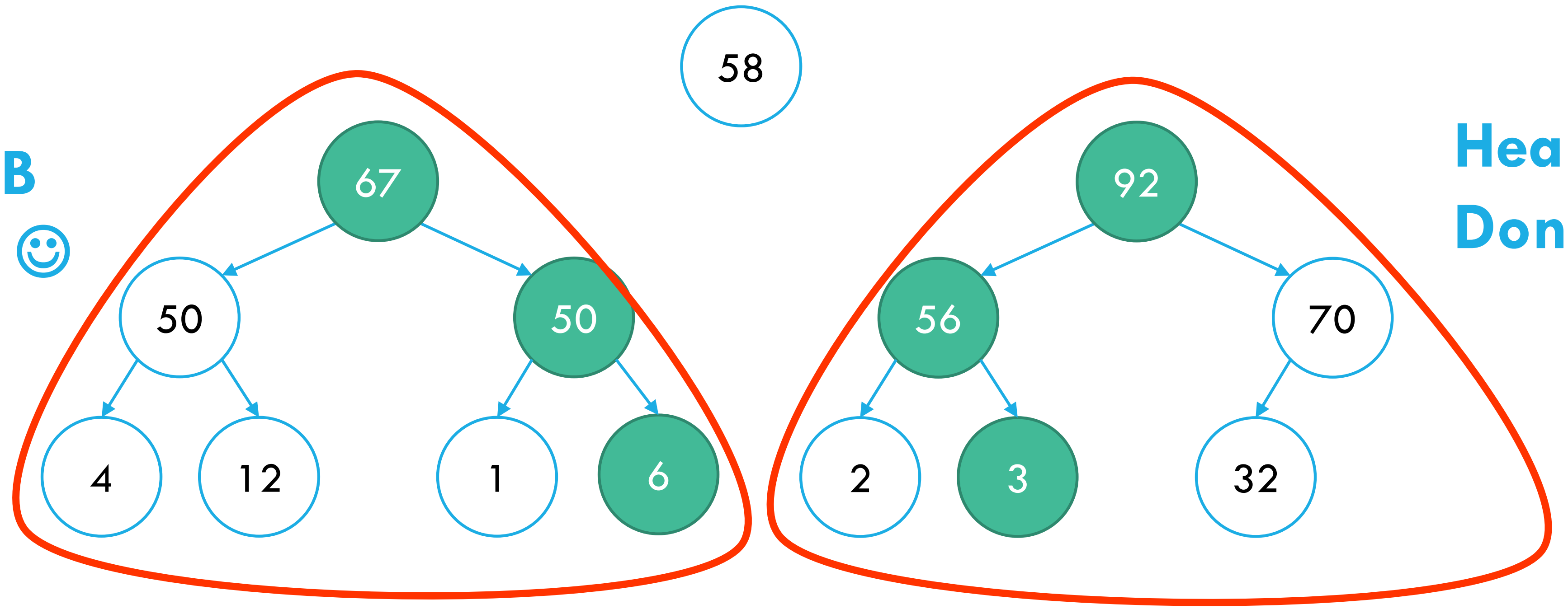
Heap A Done! ☺

CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

58	67	92	50	50	56	70	4	12	1	6	2	3	32

Heap B
Done! ☺



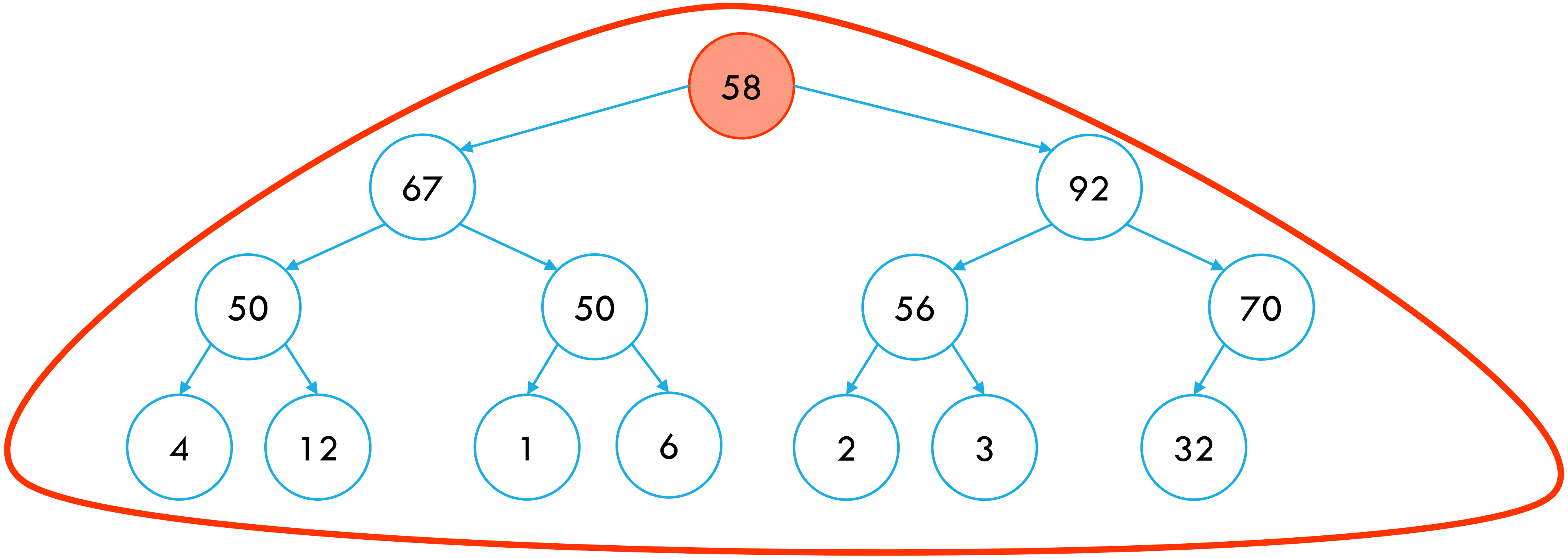
Heap A
Done! ☺

Heap B **Heap A**

CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

58	67	92	50	50	56	70	4	12	1	6	2	3	32

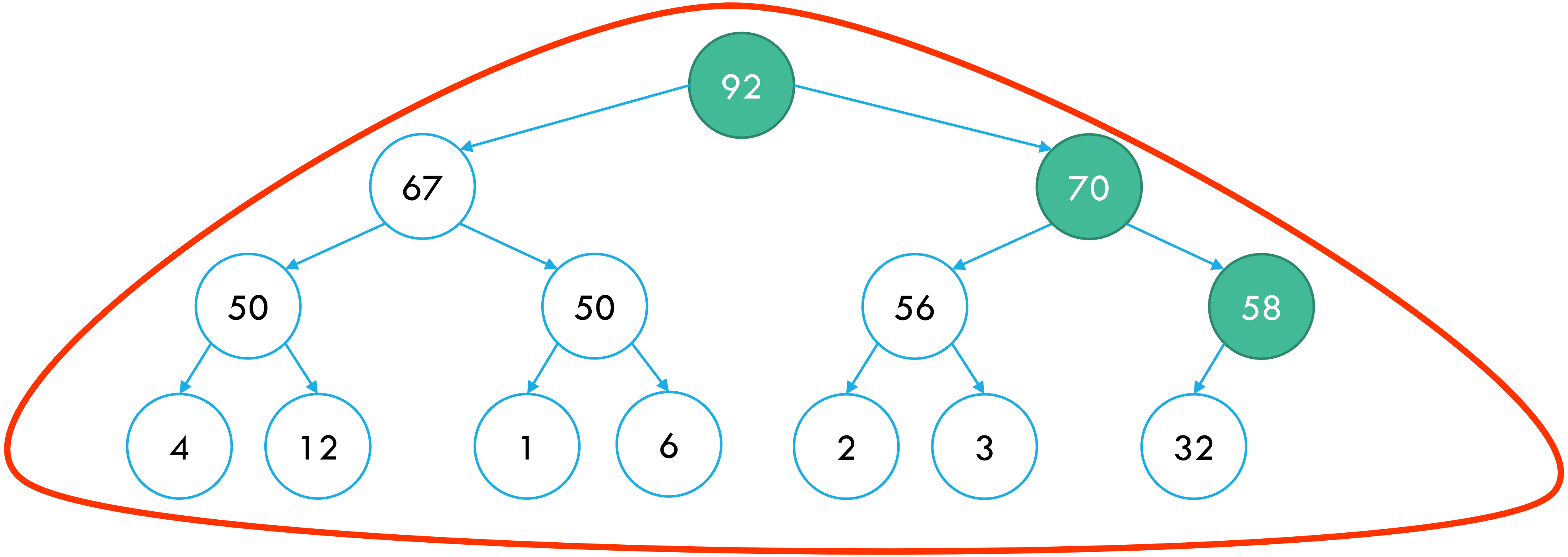


Final Heap

CLEVER CREATION IN O(n) TIME

View the input array as binary heaps with violations

92	67	70	50	50	56	58	4	12	1	6	2	3	32



Final Heap

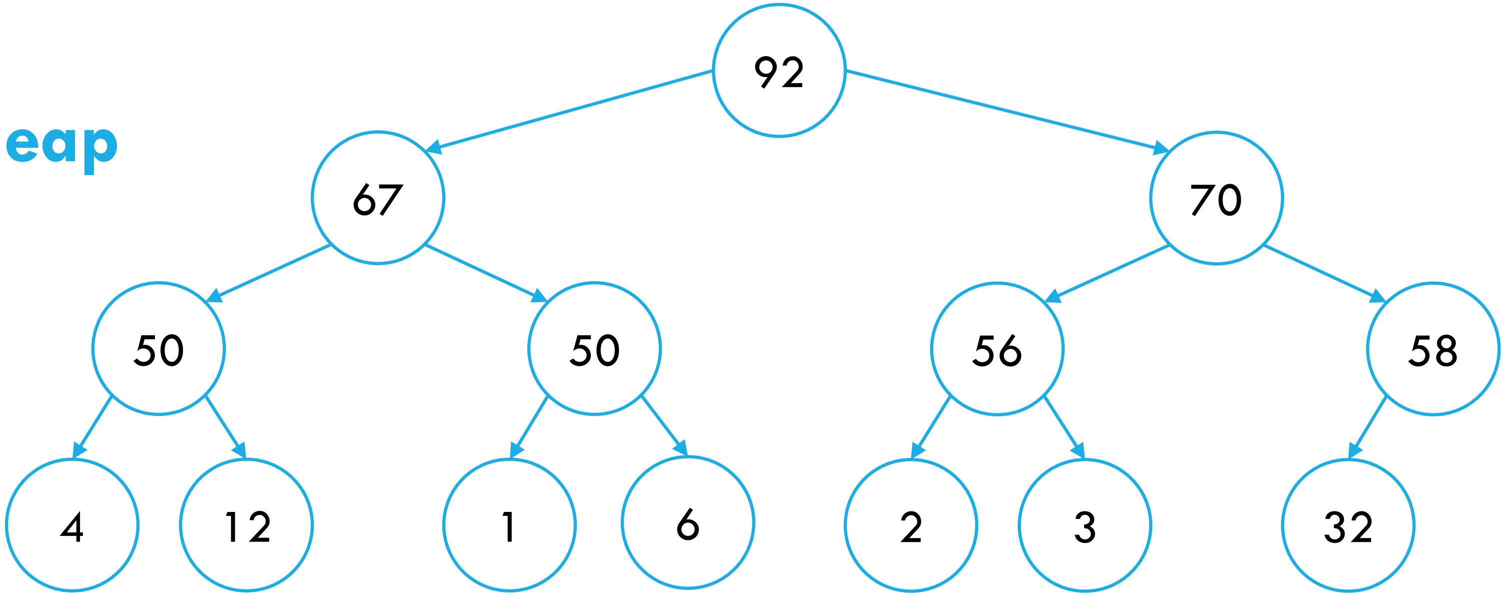
CLEVER CREATION IN $O(n)$ TIME

View the input array as binary heaps with violations

92	67	70	50	50	56	58	4	12	1	6	2	3	32

Final Heap

Done!



