PRIORITY QUEUES

David Kauchak
CS 62 – Spring 2021

1

Admin

Pre-pre enrollment

- thanks all of you that are not potential CS majors for your patience!

Autocomplete assignment

2

Binary heap

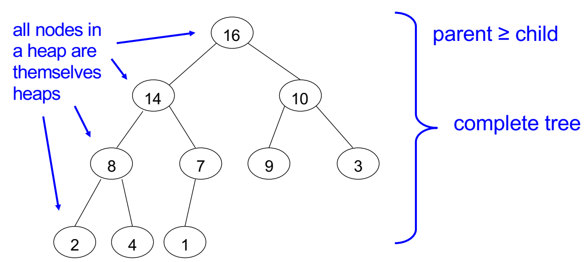
A binary tree where the value of a parent is greater than or equal to the value of its children

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

Max heap vs. min heap

3

Binary heap - references



4

Binary heap - array

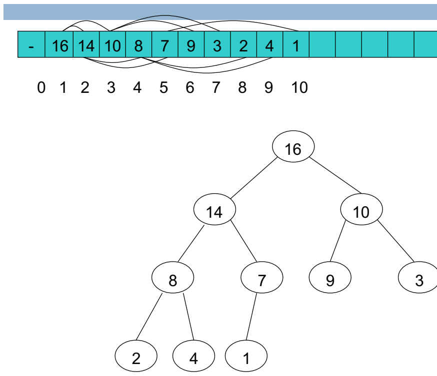
PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

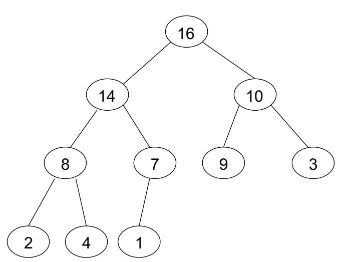
5

Binary heap - array



6

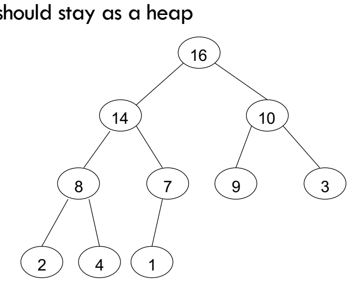
What are heaps good for?



7

ExtractMax

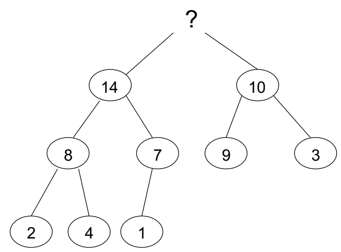
Return and remove the largest element in the set. The rest of the data should stay as a heap



8

ExtractMax

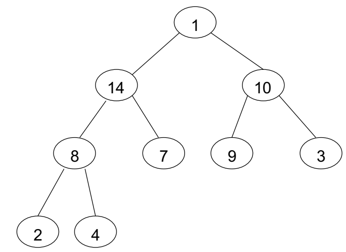
Remove the root



9

ExtractMax

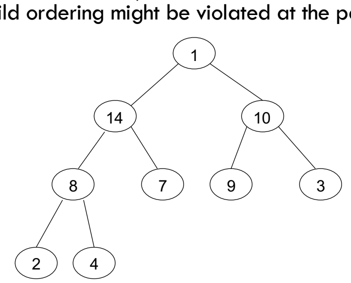
Remove the last node (rightmost leaf on the last level) and put it at the root



10

Sink

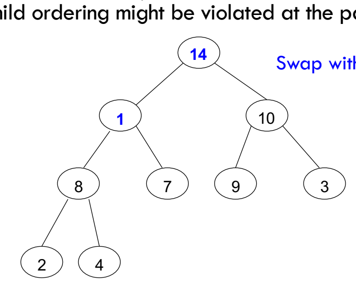
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node



