CS 1332R
WEEK 6

Binary Heaps

Priority Queues

ANNOUNCEMENTS

-

Heaps

SHAPE Property

- Complete, binary trees
  - *complete*: all levels are filled, last level filled left to right
  - *binary*: nodes have a maximum of two children

ORDER Property

- MinHeap: parent data is smaller than *both* of its children data
- **MaxHeap (your homework)**: parent data is larger than *both* of its children data
- *NOTE: siblings do not have a relational order property*

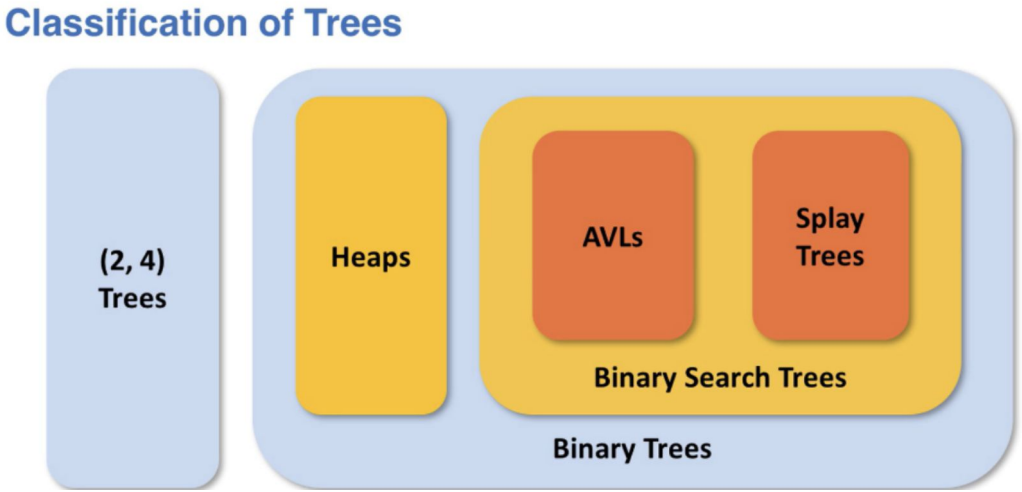
When we modify a heap, we:

1. ***Preserve the shape property.***
2. ***Fix the order property.***

Heaps

How is a heap different than a binary search tree?

How are they the same?



(1332) Heap ADT

A constructor instantiating an empty heap

A constructor instantiating a heap with an ArrayList of data

```
void add(T data)

T remove()

T getMax/Min()

int size()

boolean isEmpty()

void clear()
```

Heap: Implementation

Backed by an array

Why are we backing a tree by an array rather than nodes?

- ➔ complete tree property allows us to store the data contiguously
- ➔ less memory
- ➔ iterative & recursive algorithm implementation
- ➔ allows build-heap algorithm to run efficiently

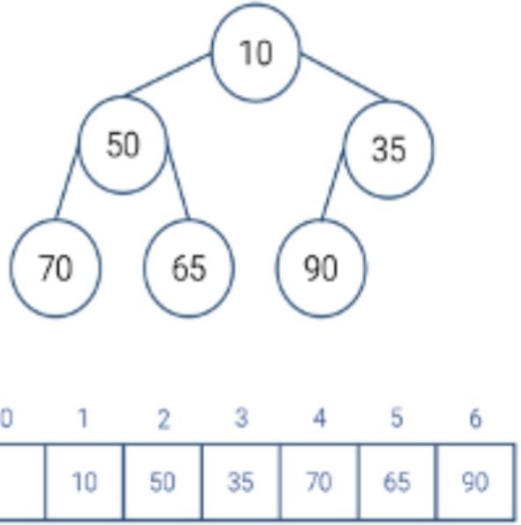
How do we preserve parent/child relationships without nodes?