Final Heap

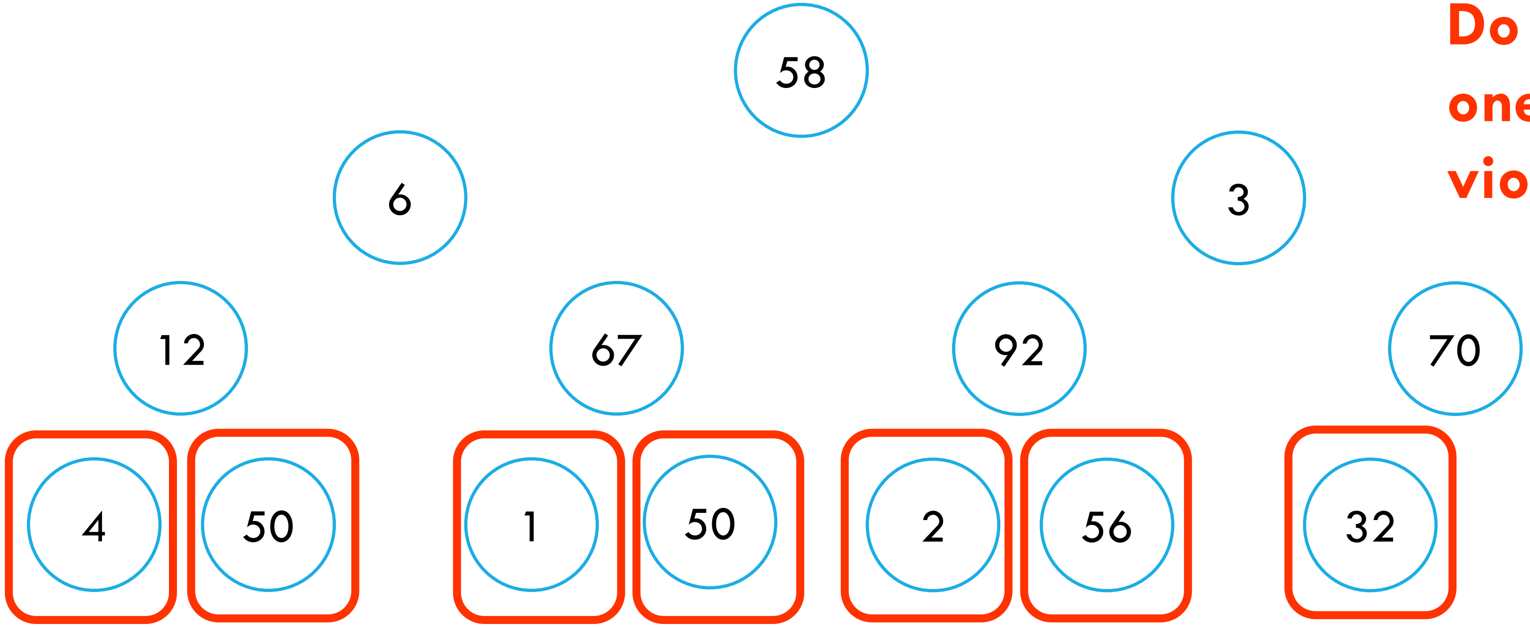
CODE FOR CREATION IN $O(n)$ TIME

```
function buildHeap(A)
    for i from A.length/2 to 1
        sink(i)
```

```
function create(A)
    for i from A.length/2 to 1
        sink(i)
```

CLEVER CREATION IN $O(n)$ TIME

58	6	3	12	67	92	70	4	50	1	50	2	56	32



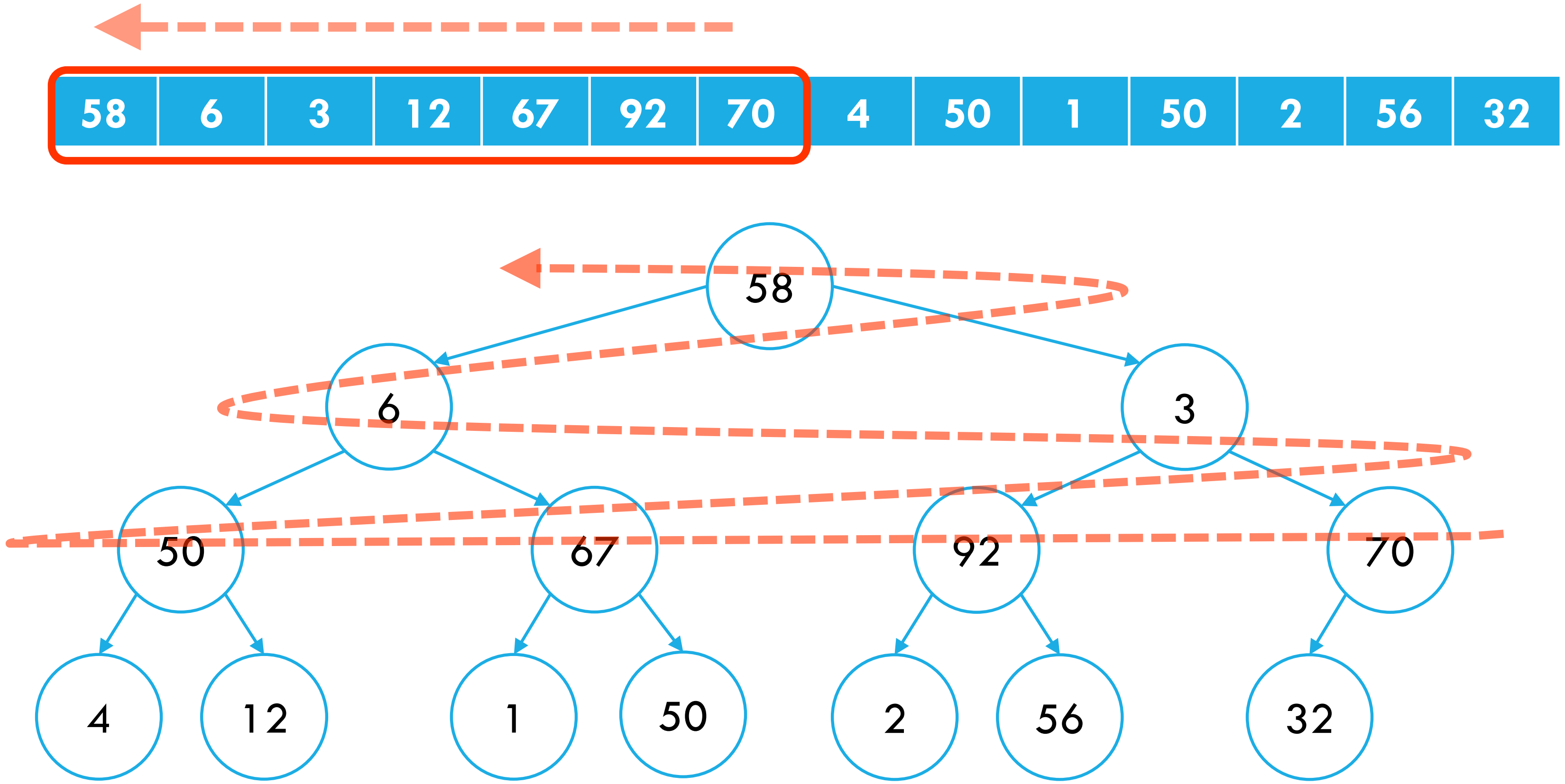
Do we need to check one element heaps for violations?

No

One Element Heaps

CLEVER CREATION IN $O(n)$ TIME

```
function create(A)
    for i from A.length/2 to 1
        sink(i)
```

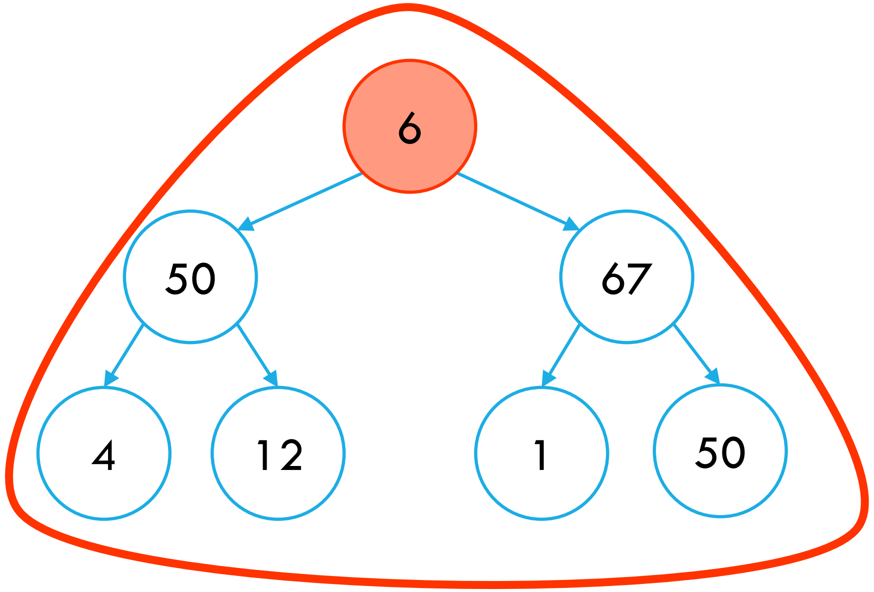


BUT WAIT, I THOUGHT YOU SAID $O(n)$?

It looks like $O(n \log n)$ time?

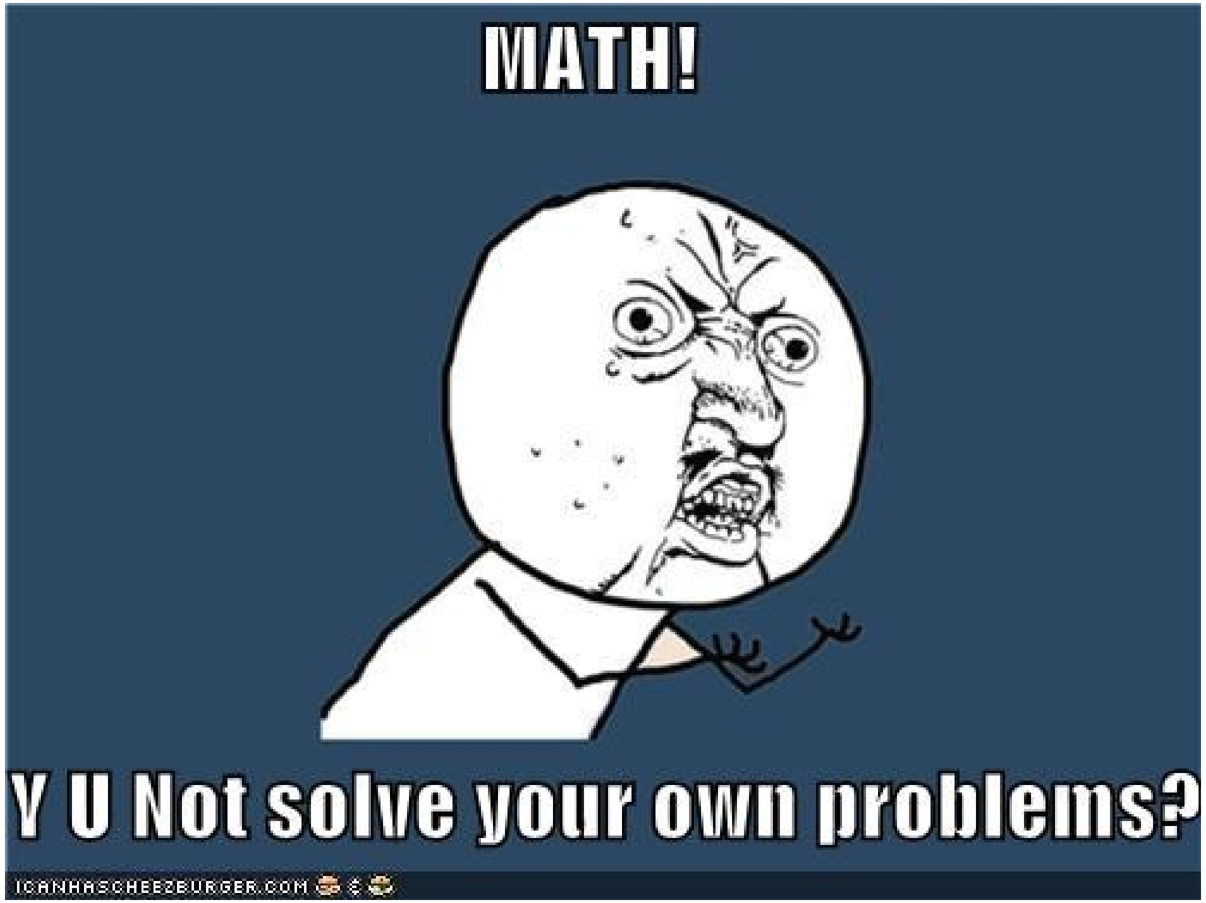
For each violation, we sink to a possible depth of $h = \log n$

Almost correct, but the analysis is too loose.



