CSC 212

Data Structures and Abstractions
Spring 2016

Lecture 14: Priority Queues and Heaps

1

Administrativia

Ranking (contest)
- congrats!

PA2
- don't have a group yet? we'll assign you one tomorrow
- radiusSearch on kd-trees on your own (ask questions!)
- come prepared for the interview on April 6th
- **15 bonus points on final** if you create a client-server application using sockets!

Consider Linux

2

Balanced BSTs

	sequential search (unordered sequence)	binary search (ordered sequence)	AVL
search	O(n)	O(log n)	O(log n)
insert	O(n)	O(n)	O(log n)
delete	O(n)	O(n)	O(log n)
min/max	O(n)	O(1)	O(log n)
floor/ceiling	O(n)	O(log n)	O(log n)
rank	O(n)	O(log n)	O(log n) **

** requires the use of 'size' at every node

3

Quiz

How to remove data from an AVL tree?

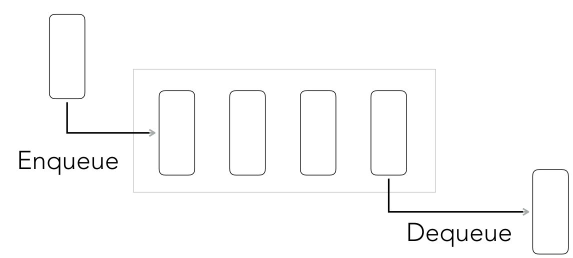
Can we sort using balanced trees? Cost?

4

Priority Queues

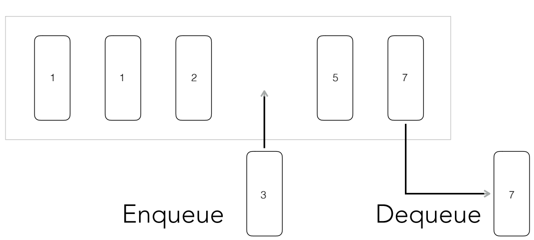
5

Queues



6

Priority Queues



7

Applications

Data Compression (huffman trees)

Network Routing

Process Scheduling (CPUs)

Artificial Intelligence (search)

Graph Algorithms

Stream Data Algorithms

HPC Task Scheduling

. . .

8

Priority Queues

Collections of <Key,Value> pairs
- **keys** are objects on which an **order** is defined

Every pair of keys must be comparable according to a **total order:**

Reflexive Property: $k \leq k$

Antisymmetric Property: if $k_1 \leq k_2$ and $k_2 \leq k_1$, then $k_1 = k_2$

Transitive Property: if $k_1 \leq k_2$ and $k_2 \leq k_3$, then $k_1 \leq k_3$

9

Priority Queues

Queues

basic operations: **enqueue**, **dequeue**

always remove the item least recently added

Priority Queues (MaxPQ)

basic operations: **insert, removeMax**

always remove the item with **highest (max) priority**

Can also be implemented as a MinPQ

10

Example (MinPQ)

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

From Algorithm Design and Applications, Goodrich & Tamassia

11

Performance?

	Sorted Array/List	Unsorted Array/List	AVL
insert			
removeMax			
max			

12

Performance

	Sorted Array/List	Unsorted Array/List	AVL
insert	O(n)	O(1)	O(log n)
removeMax	O(1)	O(n)	O(log n)
max	O(1)	O(n)	O(log n)

13

Sorting !

Running time?

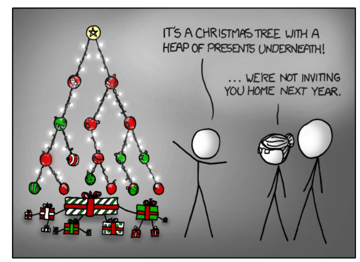
The running time of this sorting method depends on the priority queue implementation.

```
//
// what does this function do?
//
void foo(int *array, int n) {
    std::priority_queue<int> pq;

    for (int i = 0 ; i < n ; i ++)
        pq.push(array[i]); // insert()

    while (! pq.empty()) {
        array[--n] = pq.top(); // max()
        pq.pop(); // removeMax()
    }
}
```

14

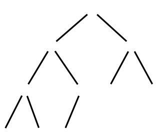


From https://xkcd.com/835/

Heaps

15

(max) Heap



Structure Property

a heap is a **complete binary tree**

Heap-Order Property

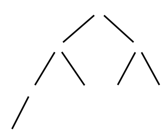
for every node **x**, **key(parent(x)) >= key(x)**

except the root, which has no parent

16

Height of a heap?

What is the minimum number of nodes in a complete binary tree of height **h**?



$n \geq 2^h$

$\log n \geq \log 2^h$

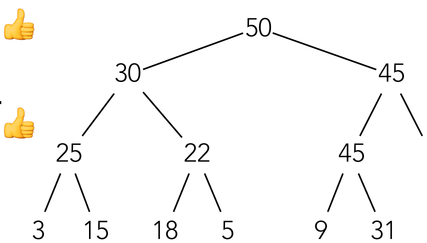
$\boxed{\log n \geq h}$

17

Example

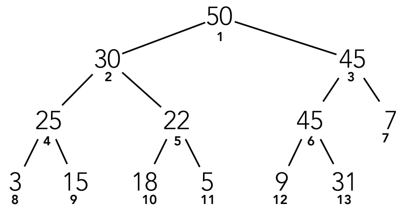
Structure Property?

Heap-order Property?