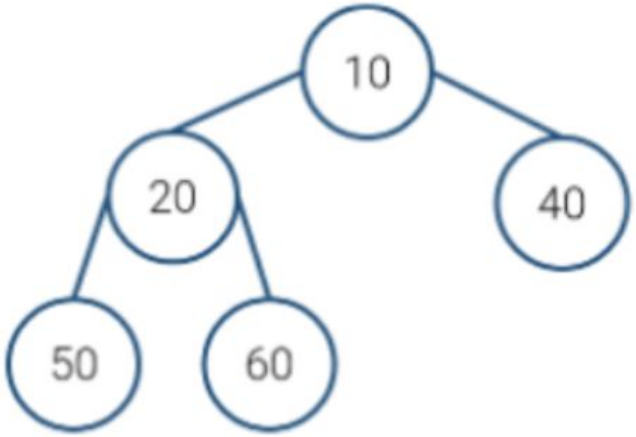
➔ Answer: math! If we **start putting data at index 1…**

- `parent = i / 2 (where i is the index of the child)`
- `left child = 2 * i (where i is the index of the parent)`
- `right child = 2 * i + 1 (where i is the index of the parent)`

Heap: Implementation

- parent = i / 2 (where i is the index of the child)
- left child = 2 * i (where i is the index of the parent)
- right child = 2 * i + 1 (where i is the index of the parent)



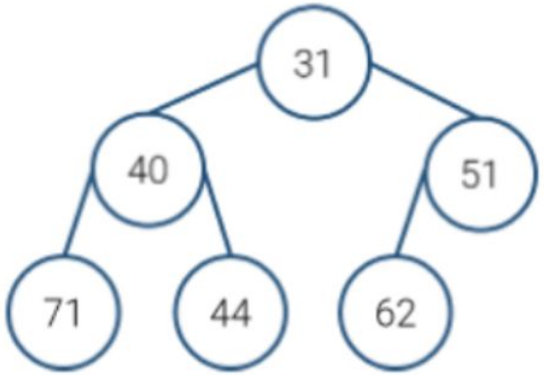


Tree

Complete Tree

MinHeap

MaxHeap

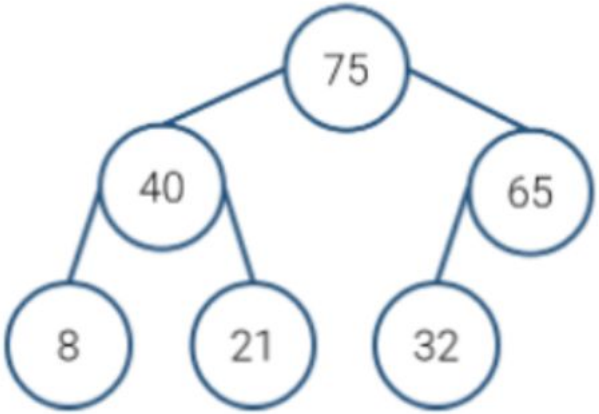


Tree

Complete Tree

MinHeap

MaxHeap

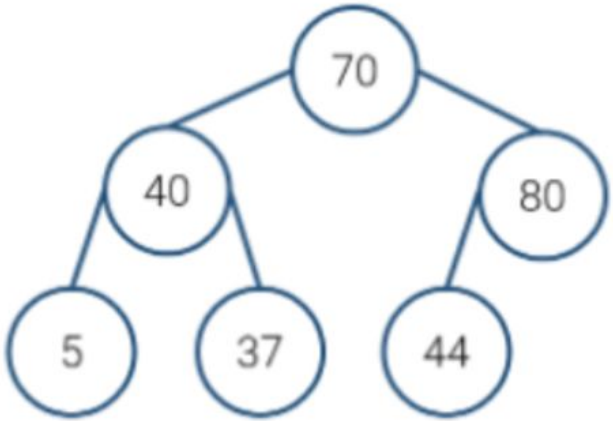


Tree

Complete Tree

MinHeap

MaxHeap

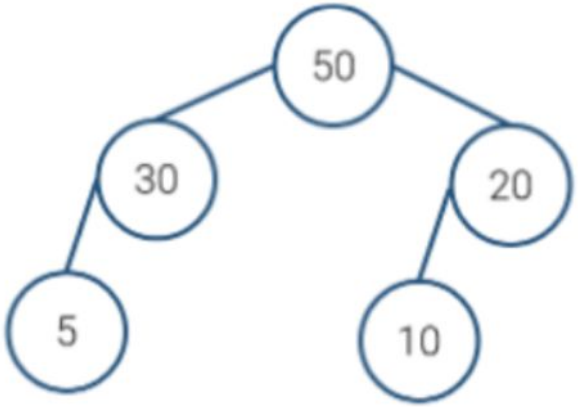


Tree

Complete Tree

MinHeap

MaxHeap



Tree

Complete Tree

MinHeap

MaxHeap

Heap: Access

- ➔ Always accessing the highest priority element, which is always at index 1
- ➔ Heaps are not used to search or iterate

```
return backingArray[1]
```

O(1) access to highest priority element.