Heap A

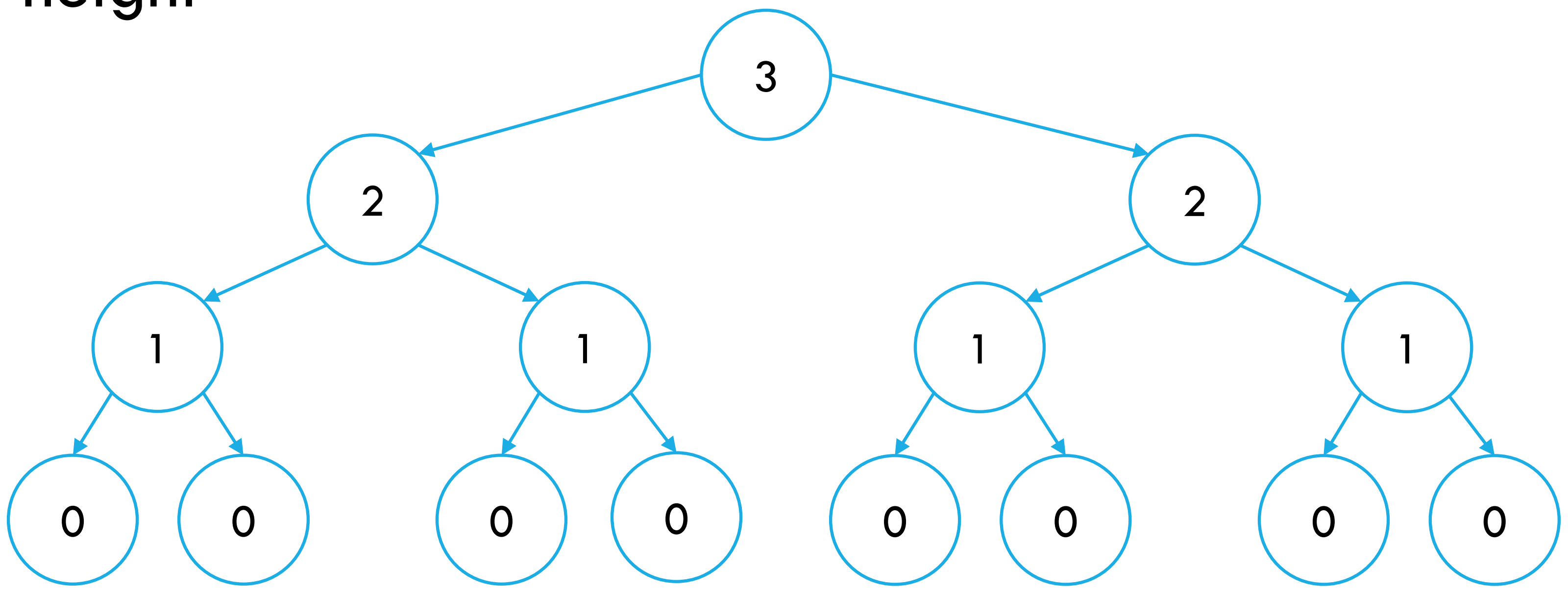
COMPLEXITY ANALYSIS: HERE COMES THE MATH



WHY BUILDHEAP IS $O(n)$

Assume a full binary tree and let $n = 2^{H+1} - 1$

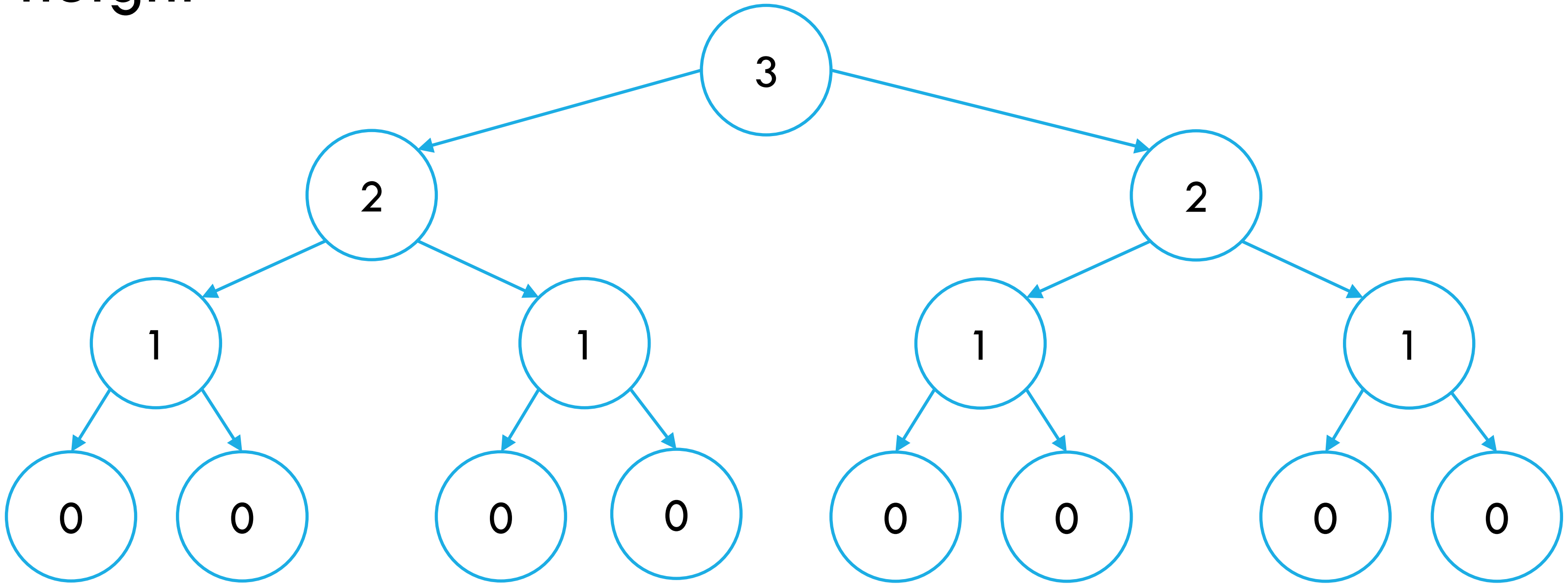
H is the height



WHY BUILDHEAP IS $O(n)$

Assume a full binary tree and let $n = 2^{H+1} - 1$

H is the height



Work Done

(h x num nodes)

$3 * 1 * c = 3c$

$2 * 2 * c = 4c$

$1 * 4 * c = 4c$

$0 * 8 * c = 0$

At each level h, there are 2^{H-h} nodes so, work done per level is $ch2^{H-h}$

WHY BUILDHEAP IS $O(n)$

Work done per level is $ch2^{H-h}$

$$T(n) = \sum_{h=0}^{H} ch2^{H-h}$$

WHY BUILDHEAP IS $O(n)$

Work done per level is $ch2^{H-h}$

$$T(n) = \sum_{h=0}^{H} ch2^{H-h} = \sum_{h=0}^{H} ch\frac{2^H}{2^h}$$

WHY BUILDHEAP IS $O(n)$

Work done per level is $ch2^{H-h}$

$$T(n) = \sum_{h=0}^{H} ch2^{H-h} = \sum_{h=0}^{H} ch\frac{2^H}{2^h} = c2^H \sum_{h=0}^{H} \frac{h}{2^h}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \le c2^H \sum_{h=0}^{\infty} \frac{h}{2^h}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \le c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right)/\left(1-\frac{1}{2}\right)^2 = 2}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \le c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right)/\left(1-\frac{1}{2}\right)^2 = 2} \le c2^H \cdot 2$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \le c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right)/\left(1-\frac{1}{2}\right)^2 = 2} \le c2^H \cdot 2 = c2^{H+1}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \leq c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right)/\left(1-\frac{1}{2}\right)^2 = 2} \leq c2^H \cdot 2 = c2^{H+1}$$

Remember that $n = 2^{H+1} - 1$ so, $n + 1 = 2^{H+1}$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \le c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right) / \left(1 - \frac{1}{2}\right)^2 = 2} \le c2^H \cdot 2 = c2^{H+1}$$

Remember that $n = 2^{H+1} - 1$ so, $n + 1 = 2^{H+1}$

$$T(n) \le c2^{H+1}$$

WHY BUILDHEAP IS $O(n)$

$$T(n) = c2^H \sum_{h=0}^{H} \frac{h}{2^h} \leq c2^H \overbrace{\sum_{h=0}^{\infty} \frac{h}{2^h}}^{\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \left(\frac{1}{2}\right)/\left(1-\frac{1}{2}\right)^2 = 2} \leq c2^H \cdot 2 = c2^{H+1}$$

Remember that $n = 2^{H+1} - 1$ so, $n + 1 = 2^{H+1}$

$$T(n) \leq c2^{H+1} = c(n+1) = O(n)$$

Done! ☺

THE (MAX) PRIORITY QUEUE ADT

Operations:

- `insert(x)` : inserts x
- `max()` : returns element with the highest priority
- `extractMax()` : returns and remove the highest priority element
- `size()` : returns the size of the queue
- `buildHeap(A)` : creates a priority queue from an array of patients

PROBLEM SOLVED: PRIORITIZING PATIENTS AT THE HOSPITAL

Your startup gets a nice paycheck

and you helped sick people get the help they needed!

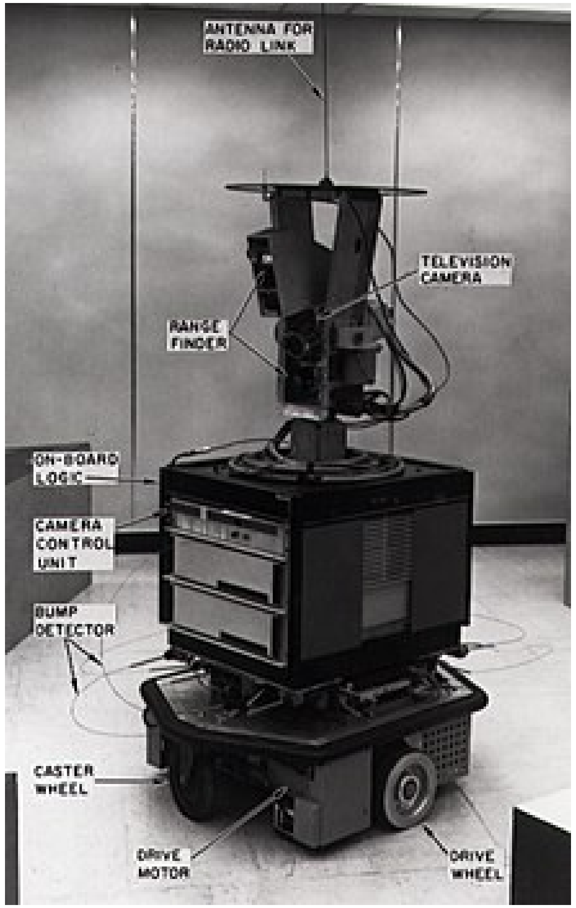
PRIORITY QUEUE'S MANY APPLICATIONS

Artificial Intelligence: A*search

Graph searching: Dijkstra's algorithm, Prim's alg

Simulation: Patient Queues, Colliding particles

Statistics: Online median in a data stream



Shakey the robot which uses A* search for planning

QUESTIONS?

CAN WE SORT USING A BINARY HEAP?

Poll Everywhere

https://bit.ly/2LvG9bq

What is the computational complexity of sorting using a binary heap?

A. $O(n)$
B. $O(n^2)$
C. $O(n \log n)$
D. $O(2^{\log n})$
E. Impossible to do!

Poll Everywhere

https://bit.ly/2LvG9bq

CAN WE SORT USING A BINARY HEAP?

What is the computational complexity of sorting using a binary heap?

A. $O(n)$
B. $O(n^2)$
C. $\boldsymbol{O(n \log n)}$
D. $O(2^{\log n})$
E. Impossible to do!

THE STRAIGHTFORWARD IDEA

```
function heapsort(A)
    n = length(A)
    sorted = new Array[n]

    binheap = buildHeap(A)
    for i = n-1 to 0
        a = binheap.extractMax()
        sorted[i] = a
```

Can we do better?

A. Yes, we can!
B. No, this is optimal.
C. Judging from your past questions, I think you're going to say "Yes" so, I pick A.
D. I pass.

THE STRAIGHTFORWARD IDEA

```
function heapsort(A)
    n = length(A)
    sorted = new Array[n]

    binheap = buildHeap(A)
    for i = n-1 to 0
        a = binheap.extractMax()
        sorted[i] = a
```

how much space does this take?

$$O(n)$$

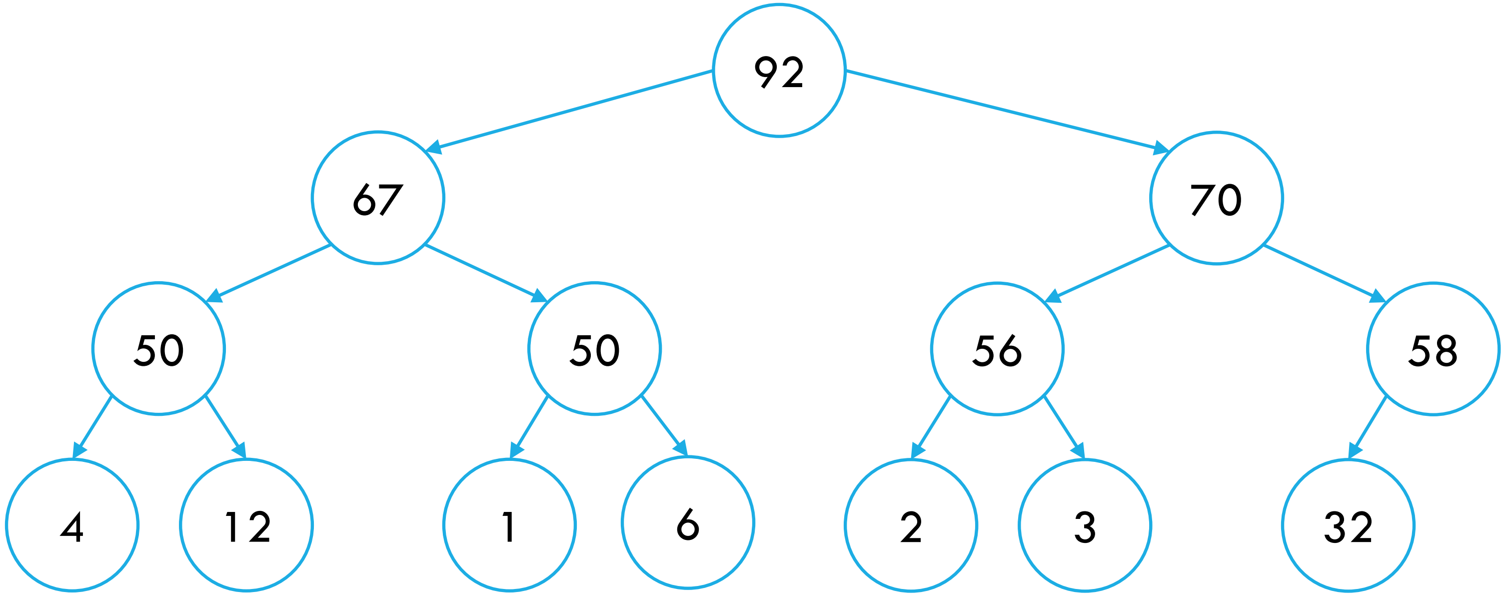
Can we do better?

