HEAPS

David Kauchak
CS 62 – Spring 2021

1

Admin

Pre-pre enrollment

OnDiskSort

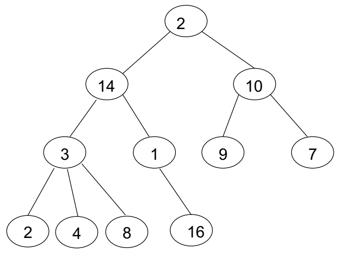
Lab tomorrow

2

Trees

A set of nodes based on a parent-child relationship

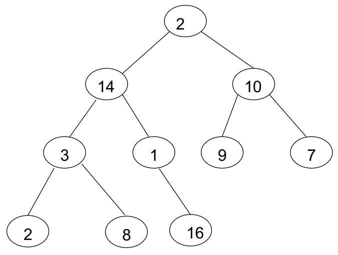
- Each node has one parent
- Root has no parent



3

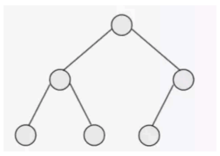
Binary tree

Each parent has at most 2 children

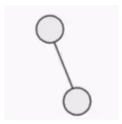


4

Full + Complete?



12

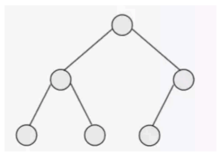


Full tree: a binary tree where every node has 0 or 2 children

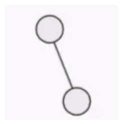
Complete: All levels except the last are completely filled and all nodes on the last level are on the left

5

Full + Complete?



12



Complete

Full + Complete

Neither

6

Implementing a binary tree

```
public class BinaryTree<E> {
    private Node root;

    private class Node {
        private E value;
        private Node left;
        private Node right;

        public Node(Node left, Node right, E value) {
            this.left = left;
            this.right = right;
            this.value = value;
        }
    }
```

7

Recursive data structure!

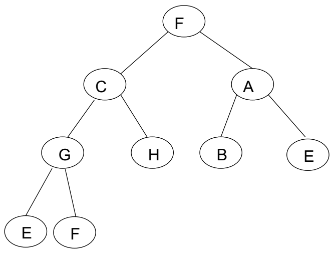
```
public class BinaryTree<E> {
    private Node root;

    private class Node {
        private E value;
        private Node left;
        private Node right;

        public Node(Node left, Node right, E value) {
            this.left = left;
            this.right = right;
            this.value = value;
        }
    }
```

```
public class LinkedList<E> {
    private Node head;

    private class Node {
        private E value;
        private Node next;

        public Node(Node next, E value) {
            this.next = next;
            this.value = value;
        }
    }
```

8

Tree traversals

```
Inorder(node):
    Inorder(left)
    visit node (e.g., print)
    Inorder(right)
```

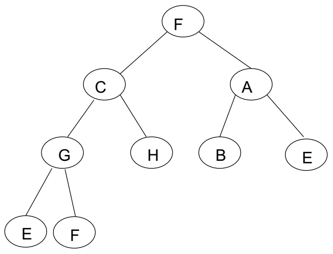


What would be printed out?

9

Tree traversals

```
Inorder(node):
    Inorder(left)
    visit node (e.g., print)
    Inorder(right)
```

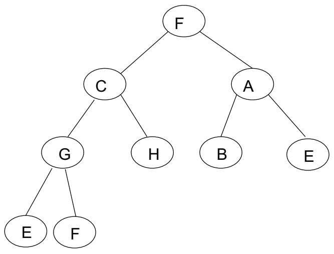


E G F C H F B A E

10

Tree traversals

```
Postorder(node):
    Postorder(left)
    Postorder(right)
    visit node (e.g., print)
```

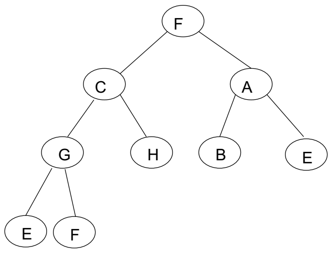


What would be printed out?