Heap: Add

When we modify a heap, we:

1. ***Preserve the shape property.***
2. ***Fix the order property.***

(Check for resize case.)

1. **Preserve Shape:** Add our data at the end of the last level.

What index do we add at?

`size`

2. **Fix Order:** Perform upheap.

When we modify a heap, we:

1. *Preserve the shape property.*
2. ***Fix the order property.***

Heap: Add → Upheap

➔ "Bubble" up data to the correct spot
➔ Can be done recursively *or* iteratively

PROCEDURE

1. **Compare child data with its parent**.
2. If child data is of a higher priority than its parent, swap.
3. Repeat until:
   a. Order property is met
   b. Root is reached

csvistool.com

Heap: Remove

When we modify a heap, we:

1. ***Preserve the shape property.***
2. ***Fix the order property.***

1. **Preserve Shape:** Save the root data (`backingArray[1]`), replace it with the data in the last leaf (`backingArray[size]`). Remove the last leaf.
2. **Fix Order:** Perform downHeap.

When we modify a heap, we:

1. *Preserve the shape property.*
2. ***Fix the order property.***

Heap: Remove → DownHeap

➔ "Bubbling" data down to its correct spot
➔ Can be done recursively *or* iteratively

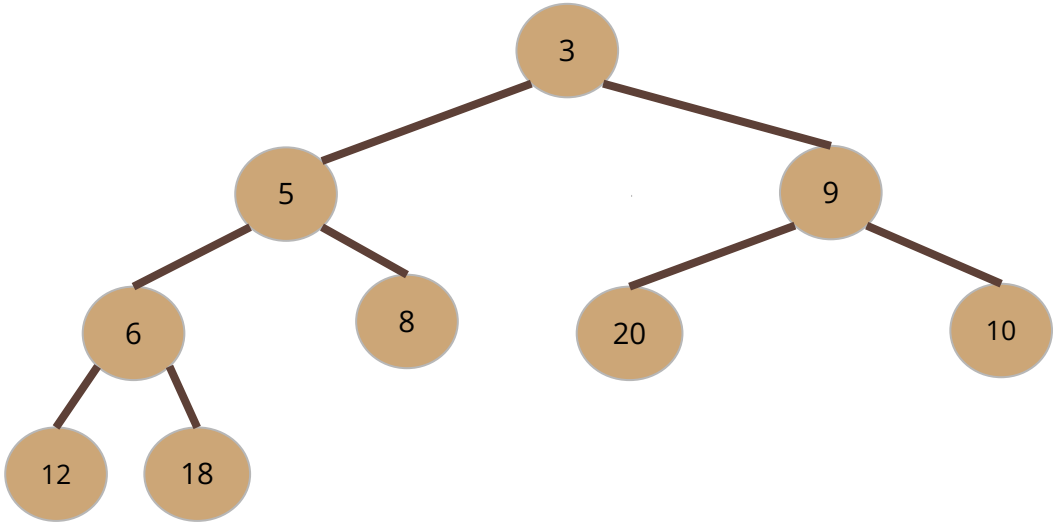
PROCEDURE

1. **Compare parent with its two children**.
2. If data is of a lower priority than one or both of its children, swap with the highest priority child.
3. Repeat until
   a. order property is met
   b. last parent node is reached

csvistool.com

Heap: Practice

Perform the following operations on this MinHeap.



add(40)

add(90)