A. **Yes, we can!**
B. No, this is optimal.
C. Judging from your past questions, I think you're going to say "Yes" so, I pick A.
D. I pass.

THE IDEA: SWAP WITH THE END, THEN SINK

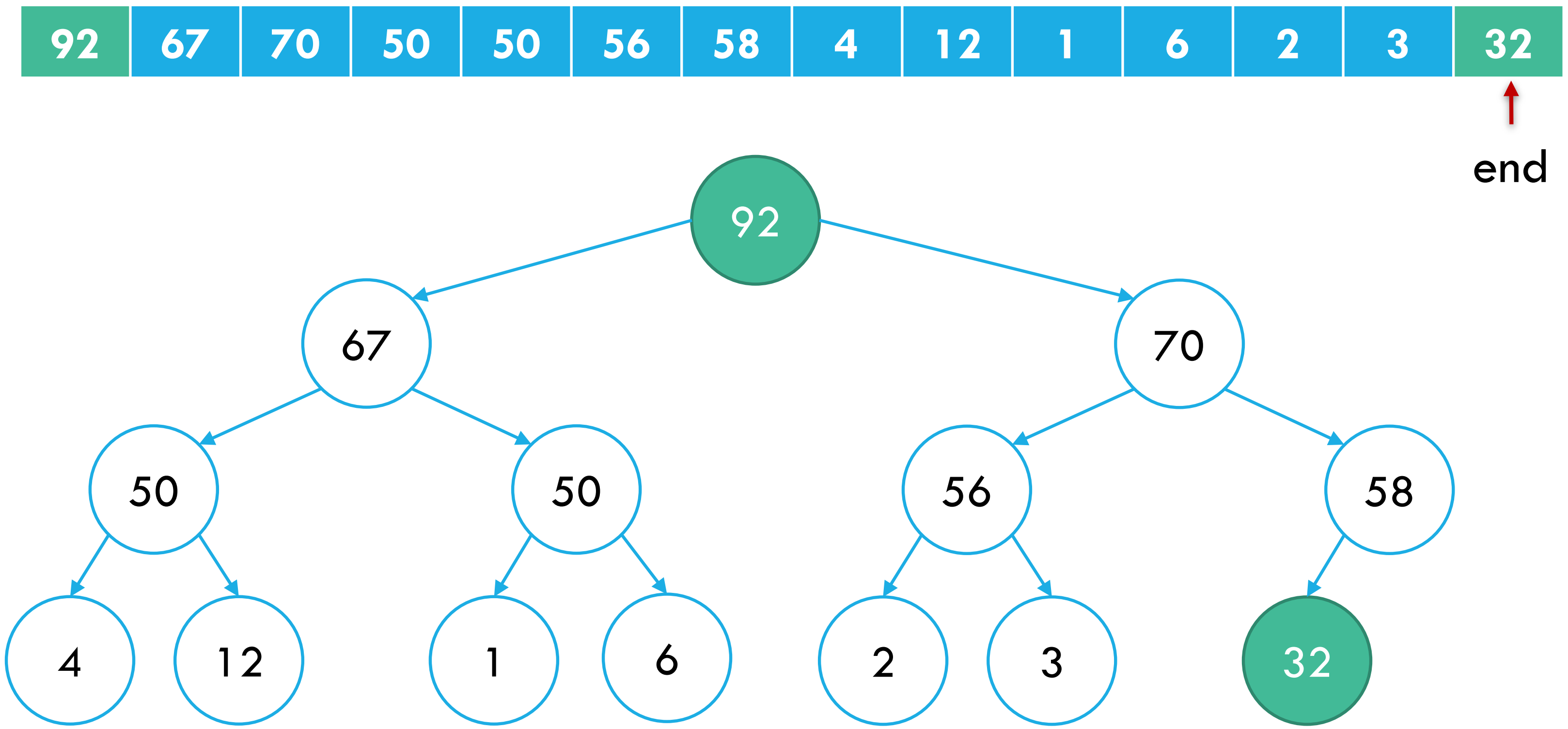
92	67	70	50	50	56	58	4	12	1	6	2	3	32

end



THE IDEA: SWAP WITH THE END, THEN SINK

Swap the green elements

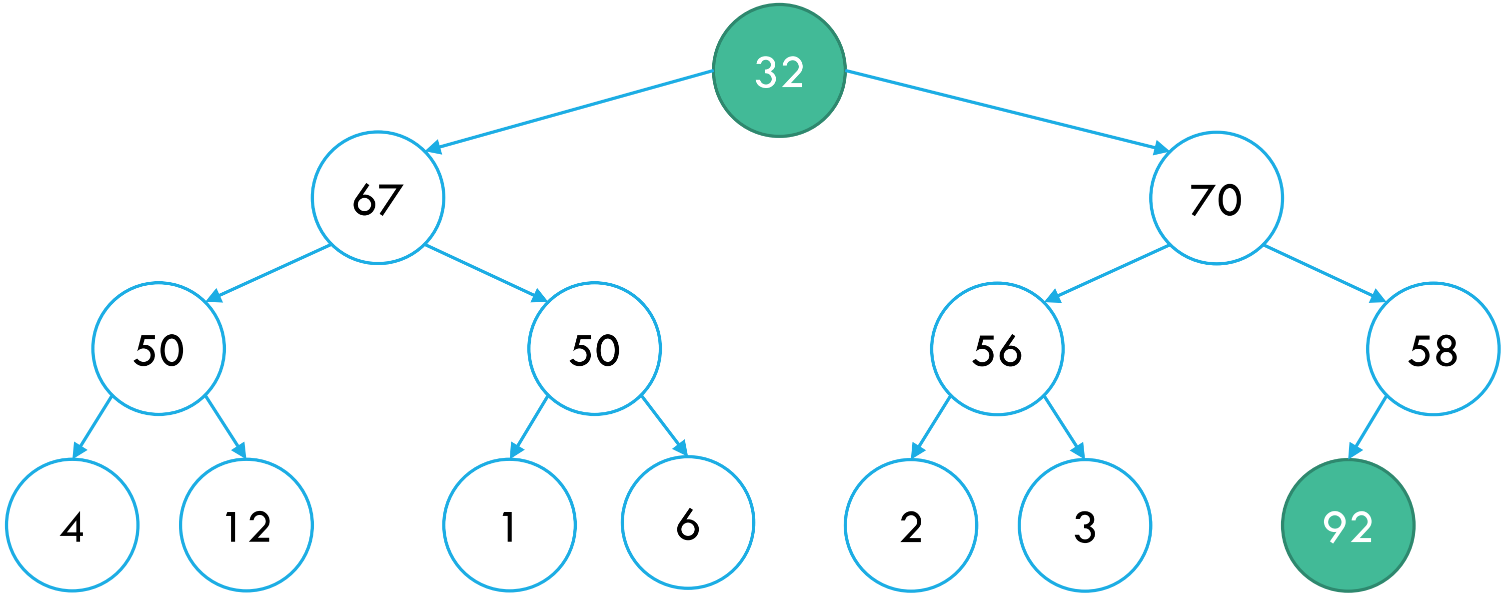


THE IDEA: SWAP WITH THE END, THEN SINK

Swap the green elements

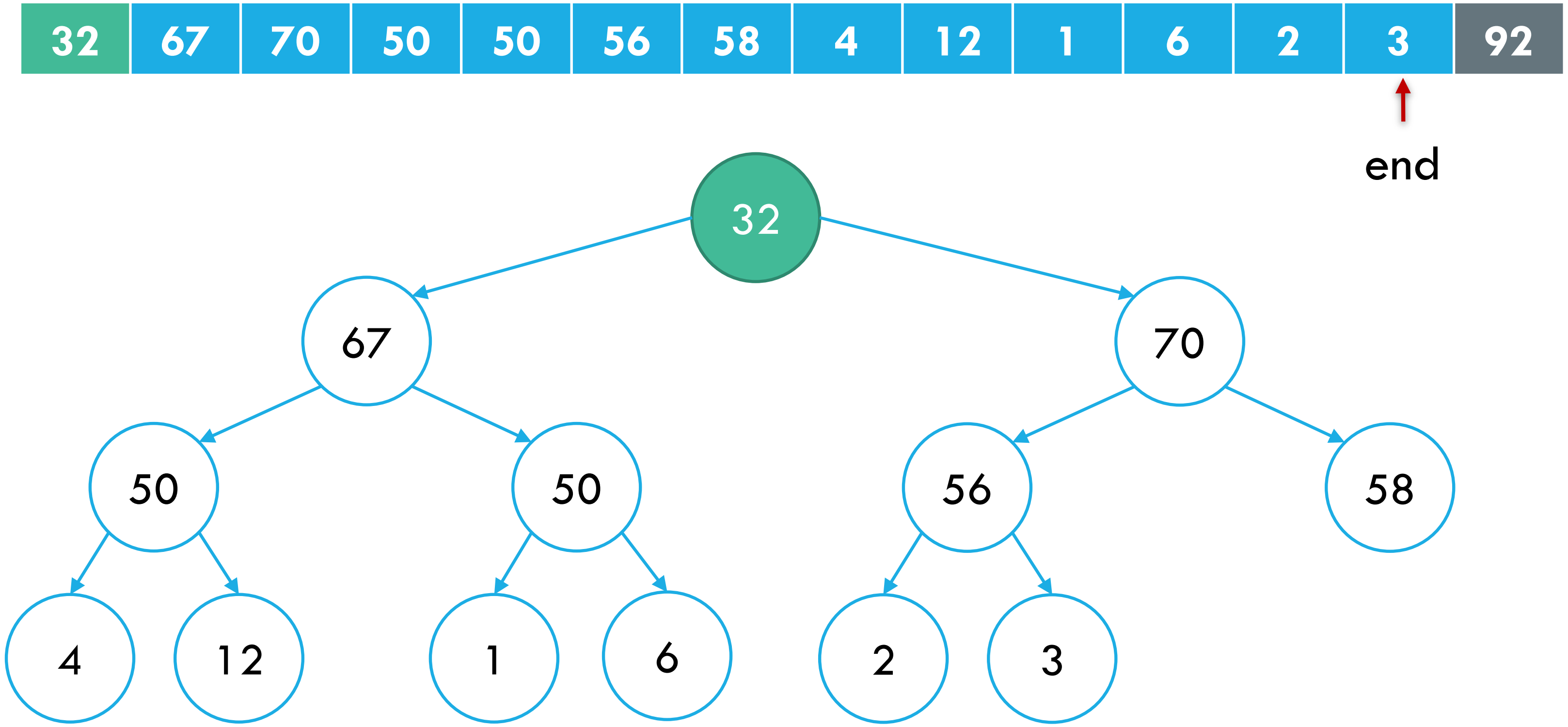
32	67	70	50	50	56	58	4	12	1	6	2	3	92

end