11

Tree traversals

```
Postorder(node):
    Postorder(left)
    Postorder(right)
    visit node (e.g., print)
```

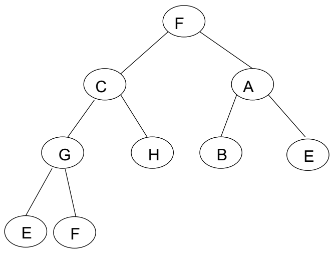


E F G H C B E A F

12

Tree traversals

```
Preorder(node):
    visit node (e.g., print)
    Preorder(left)
    Preorder(right)
```

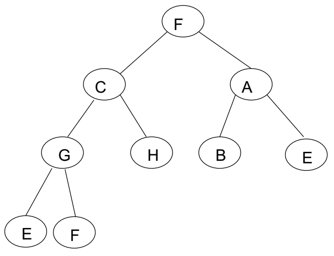


What would be printed out?

13

Tree traversals

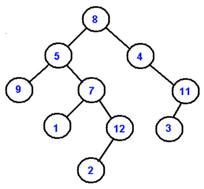
```
Preorder(node):
    visit node (e.g., print)
    Preorder(left)
    Preorder(right)
```



F C G E F H A B E

14

in-order, post-order, pre-order



```
Inorder(node):
    Inorder(left)
    visit node (e.g., print)
    Inorder(right)
```

```
Postorder(node):
    Postorder(left)
    Postorder(right)
    visit node (e.g., print)
```

```
Preorder(node):
    visit node (e.g., print)
    Preorder(left)
    Preorder(right)
```

15

Binary heap

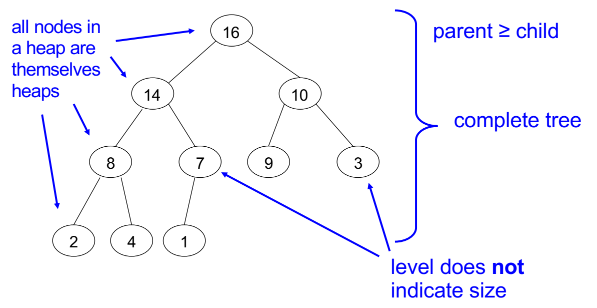
A binary tree where the value of a parent is greater than or equal to the value of its children

Additional restriction: the tree must be **complete**!

Max heap vs. min heap

16

Binary heap - references


all nodes in a heap are themselves heaps
16
14
10
8
7
9
3
2
4
1
parent ≥ child
complete tree
level does **not** indicate size

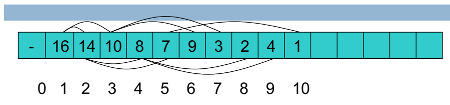
17

Binary heap - array

```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

18

Binary heap - array

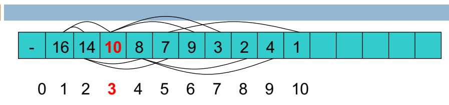


Note 0 is empty!

```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

19

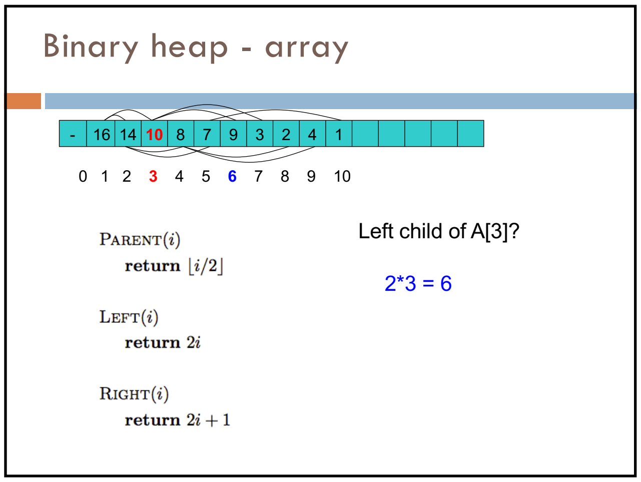
Binary heap - array



```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

Left child of A[3]?

20



Binary heap - array

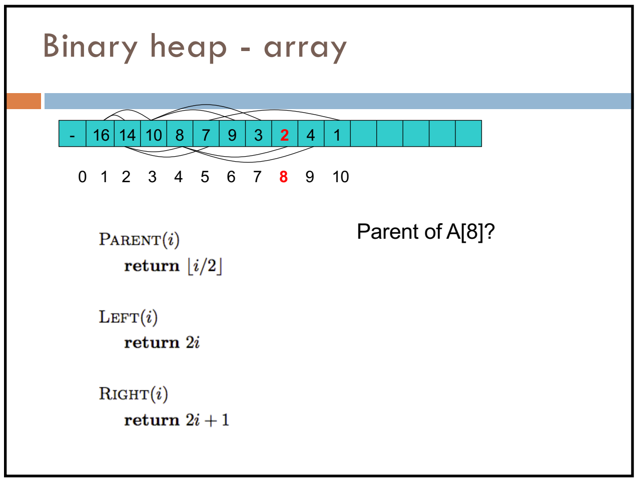
-	16	14	10	8	7	9	3	2	4	1					
0	1	2	3	4	5	6	7	8	9	10					

```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

Left child of A[3]?