11

Sink

Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node

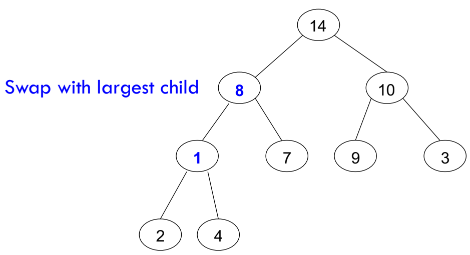


Swap with largest child

12

Sink

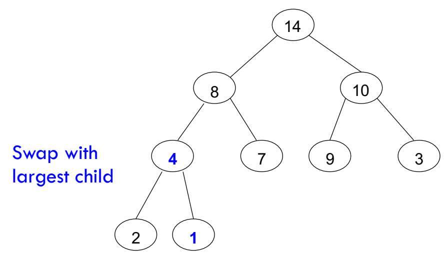
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node



13

Sink

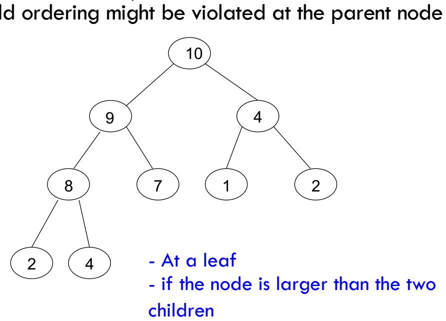
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node



14

Sink: when are we done?

Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node



15

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);
        }
    }
}
```

16

sink runtime

```java
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?

17

sink runtime

```java
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)

18

sink runtime

```java
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)

19

ExtractMax

```java
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;
}
```

- largest value is at the root (index 1)
- move the last value to the root
- remove the last item
- call sink on the root
- return the largest value

20

ExtractMax

```java
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?

21

ExtractMax

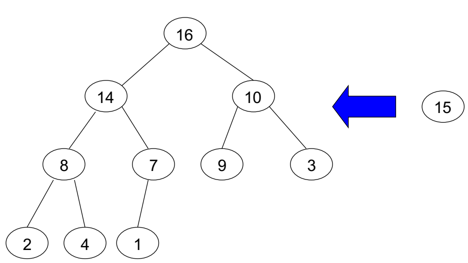
```java
public E extractMax() {
    E maxVal = data.get(1);
    data.set(1, data.get(data.size()-1));
    data.remove(data.size()-1);
    sink(1);
    return maxVal;
}
```

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

22

Insert

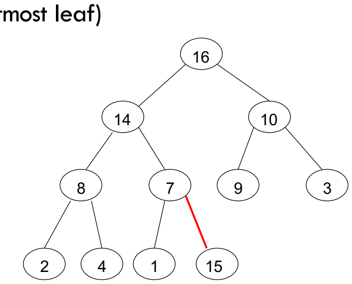
How do we insert a value into a heap?



23

Insert

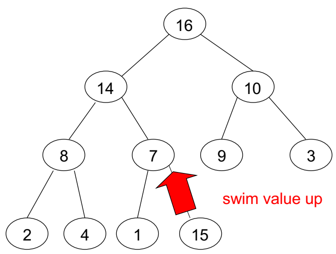
Insert the value at the end of the array (or as the rightmost leaf)



24

Insert

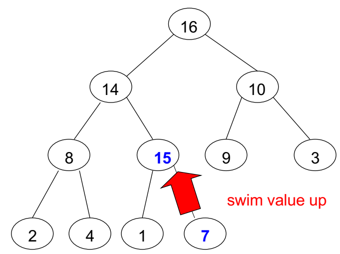
Swap the value up until it's in the right place



25

Insert

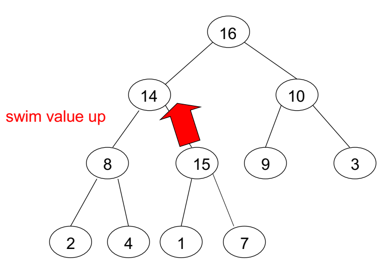
Swap the value up until it's in the right place



26

Insert

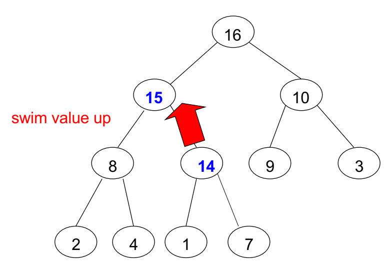
Swap the value up until it's in the right place



27

Insert

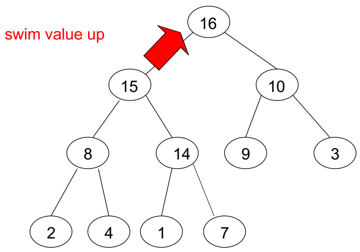
Swap the value up until it's in the right place



28

Insert

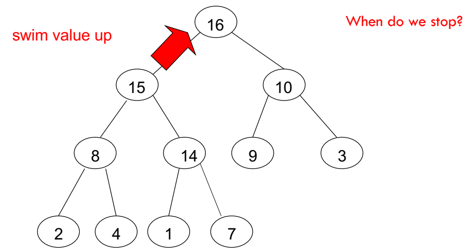
Swap the value up until it's in the right place



29

Insert

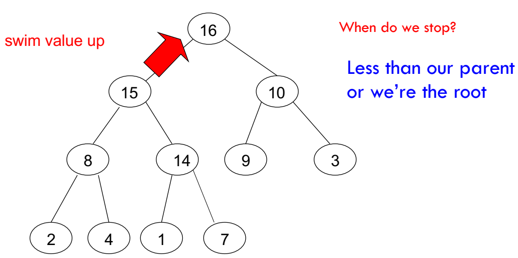
Swap the value up until it's in the right place



30

Insert

Swap the value up until it's in the right place



31

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

32

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

What's the worst case runtime?

33

swim/percolate up