THE IDEA: SWAP WITH THE END, THEN SINK

Move the end marker

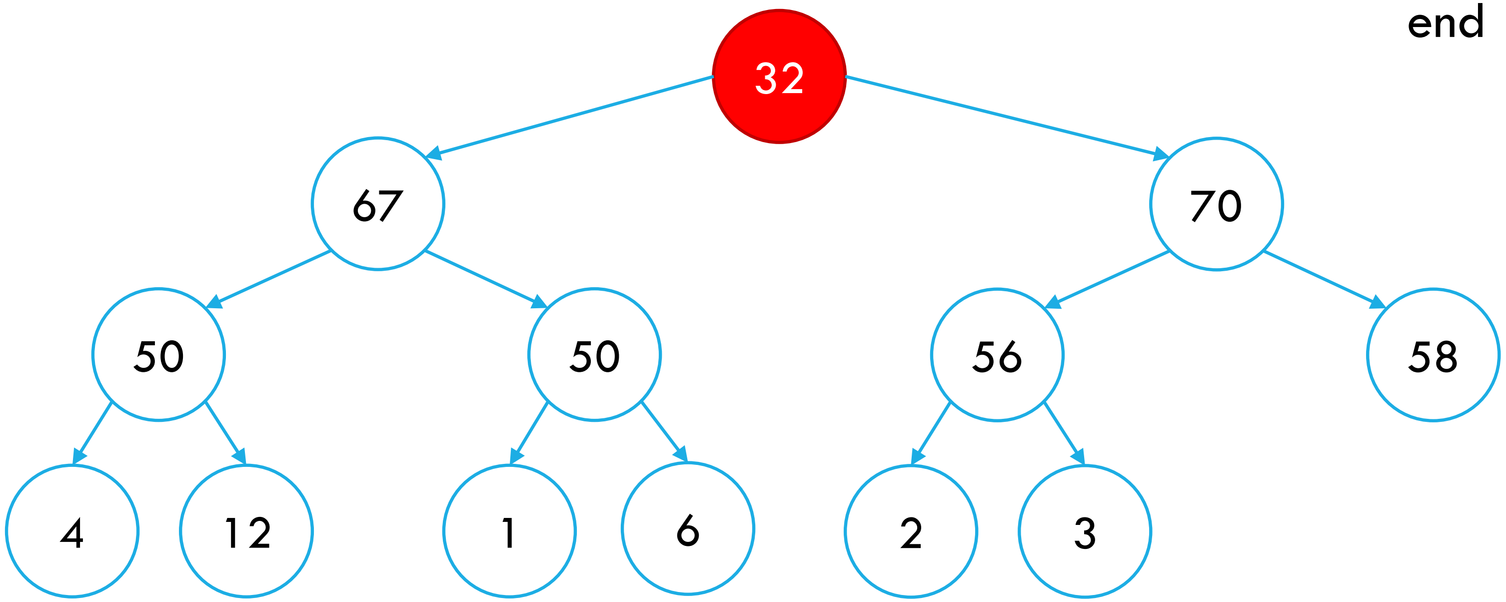


THE IDEA: SWAP WITH THE END, THEN SINK

Fix MaxHeap violation. How?

32	67	70	50	50	56	58	4	12	1	6	2	3	92

end

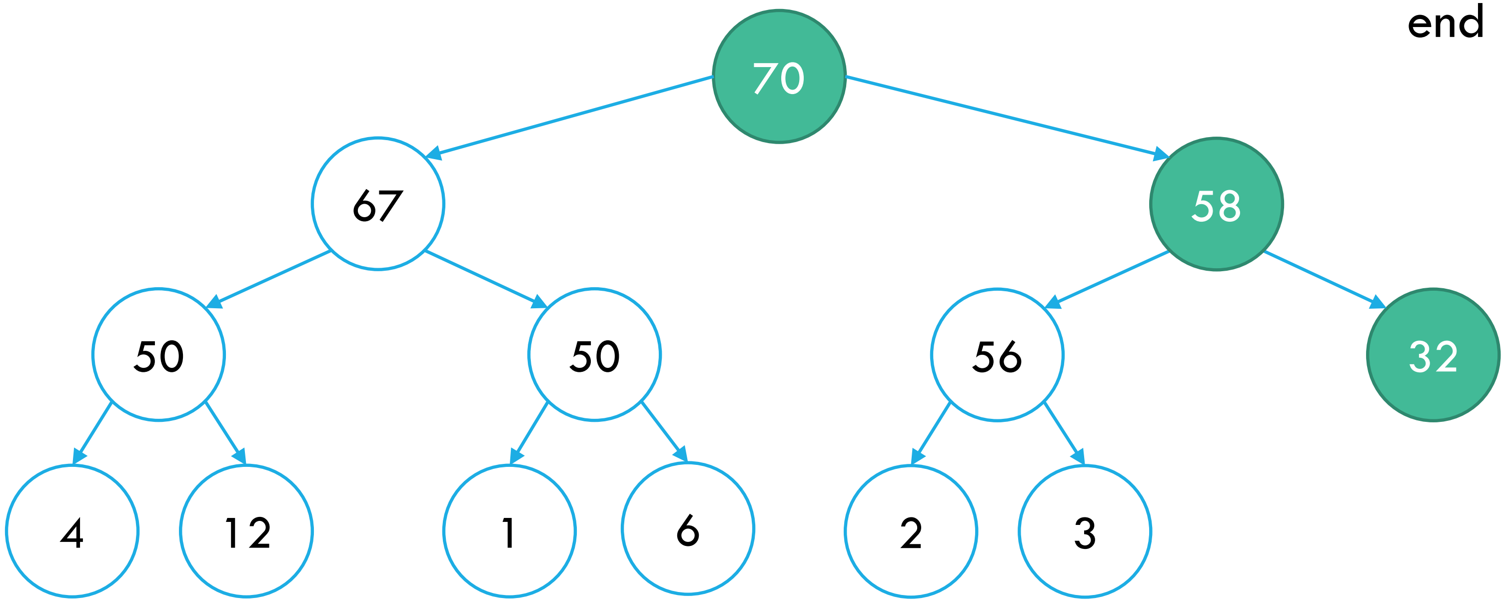


THE IDEA: SWAP WITH THE END, THEN SINK

Fix MaxHeap violation. Sink!

70	67	58	50	50	56	32	4	12	1	6	2	3	92

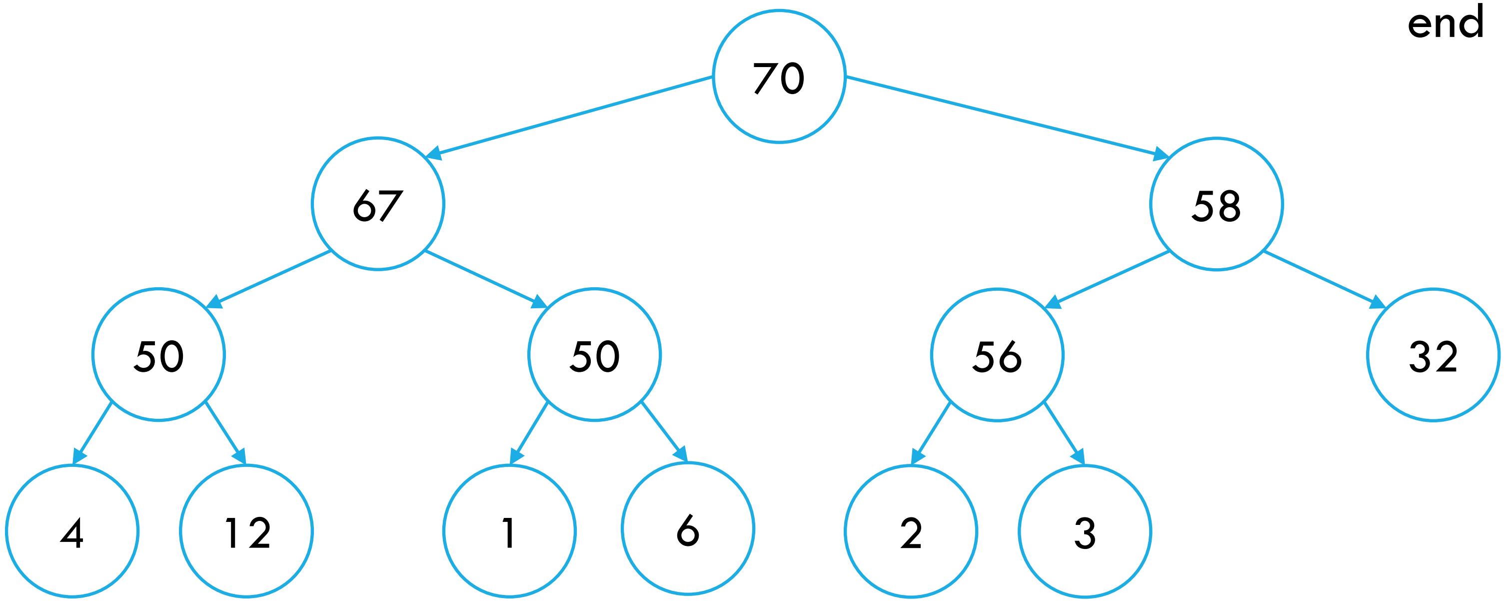
end



THE IDEA: SWAP WITH THE END, THEN SINK

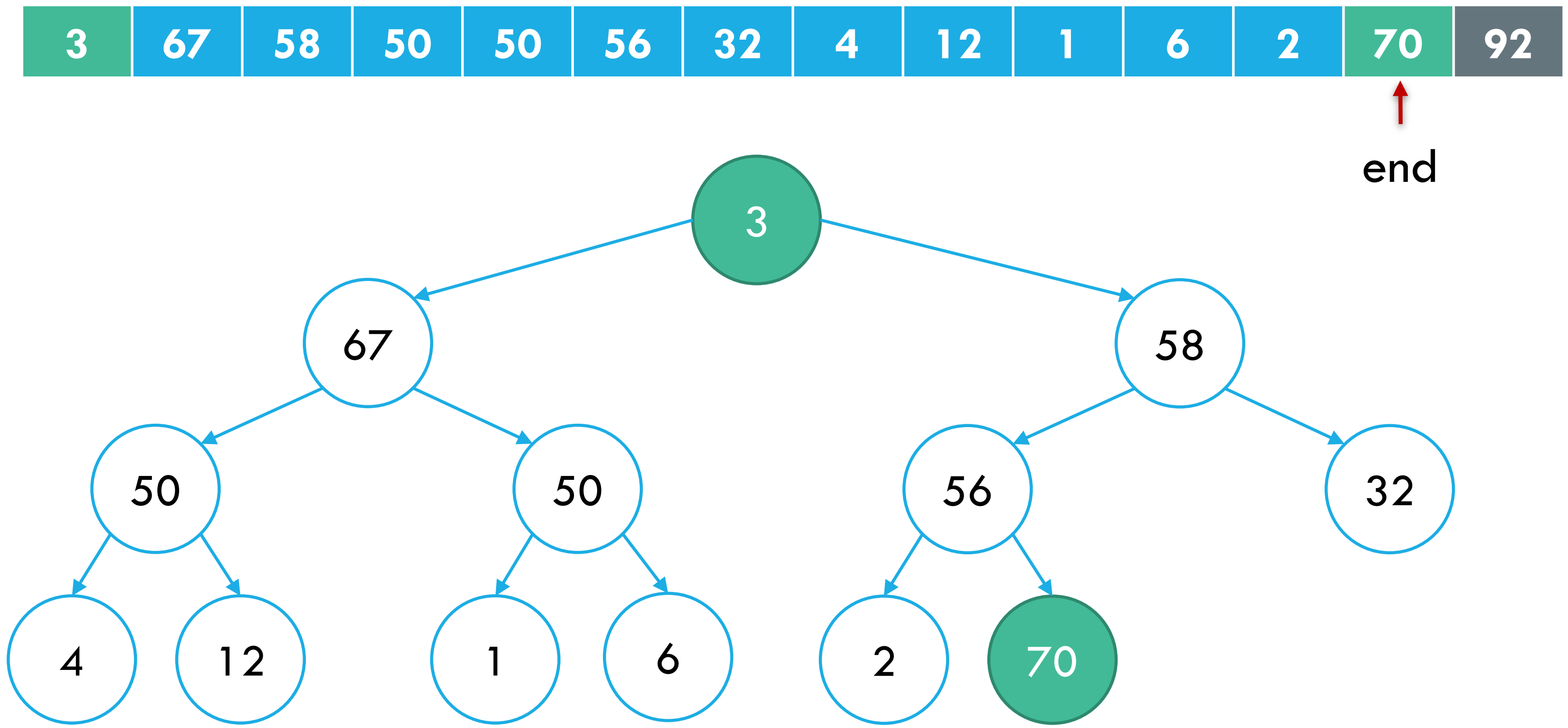
You now have 1 element in the right place. Repeat!