2*3 = 6

21



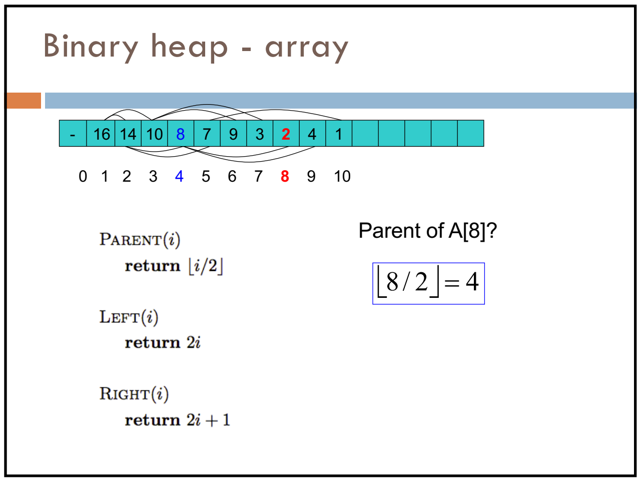
Binary heap - array

-	16	14	10	8	7	9	3	2	4	1					
0	1	2	3	4	5	6	7	8	9	10					

```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

Parent of A[8]?

22



Binary heap - array

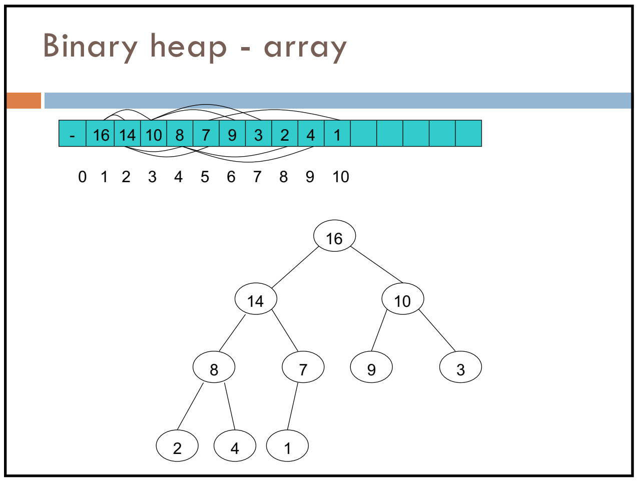
-	16	14	10	8	7	9	3	2	4	1					
0	1	2	3	4	5	6	7	8	9	10					