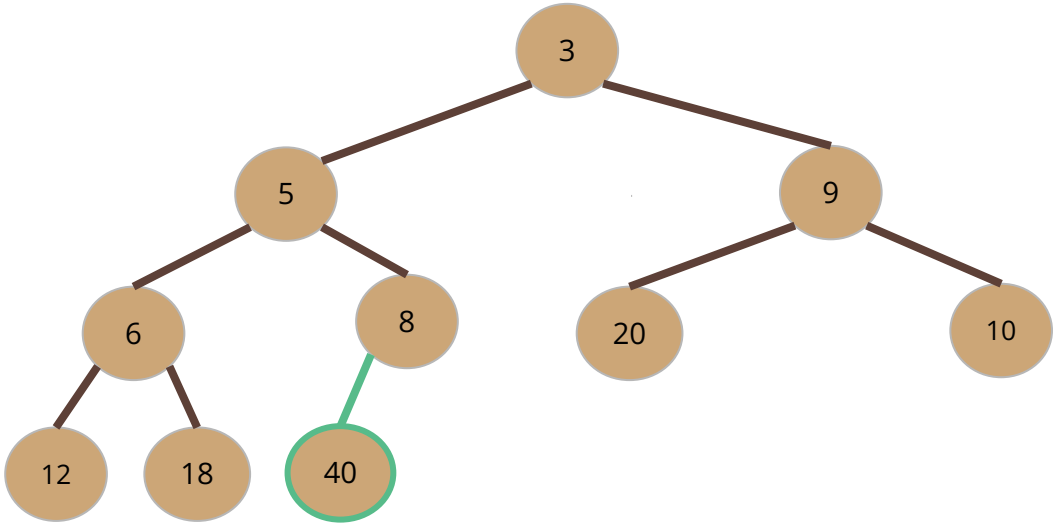
remove()

add(14)

remove()

Heap: Practice

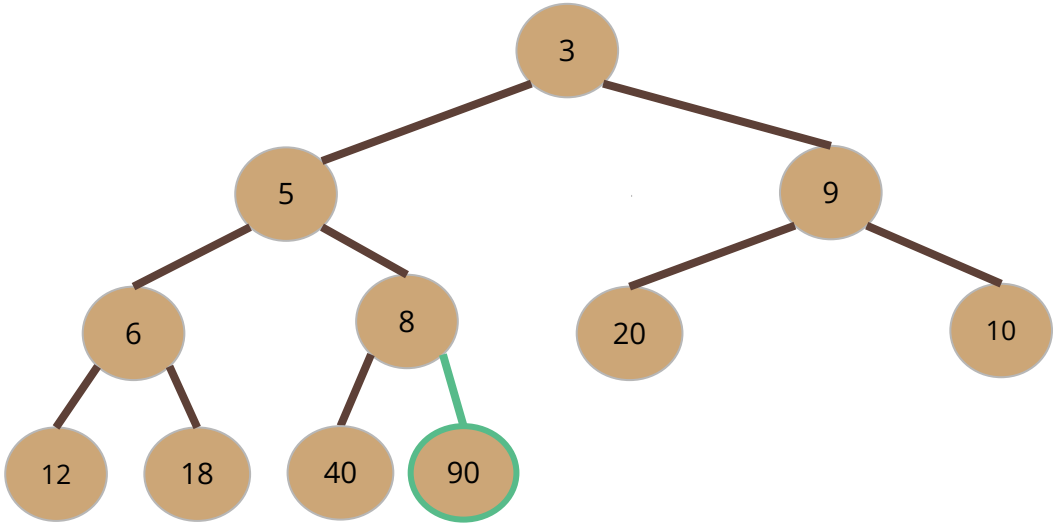
Perform the following operations on this MinHeap.



add(40)

Heap: Practice

Perform the following operations on this MinHeap.