70	67	58	50	50	56	32	4	12	1	6	2	3	92
												end	



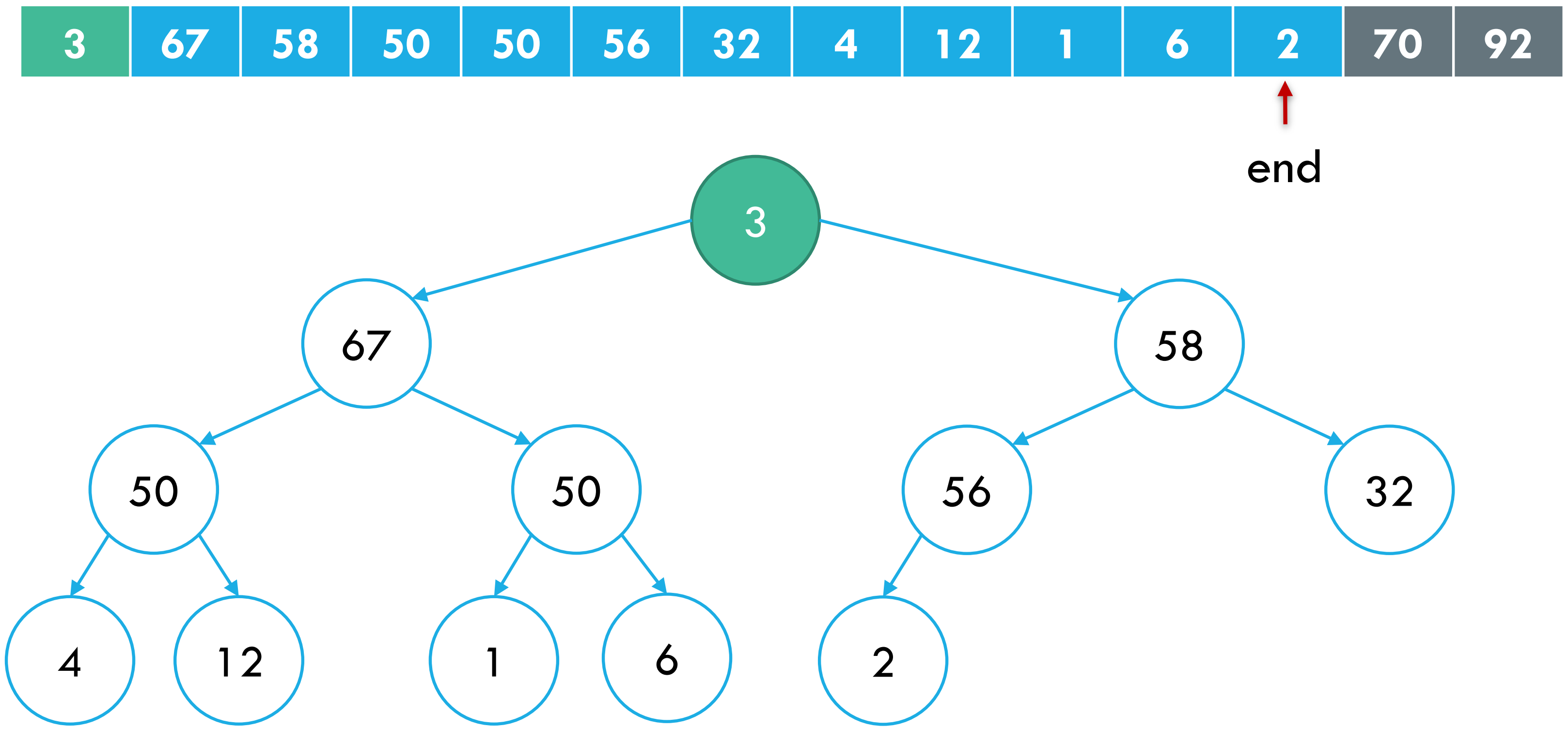
THE IDEA: SWAP WITH THE END, THEN SINK

Swap.



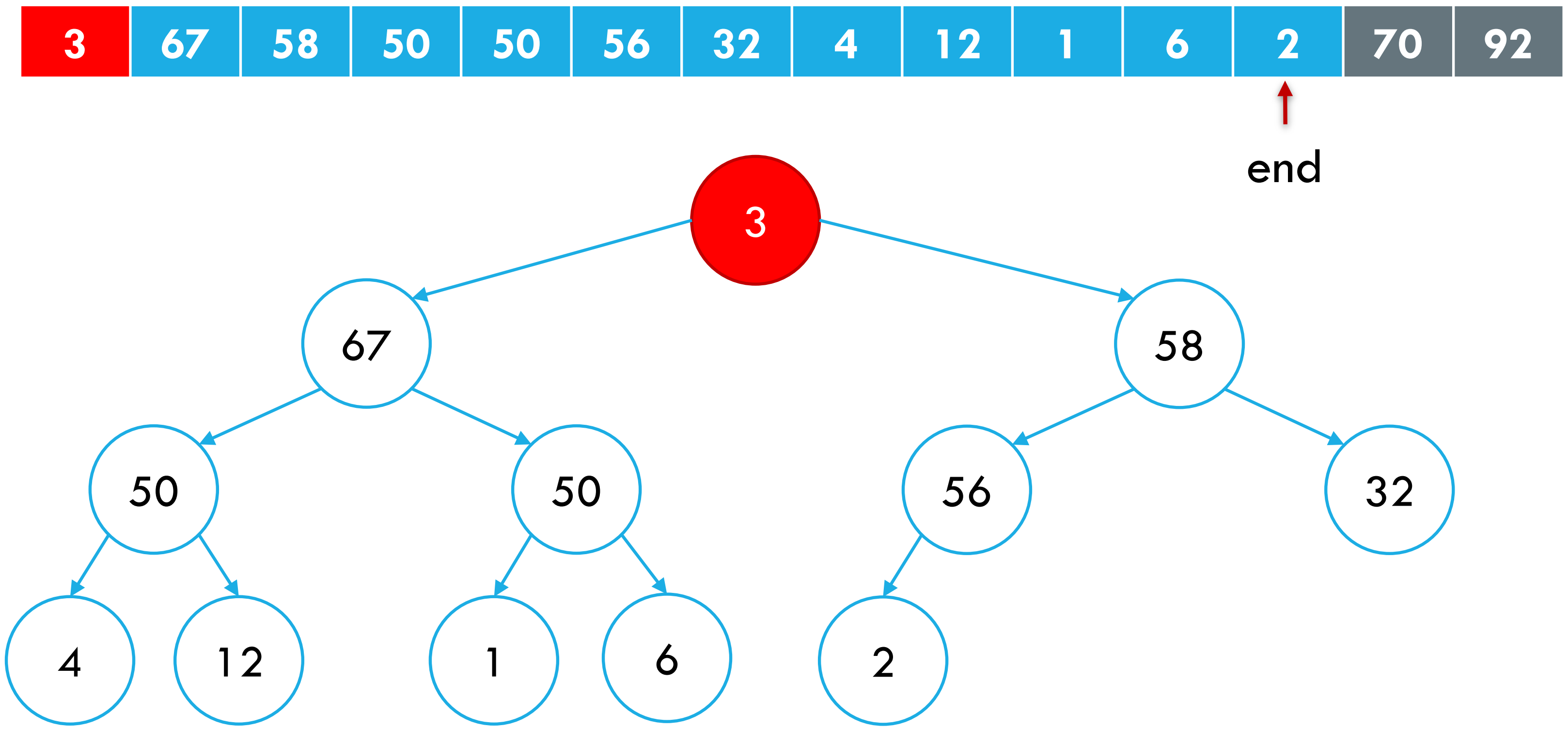
THE IDEA: SWAP WITH THE END, THEN SINK

Move end.



THE IDEA: SWAP WITH THE END, THEN SINK

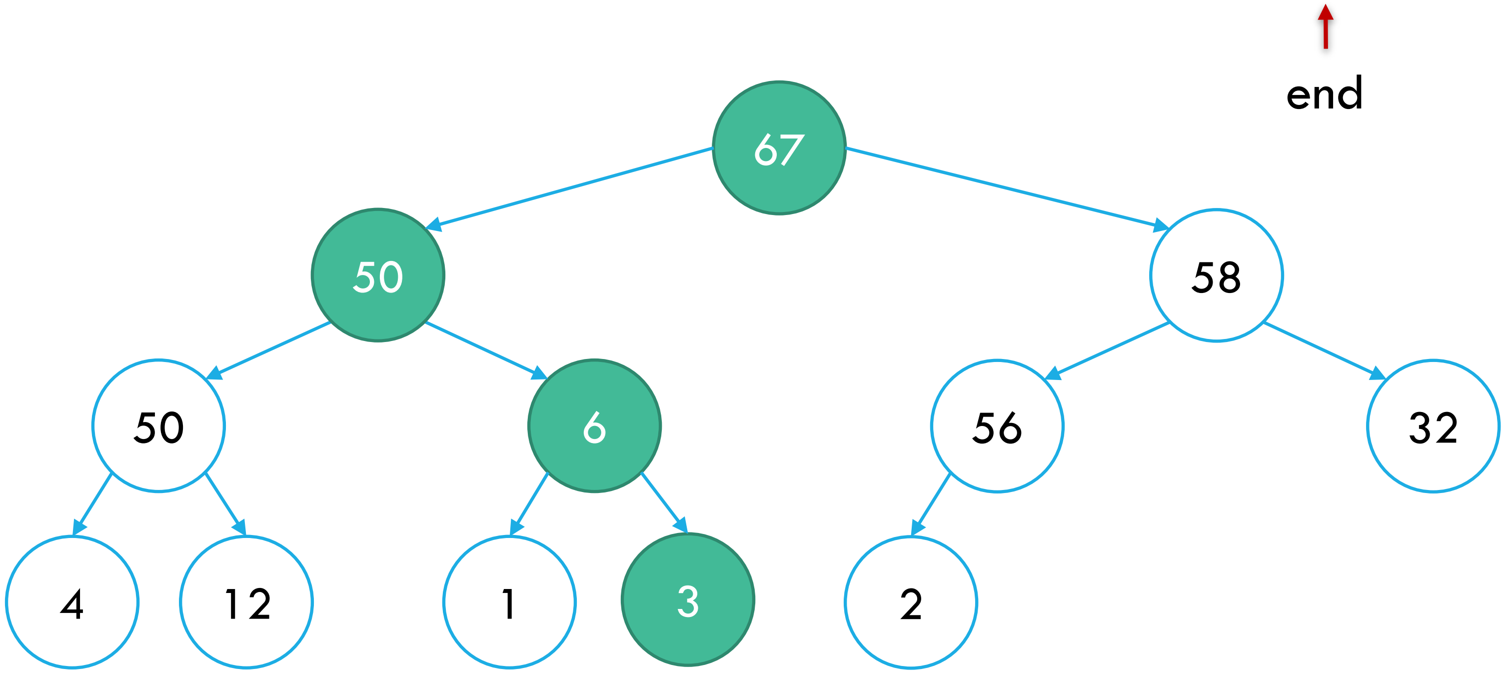
Sink.



THE IDEA: SWAP WITH THE END, THEN SINK

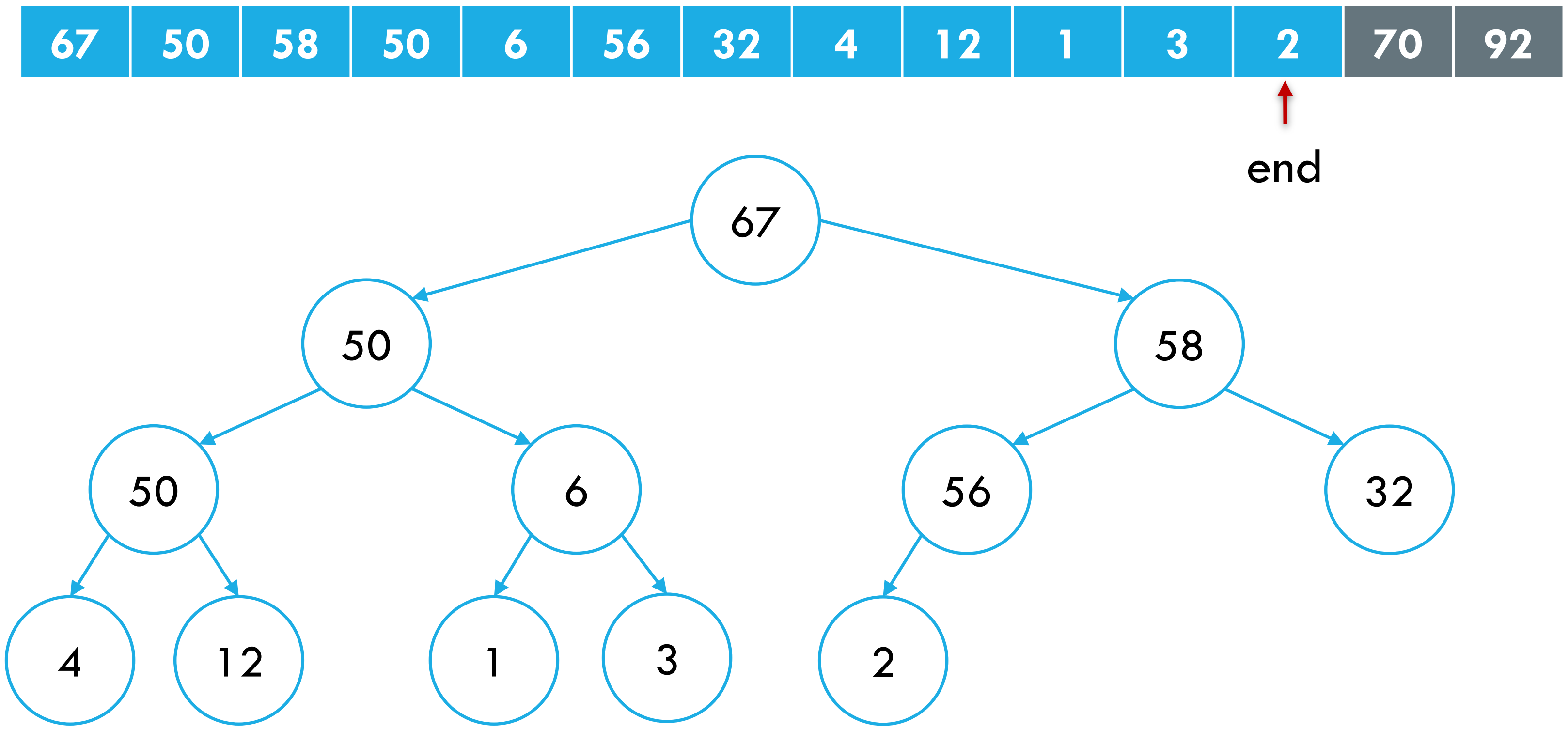
Sink.