```
PARENT($i$)
    return $\lfloor i/2 \rfloor$

LEFT($i$)
    return $2i$

RIGHT($i$)
    return $2i + 1$
```

Parent of A[8]?

$\boxed{\lfloor 8/2 \rfloor = 4}$

23



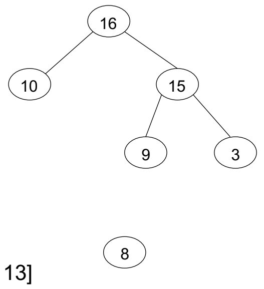
Binary heap - array

-	16	14	10	8	7	9	3	2	4	1					
0	1	2	3	4	5	6	7	8	9	10					

24

Identify the valid heaps

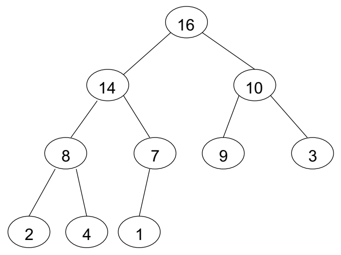
[-, 15, 12, 3, 11, 10, 2, 1, 7, 8]



[-, 20, 18, 10, 17, 16, 15, 9, 14, 13]

25

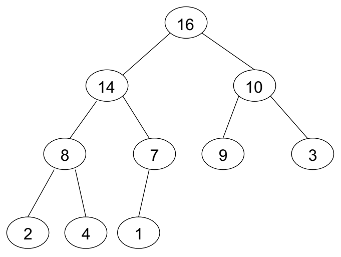
What are heaps good for?



26

What are heaps good for?

What's the largest value in this heap?

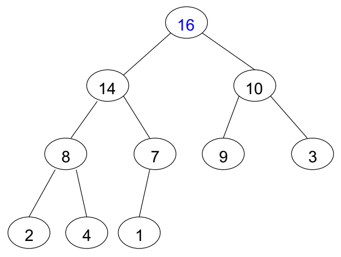


27

What are heaps good for?

What's the largest value in this heap?

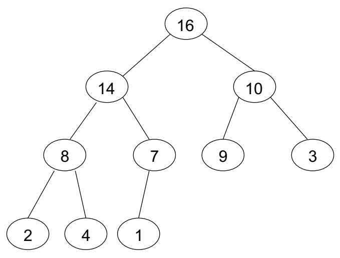
Heaps are good at min/max operations (depending on min/max ordering)!



28

What are heaps good for?

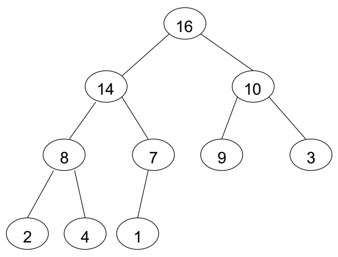
What's the 2nd largest value? The 3rd? The 4th?



29

ExtractMax

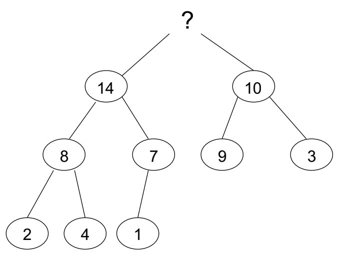
Return and remove the largest element in the set. The rest of the data should stay as a heap



30

ExtractMax

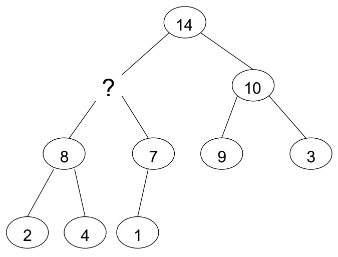
Return and remove the largest element in the set. The rest of the data should stay as a heap



31

ExtractMax

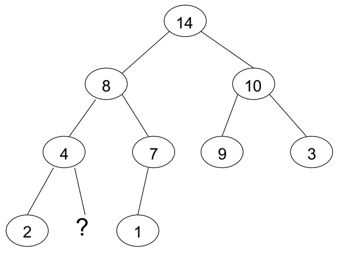
Return and remove the largest element in the set. The rest of the data should stay as a heap



32

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

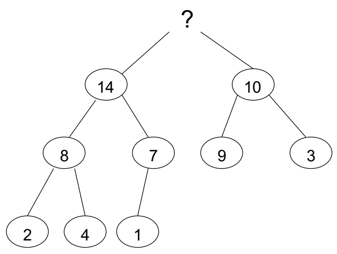


Now what? Is this a heap?

33

ExtractMax

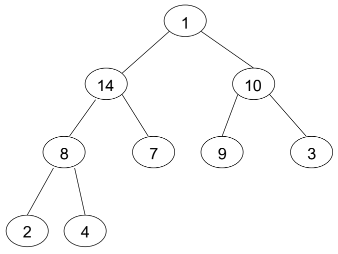
Return and remove the largest element in the set. The rest of the data should stay as a heap



34

ExtractMax

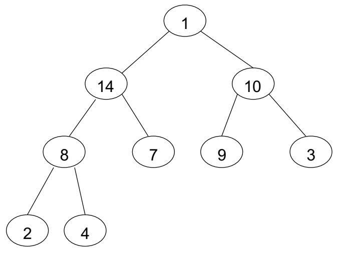
Return and remove the largest element in the set. The rest of the data should stay as a heap



35

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

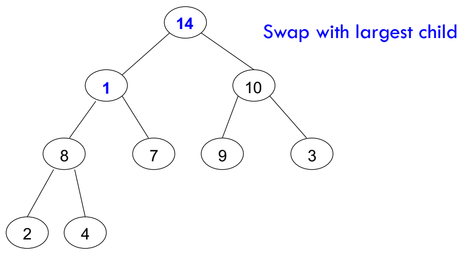


Now what?

36

ExtractMax

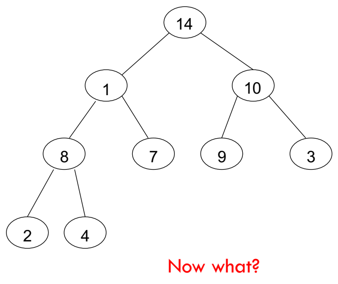