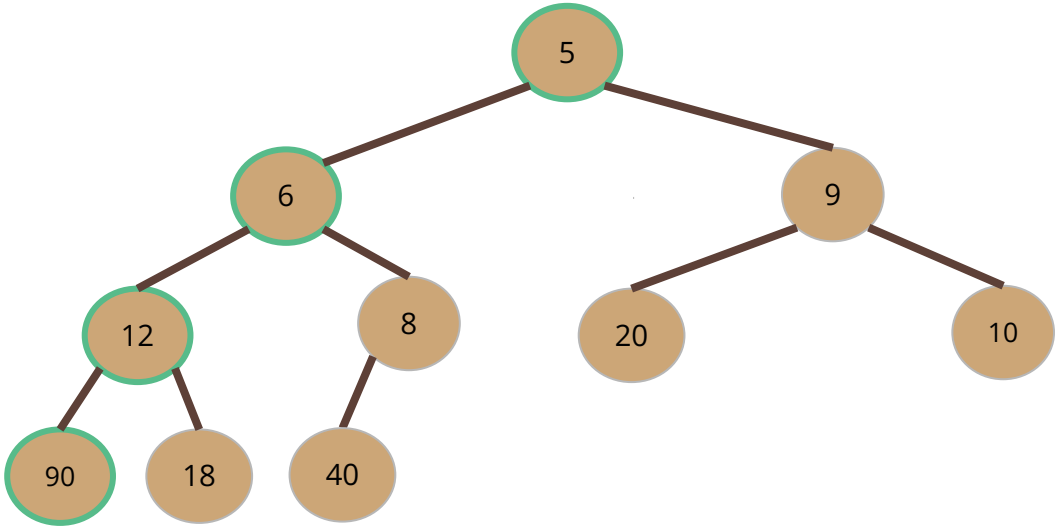


add(40)

add(90)

Heap: Practice

Perform the following operations on this MinHeap.



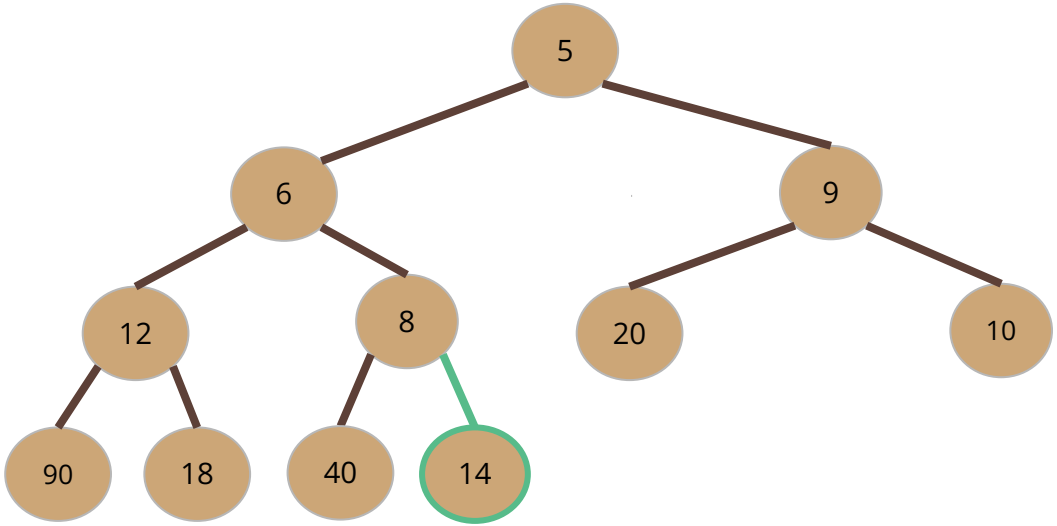
add(40)

add(90)

remove()

Heap: Practice

Perform the following operations on this MinHeap.



add(40)

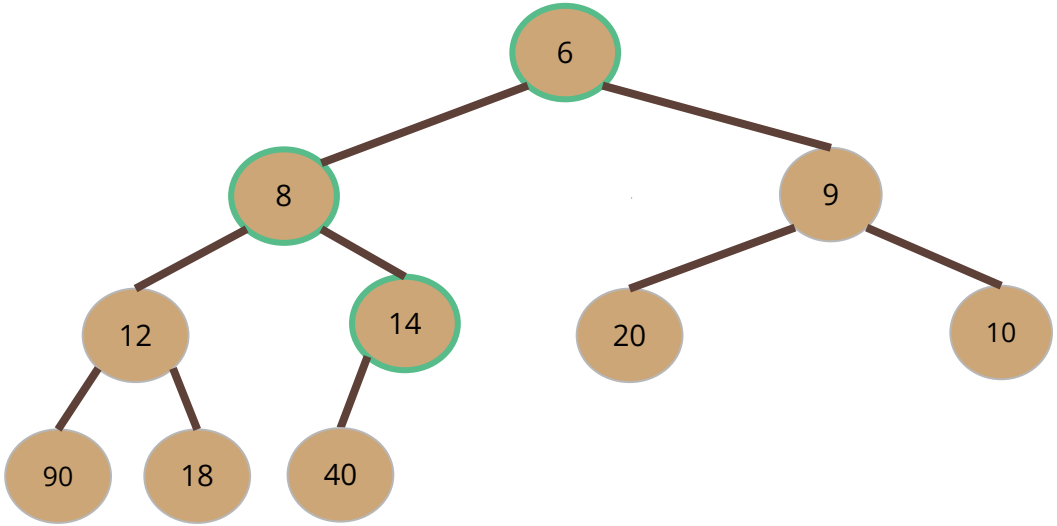
add(90)