```
private void swim(int i) {
    if( i > 1 ) {
        E value = data.get(i);
        E parentVal = data.get(parent(i));

        if( value.compareTo(parentVal) > 0 ) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```

What's the worst case runtime? O(height of tree) = O(log n)

34

insert

```
public void insert(E val) {
    data.add(val);
    swim(data.size()-1);
}
```

35

insert

```
public void insert(E val) {
    data.add(val);
    swim(data.size()-1);
}
```

What's the worst case runtime?

36

insert

```java
public void insert(E val) {
    data.add(val);
    swim(data.size()-1);
}
```

What's the worst case runtime? O(log n)

37

Heaps summarized

Very good at extracting min/max (depending on heap ordering)

	best	worst	average
max	O(1)	O(1)	O(1)
extractMax	O(1)	O(log n)	O(log n)
insert	O(1)	O(log n)	O(log n)
change node	O(1)	O(log n)	O(log n)

38

Heapsort

Could we sort data with a heap?

What would be the runtime (best, average, worst)?

	best	worst	average
max	O(1)	O(1)	O(1)
extractMax	O(1)	O(log n)	O(log n)
insert	O(1)	O(log n)	O(log n)
change node	O(1)	O(log n)	O(log n)

39

Heapsort

Build a heap out of the data (e.g., insert n items into heap)

Call extractMin n times and add to answer

40

Heapsort runtime

Build a heap out of the data (e.g., insert n items into heap)

Call extractMin n times and add to answer

Best case?

Worst case?

Average case?

41

Heapsort runtime

Best case? O(n) – when all items have the same value

Worst case? O(n log n)

Average case? O(n log n)

	best	worst	average
max	O(1)	O(1)	O(1)
extractMax	O(1)	O(log n)	O(log n)
insert	O(1)	O(log n)	O(log n)
change node	O(1)	O(log n)	O(log n)

42

Heapsort

Build a heap out of the data (e.g., insert n items into heap)

Call extractMin n times and add to answer

Stable? No.