67	50	58	50	6	56	32	4	12	1	3	2	70	92



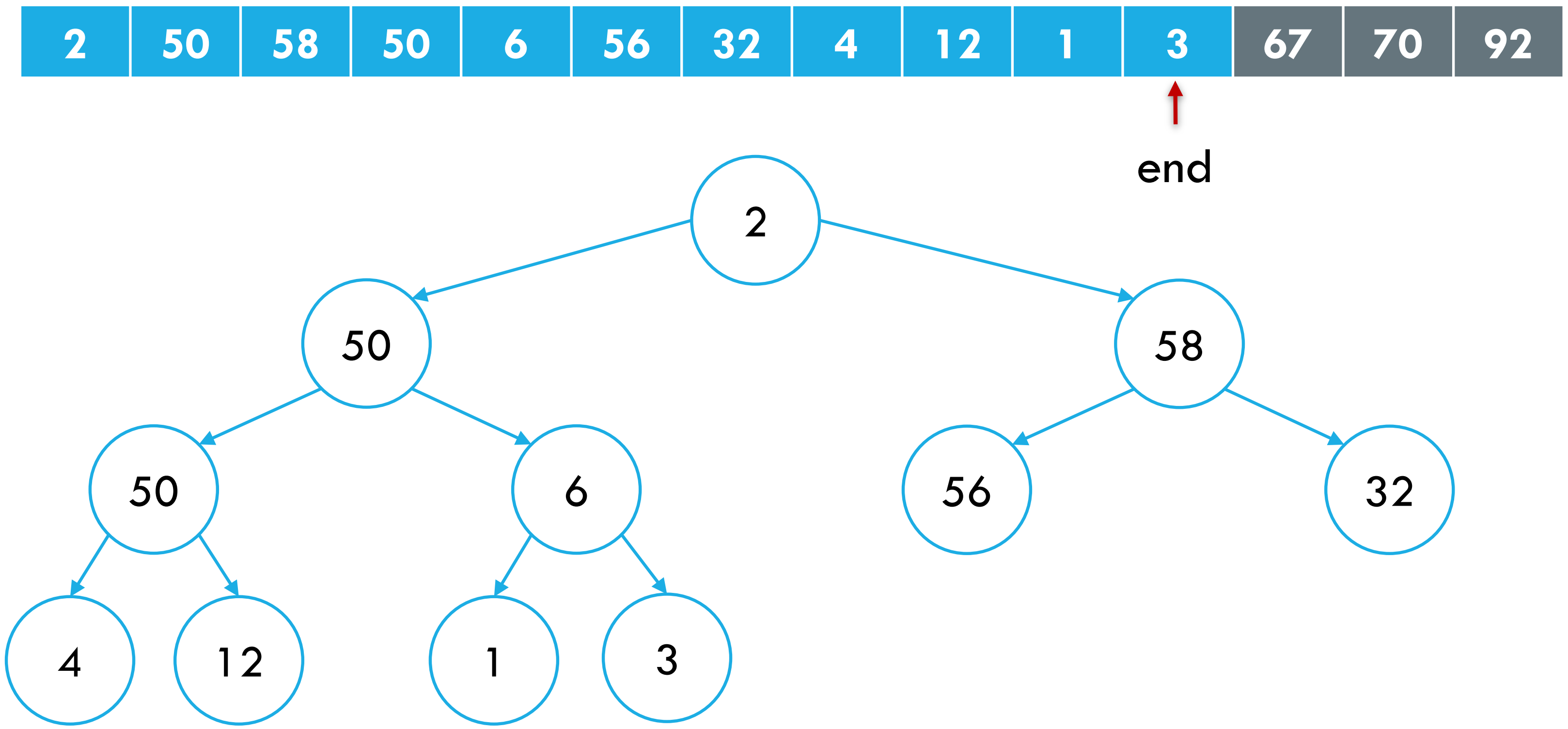
THE IDEA: SWAP WITH THE END, THEN SINK

2 elements in the right place! Repeat!



THE IDEA: SWAP WITH THE END, THEN SINK

Swap and Move.

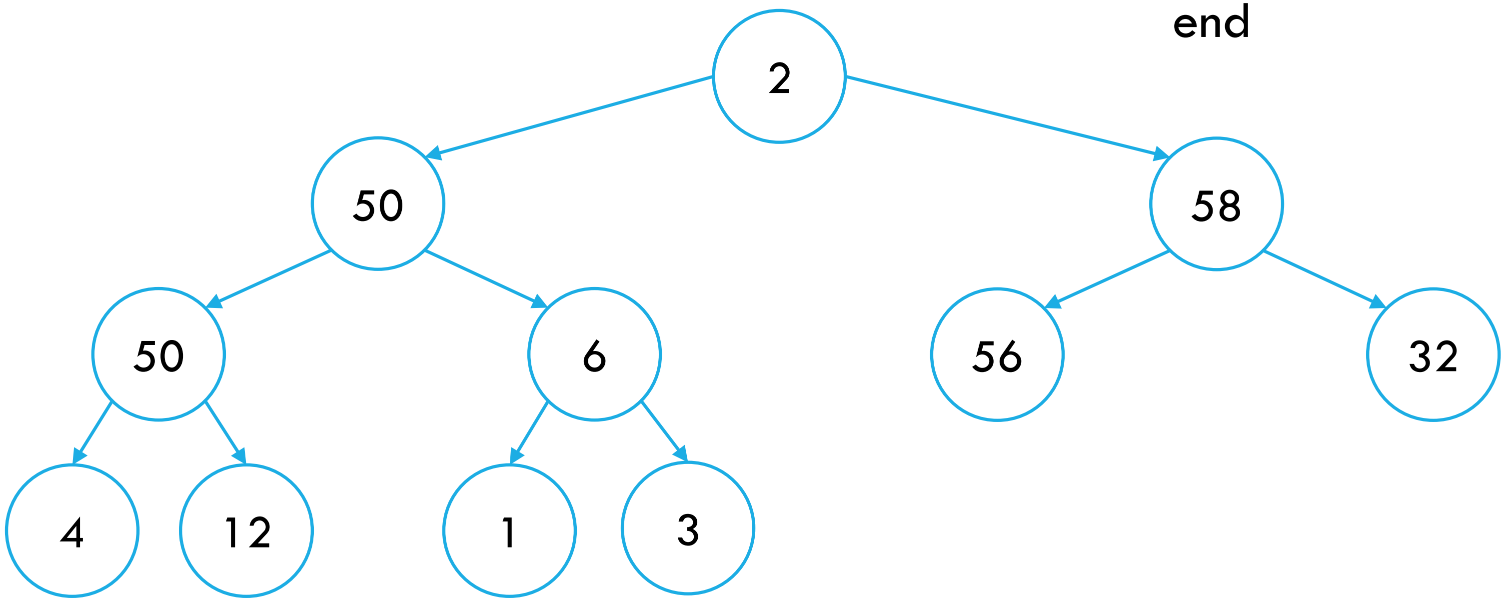


THE IDEA: SWAP WITH THE END, THEN SINK

Sink!

2	50	58	50	6	56	32	4	12	1	3	67	70	92

end

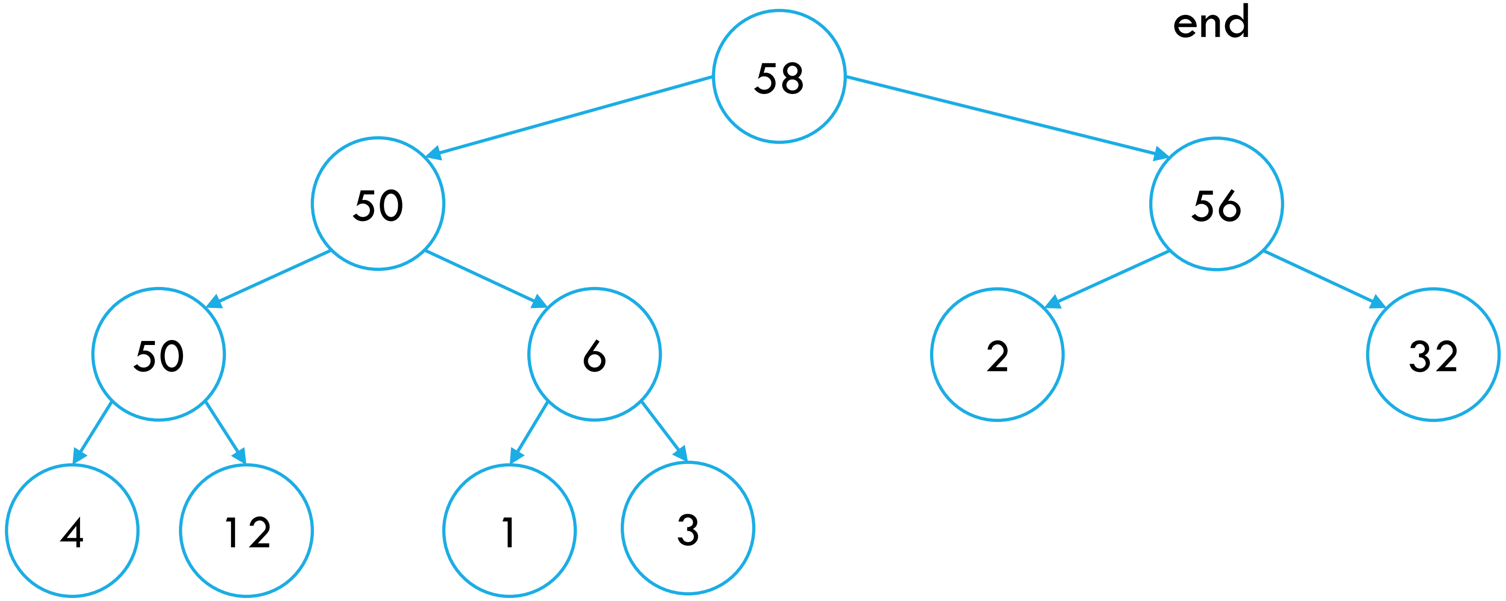


THE IDEA: SWAP WITH THE END, THEN SINK

Sink!

58	50	56	50	6	2	32	4	12	1	3	67	70	92

end

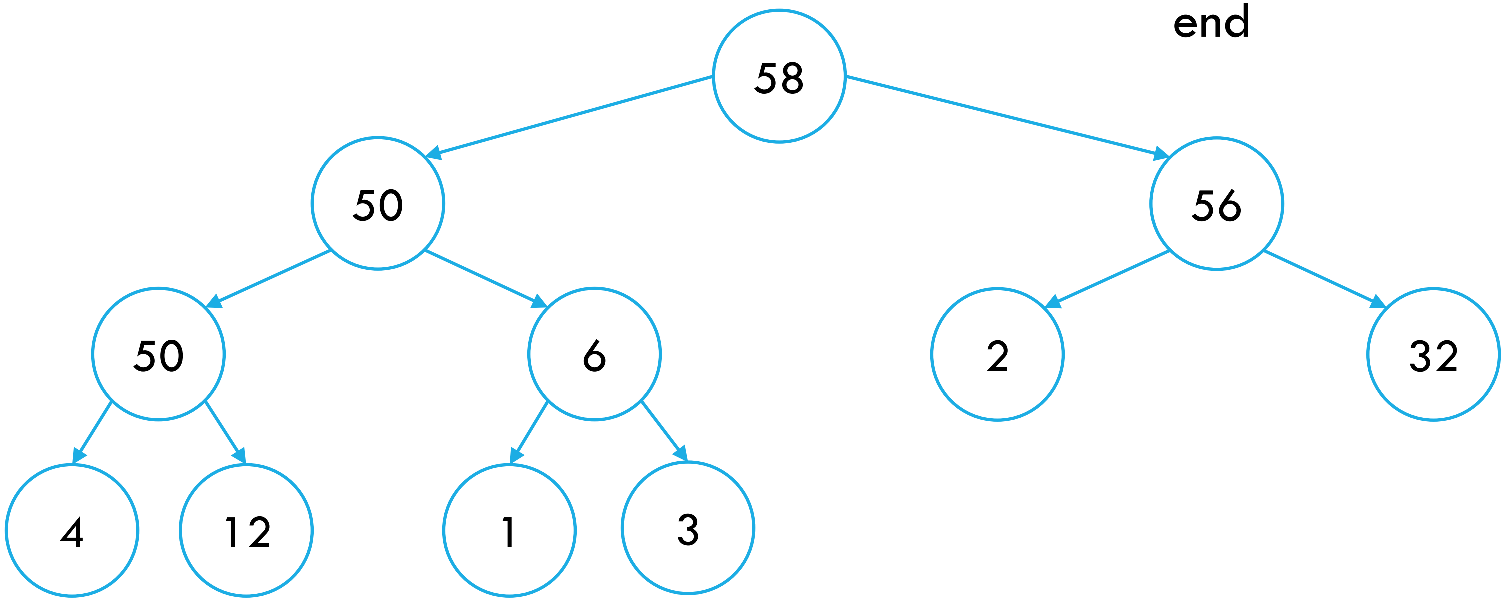


THE IDEA: SWAP WITH THE END, THEN SINK

3 elements in the right place! Repeat…

58	50	56	50	6	2	32	4	12	1	3	67	70	92

end



SOME TIME LATER...