remove()

add(14)

Heap: Practice

Perform the following operations on this MinHeap.



add(40)

add(90)

remove()

add(14)

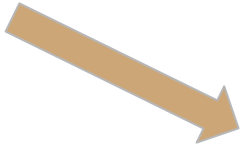
remove()

Heap ADT

A constructor instantiating an empty heap

A constructor instantiating a heap with an ArrayList of data

```
void add(T data)

T remove()

T getMax/Min()

int size()

boolean isEmpty()

void clear()
```



How did we do this in a BST?

We added every element, which was **O(nlogn)**.

How can we do this in a heap?

buildHeap - O(n)

Heap: BuildHeap

- Used to create a heap from a collection of data
  - Alternative: Add each piece of data, which is O(nlogn).
- BuildHeap() is O(n), which we will explain later, so it is necessary to use this algorithm

PROCEDURE

1. Copy over all data in the list to the backing array.
2. Iterate from size / 2 to index 1, perform the downHeap algorithm at every index.

Why do we have to iterate backwards, going up the heap?

DownHeap only works over an already valid heap.

Why do we start at index size / 2?

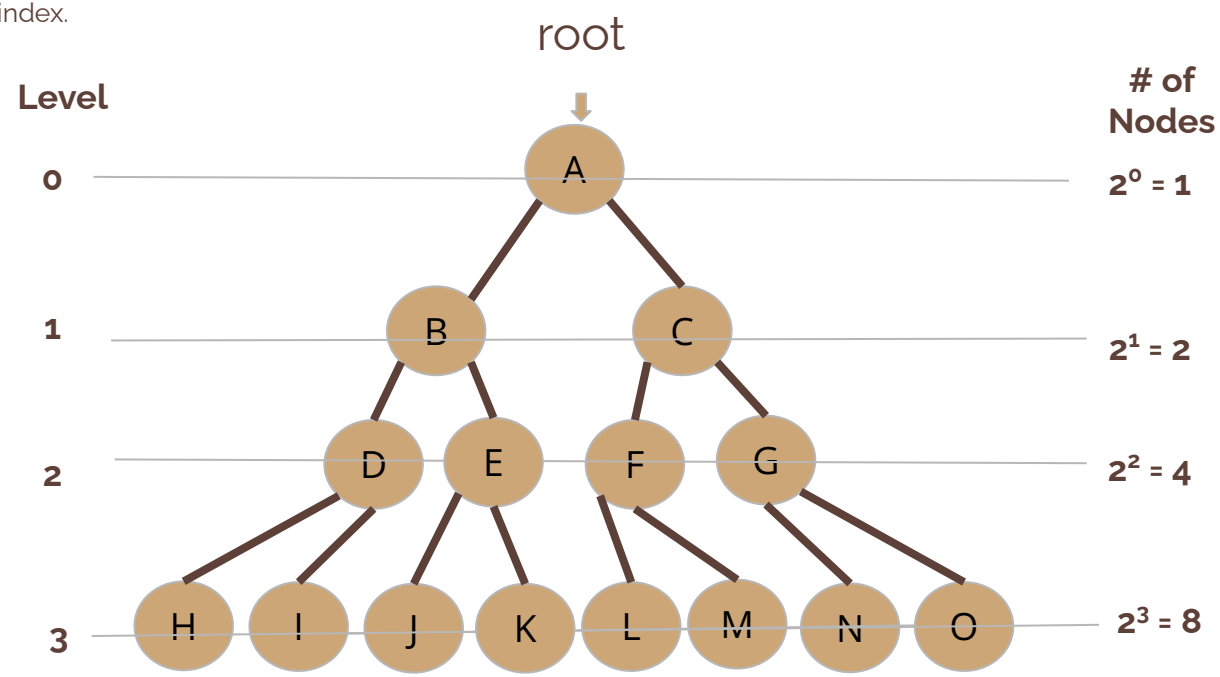
This is the index of the last parent.