Return and remove the largest element in the set. The rest of the data should stay as a heap



37

ExtractMax

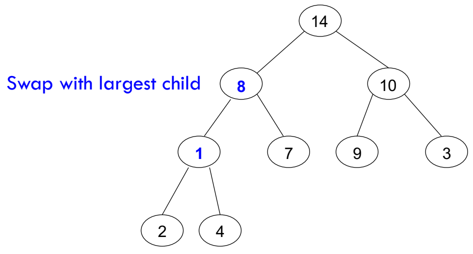
Return and remove the largest element in the set. The rest of the data should stay as a heap



38

ExtractMax

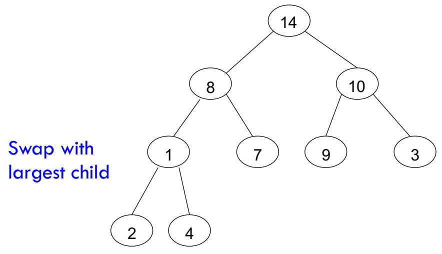
Return and remove the largest element in the set. The rest of the data should stay as a heap



39

ExtractMax

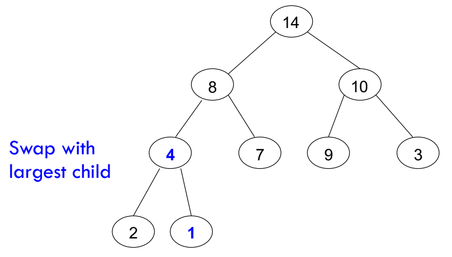
Return and remove the largest element in the set. The rest of the data should stay as a heap



40

ExtractMax

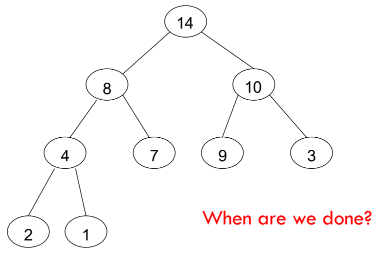
Return and remove the largest element in the set. The rest of the data should stay as a heap



41

ExtractMax

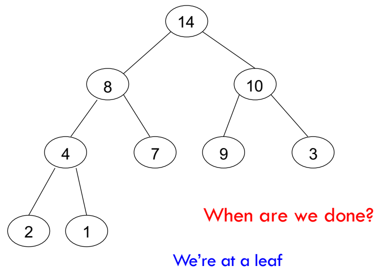
Return and remove the largest element in the set. The rest of the data should stay as a heap



42

ExtractMax

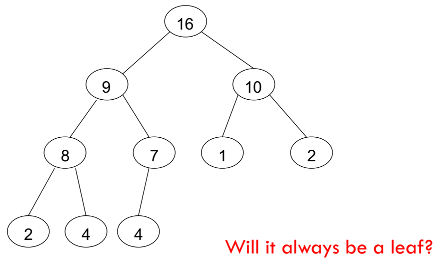
Return and remove the largest element in the set. The rest of the data should stay as a heap



43

ExtractMax

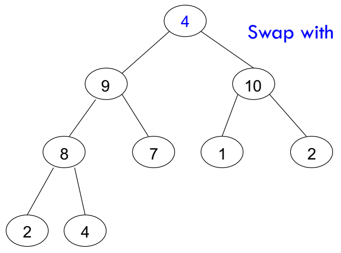
Return and remove the largest element in the set. The rest of the data should stay as a heap



44

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

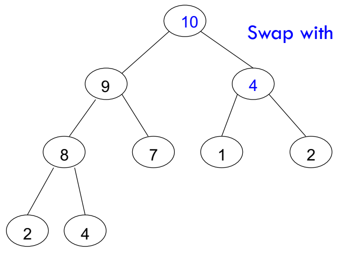


Swap with largest child

45

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

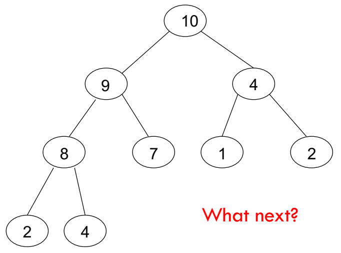


Swap with largest child

46

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap

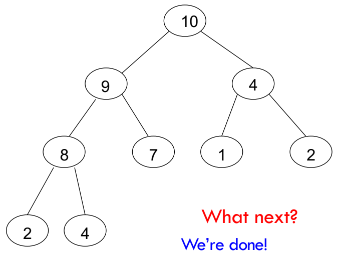


What next?

47

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap



What next?

We're done!
Also stop if the node is larger than the two children

48

sink/heapify/demote

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

49

sink runtime

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

What is the worst case runtime?

50

sink runtime

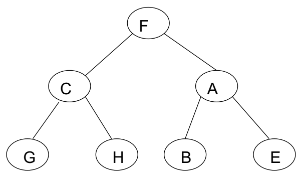
```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