All elements in the right place! Be Happy!

1	2	3	4	6	12	32	50	50	56	58	67	70	92

end

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

Creating a binary heap in place takes $O(n)$

Swapping and Sinking takes ___________

Note: uses modified sink and swim with array length

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

Creating a binary heap in place takes $O(n)$

Swapping and Sinking takes $O(n \log n)$

Note: uses modified sink and swim with array length

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

How much additional space does heapsort require?

A. $O(n)$
B. $O(n^2)$
C. $O(n \log n)$
D. $O(2^{\log n})$
E. $O(1)$

Note: uses modified sink and swim with array length

HEAPSORT CODE

```
function heapsort(A)
        n = A.length
        // create the binary heap
        for i = n/2 to 1
                sink(A, k, n)
        // swap and sink
        while (n > 1)
                swap(A, 1, n);
                sink(A, 1, --n);
```

How much additional space does heapsort require?

A. $O(n)$
B. $O(n^2)$
C. $O(n \log n)$
D. $O(2^{\log n})$
E. $\boldsymbol{O}(1)$

Note: uses modified sink and swim with array length

HEAPSORT COMPARED TO QUICKSORT

Heapsort is optimal for time and space but:

- inner loop is longer than quicksort.
- bad caching (children are “far” from parents).
- not stable.

Quicksort is still faster in practice.

INTROSORT: IMPROVING QUICKSORT!

Introsort:

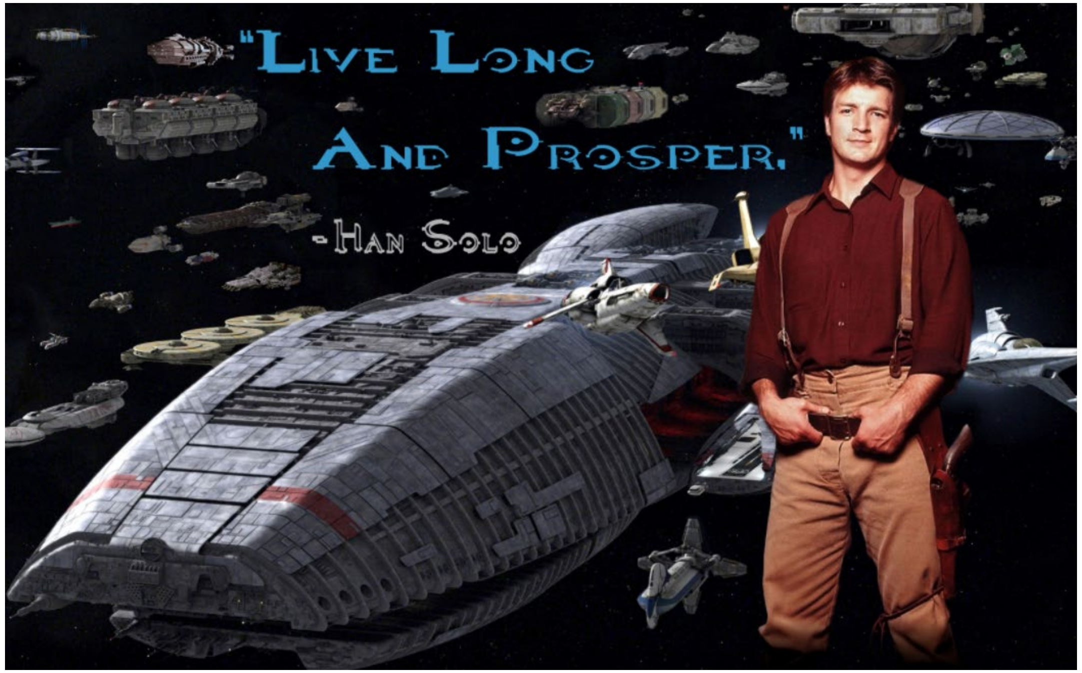
- As fast as quicksort in practice
- $O(n \log n)$ worst case!

Idea:

- Run quicksort
- Cutoff to heapsort if stack depth exceeds $2 \log n$
- Cutoff to insertion sort for $n = 16$

Used in C++ STL, Microsoft .NET

Also check out Timsort (mergesort + insertion sort)



Sorting algorithms: summary

	inplace?	stable?	best	average	worst	remarks
selection	✔		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges
insertion	✔	✔	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small n or partially ordered
shell	✔		$n \log_3 n$	?	$c\, n^{3/2}$	tight code; subquadratic
merge		✔	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
timsort		✔	n	$n \lg n$	$n \lg n$	improves mergesort when preexisting order
quick	✔		$n \lg n$	$2 n \ln n$	$\frac{1}{2} n^2$	$n \log n$ probabilistic guarantee; fastest in practice
3-way quick	✔		n	$2 n \ln n$	$\frac{1}{2} n^2$	improves quicksort when duplicate keys
heap	✔		$3 n$	$2 n \lg n$	$2 n \lg n$	$n \log n$ guarantee; in-place
?	✔	✔	n	$n \lg n$	$n \lg n$	holy sorting grail

From Sedgewick and Wayne's slides
http://algs4.cs.princeton.edu/lectures/24PriorityQueues.pdf

VARIATIONS: MIN HEAP

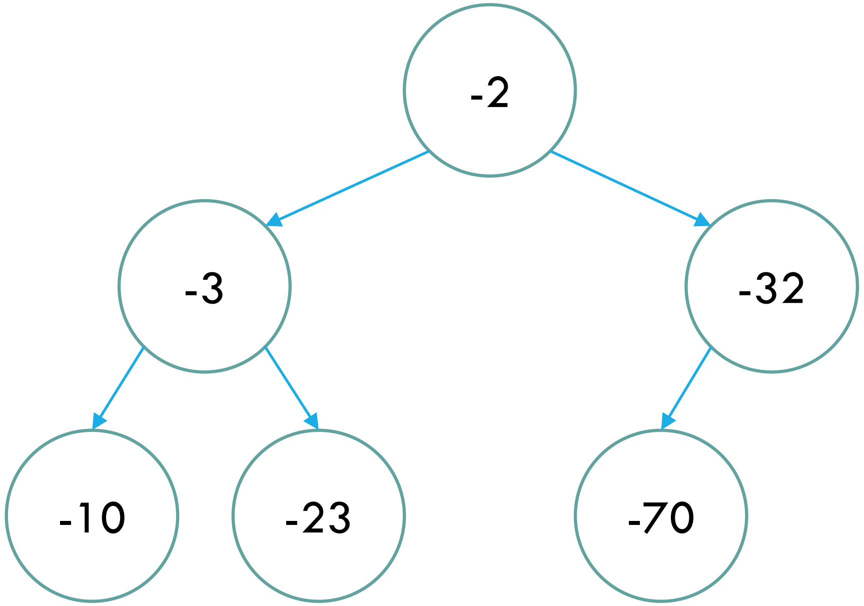
What if we want the minimum item?

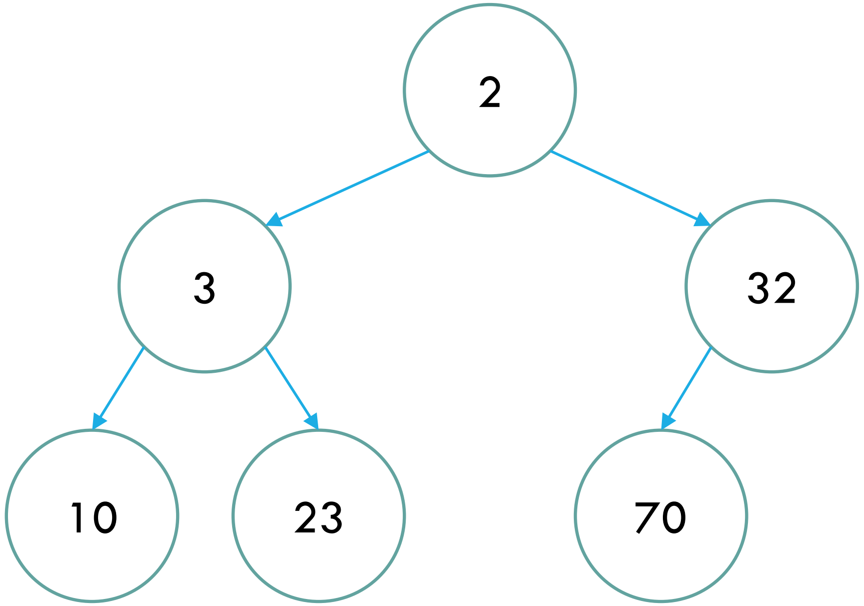
Easy hack:

- (provided only non-negative integer keys) negate all keys, e.g., 10 becomes -10

More Correct:

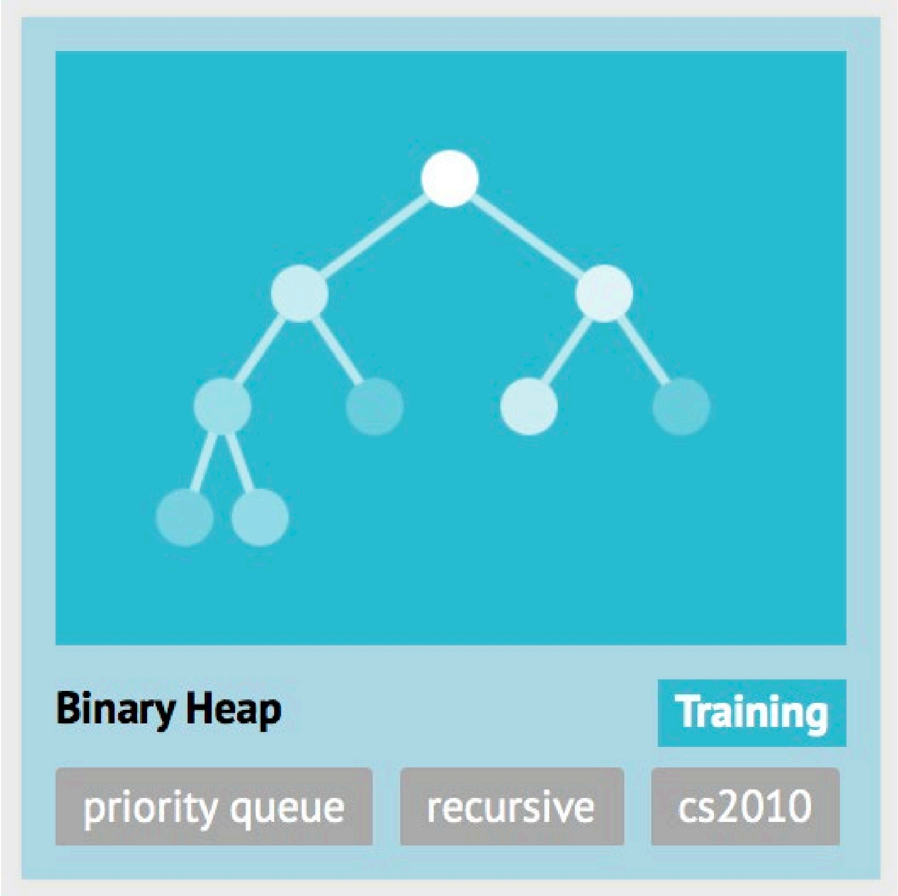
- Modify heap maintenance methods
  - This is a nice exercise (and not difficult)
- Or change the comparator





BINARY HEAP AND HEAPSORT

More info and visualizations on Visualgo.net!



QUESTIONS?

SUMMARY: LEARNING OUTCOMES

By the end of this session, students should be able to:

- Describe the **priority queue ADT** and its operations
- Describe the **binary heap data structure** and explain how it works
- Analyze the **performance of the binary heap data** structure
- Describe the **heapsort algorithm** and explain how it works
- Analyze the **performance of heapsort**

ADMINISTRATIVE ISSUES

Quiz 1 is tomorrow

- Wednesday (4th Sept) during Lecture
- Open-book quiz
- No magnifying glass.
- No electronic equipment allowed.

BEFORE LECTURE NEXT WEEK

Go to Visualgo.net and do the Binary Search Tree (BST)Module:

- https://visualgo.net/en/bst
- Review: 1-14 (AVL Tree)
- Optional: 15