In-place? Not this implementation, but can be done without too much trouble

43

Sorting summarized

	in-place?	stable?	Best	Average	Worst	Notes
Selection	X		$O(n^2)$	$O(n^2)$	$O(n^2)$	n swaps
Insertion	X	X	O(n)	$O(n^2)$	$O(n^2)$	use for partially ordered
Merge		X	O(n log n)	O(n log n)	O(n log n)	guaranteed, stable
Quick	X		O(n log n)	O(n log n)	$O(n^2)$	fastest in practice
Heap	X		O(n)	O(n log n)	O(n log n)	guaranteed, in-place

44

Priority Queues

Queues work well for keeping track of sequential ordering when everything is equivalent (e.g., waiting in line to get lunch!)

Some queues everything is not equivalent (e.g., ER waiting room)

Priority queues support add/remove **orderd by a weight/priority**

45

Priority Queues

Applications?

46

Priority Queues

Applications?

- process scheduling (e.g., ‘top’ command)
- network traffic scheduling
- Many algorithms
  - Search algorithms (A*)
  - Shortest paths algorithms (Dijsktra’s)
  - Minimum spanning trees (Prim’s)
  - Huffman codes

47

Priority queue interface

```java
public interface PriorityQueue <E extends Comparable<E>>{
    /**
     * Returns the smallest value in the queue if non-empty
     *
     * @return the smallest value in the priority queue
     */
    public E extractMin();

    /**
     * Adds the specified item to the priority queue
     *
     * @param data data item to be added
     */
    public void add(E data);

    /**
     * Returns the number of elements in the queue
     *
     * @return size of the queue
     */
    public int size();

    public boolean isEmpty();
}
```

48

Priority queue

two key methods:
- add
- extractMin (highest priority)

How can we do this?

See how many options you can come up with that have *different* runtimes for operations!

49

Option 1: unordered ArrayList

add:

extractMin:

50

Option 1: unordered ArrayList

add: add to the end of the ArrayList

extractMin: search for the smallest, return and remove it

Worst case running times?

51

Option 1: unordered ArrayList

add: add to the end of the ArrayList
O(1) amortized

extractMin: search for the smallest, return and remove it
O(n)

Worst case running times?

52

Option 1b: unordered LinkedList

add: add to the end of the linked list

extractMin: search for the smallest, return and remove it

Worst case running times?

53

Option 1b: unordered LinkedList

add: add to the end of the linked list

O(1)

extractMin: search for the smallest, return and remove it

O(n)

Worst case running times?

54

Option 1b: unordered LinkedList

```
public class SimpleLinkedListPriorityQueue<E extends Comparable<E>> implements Priority
    private LinkedList<E> pq = new LinkedList<E>();

    public void add(E data) {
        pq.add(data);
    }

    public E extractMin(){
        E min = pq.get(0);

        for( E temp: pq ){
            if( temp.compareTo(min) < 0 ){
                min = temp;
            }
        }

        pq.remove(min);
        return min;
    }

    public boolean isEmpty() {
        return pq.size() == 0;
    }

    public int size() {
        return pq.size();
    }
}
```

55

Option 2: sorted order linked list

add:

extractMin:

56

Option 2: sorted order linked list

add: search for the correct location and insert

extractMin: remove and return the first thing from the list

Worst case running times?

57

Option 2: sorted order linked list

add: search for the correct location and insert
O(n)

extractMin: remove and return the first thing from the list
O(1)

Worst case running times?

58

Option 3: heap

add/insert

extractMin

Worst case running times?

59

Option 3: heap

add/insert
O(log n)

extractMin
O(log n)

Worst case running times?

60

Priority queues summarized

Different scenarios may benefit from different implementations

Priority queue ≠ heap

	add	extractMin
unordered linked list	O(1)	O(n)
sorted linked list	O(n)	O(1)
heap	O(log n)	O(log n)

61