PROCEDURE

1. Copy over all data in the list to the backing array.
2. Iterate from size / 2 to index 1, perform the downHeap algorithm at every index.

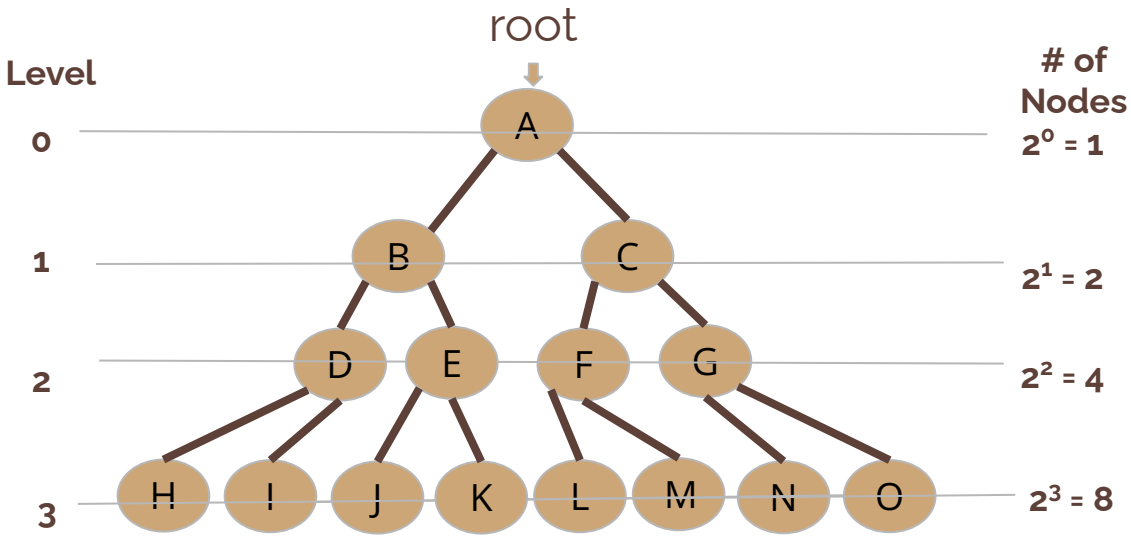
Heap: BuildHeap

How is buildHeap O(n)?



Heap: Efficiencies

Adding	Removing	Accessing	Up-heap	Down-heap
*O(logn)**	*O(logn)*	*O(1)*	*O(logn)*	*O(logn)*



Heap: Efficiency Questions

Adding	Removing	Accessing	Up-heap	Down-heap
*O(logn)**	*O(logn)*	*O(1)*	*O(logn)*	*O(logn)*

1. What is the runtime of accessing the smallest element of a MinHeap? **O(1)**
2. What is the runtime of getting the 5 largest elements of a MaxHeap? **O(logn)**
3. What is the runtime of converting a MinHeap to a MaxHeap? **O(nlogn)**

Priority Queue

- ➔ A queue that always dequeues the minimum or maximum element - ***not FIFO***
- ➔ Elements can be enqueued with a priority function
- ➔ Implemented by a Heap for best efficiency
- ➔ ***Java's Priority Queue is a MinHeap****
  - ◆ **Pass in a Comparator to the constructor to define the priority as you wish**

```
void enqueue(T data)

T dequeue()
```

LEETCODE PROBLEMS

215. Kth Largest Element in an Array (MEDIUM)

373. Find K Pairs with Smallest Sum (MEDIUM)

295. Find Median From Data Stream (HARD)

Any questions?

Name
Office Hours
Contact

Name
Office Hours
Contact

Let us know if there is anything specific you want out of recitation!