What is the worst case runtime? O(height of tree)

51

Nodes in a binary tree

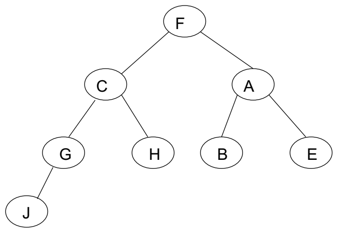
What is the tallest you can make a complete tree, using the fewest nodes?



52

Nodes in a binary tree

What is the tallest you can make a complete tree, using the fewest nodes?

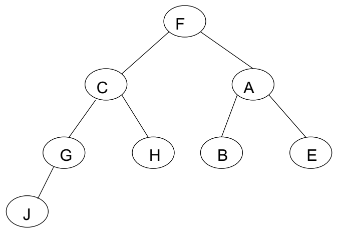


$1 + 2 + 4 + 8 + \cdots + 2^{h-1} + 1 = 2^h$

53

Nodes in a binary tree

What is the tallest you can make a complete tree, using the fewest nodes?



$n = 2^h \Rightarrow h = \log(n)$

54

sink runtime

```
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);

        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);

        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```

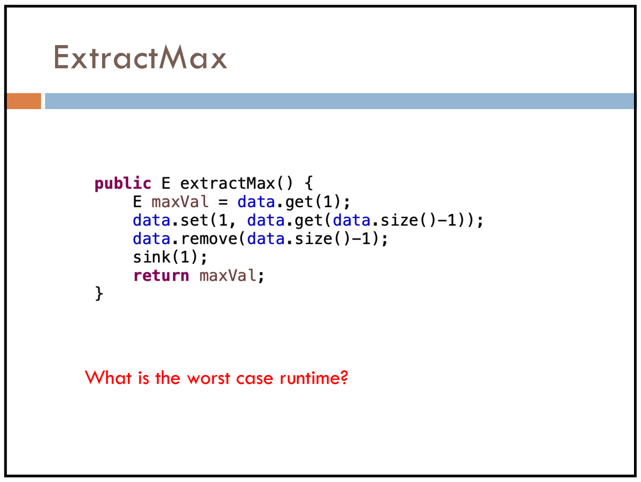
What is the worst case runtime? O(log n)

55

ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

56

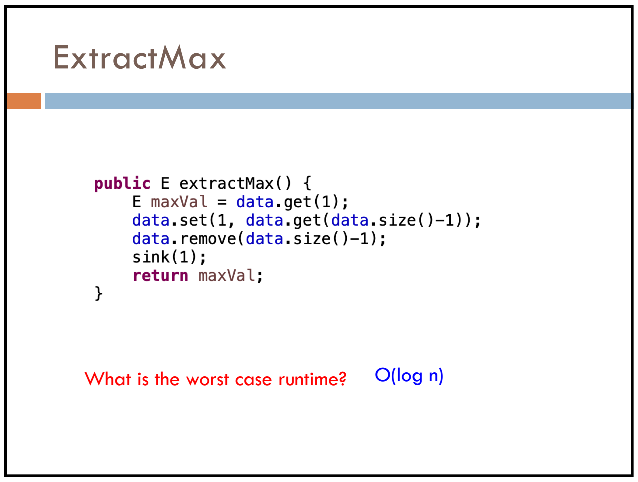


ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

What is the worst case runtime?

57



ExtractMax

```
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

What is the worst case runtime? O(log n)

58