18

Implementation



parent(i)
`floor(i/2)`

left_child(i)
`i*2`

right_child(i)
`i*2 + 1`

Complete tree …

50	30	45	25	22	45	7	3	15	18	5	9	31
1	2	3	4	5	6	7	8	9	10	11	12	13

19

insert

20

Insert

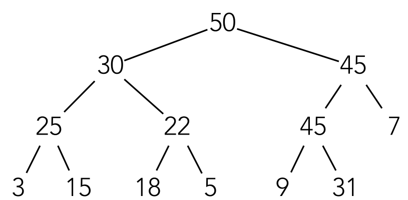
Append new element to the end of array

Check heap-order property

- if violated, **Up-Heap** (swap with parent)
  - **repeat** until heap-order is restored
- if not, we are done

O(log n)

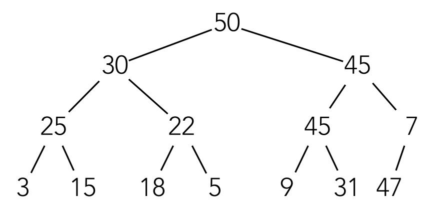
21



insert 47

50	30	45	25	22	45	7	3	15	18	5	9	31			

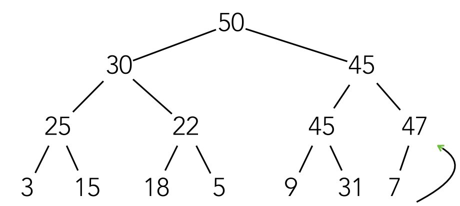
22



insert 47

50	30	45	25	22	45	7	3	15	18	5	9	31	47		

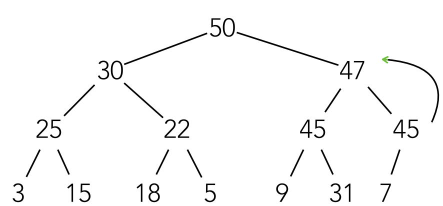
23



insert 47

50	30	45	25	22	45	47	3	15	18	5	9	31	7		

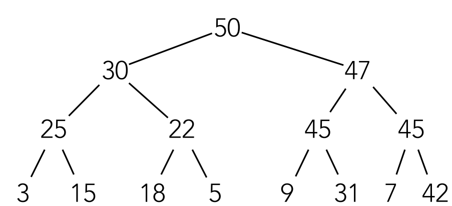
24



insert 47

50	30	47	25	22	45	45	3	15	18	5	9	31	7		

25



insert 42

50	30	47	25	22	45	45	3	15	18	5	9	31	7	42	

26

removeMax

27

removeMax

Max element is the the **first** element of the array
- the root of the heap

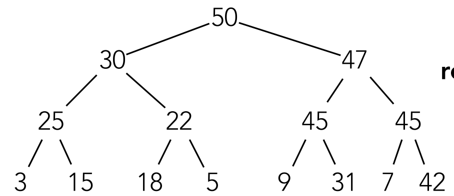
Copy last element of array to first position
- then decrement array size by 1 (removes last element)

Check heap-order property
- if violated, **Down-Heap** (swap with **larger** child)
  - **repeat** until heap-order is restored
- if not, we are done

O(log n)

28

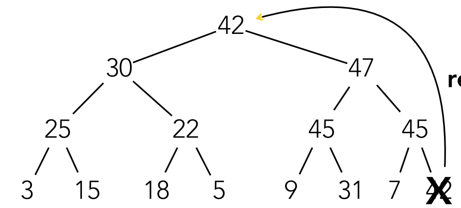
removeMax



50	30	47	25	22	45	45	3	15	18	5	9	31	7	42	

29

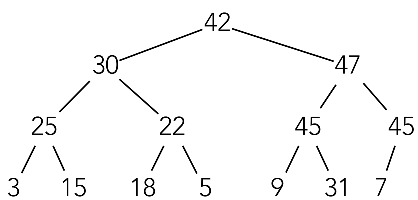
removeMax



42	30	47	25	22	45	45	3	15	18	5	9	31	7	42	

30

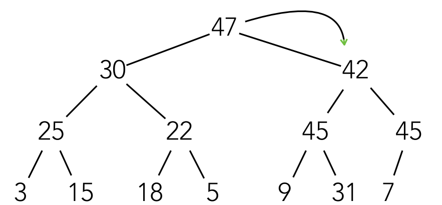
removeMax



42	30	47	25	22	45	45	3	15	18	5	9	31	7		

31

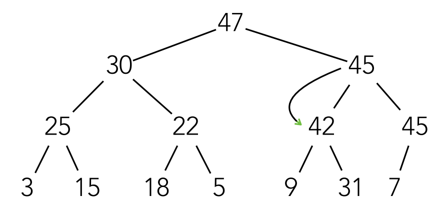
removeMax



47	30	42	25	22	45	45	3	15	18	5	9	31	7		

32

removeMax



47	30	45	25	22	42	45	3	15	18	5	9	31	7		

33

Performance

	Sorted Array/List	Unsorted Array/List	AVL	Heap
insert	O(n)	O(1)	O(log n)	
removeMax	O(1)	O(n)	O(log n)	
max	O(1)	O(n)	O(log n)	
insert N	O(n^2)	O(n)	O(n log n)	

34

Performance

	Sorted Array/List	Unsorted Array/List	AVL	Heap
insert	O(n)	O(1)	O(log n)	O(log n)
removeMax	O(1)	O(n)	O(log n)	O(log n)
max	O(1)	O(n)	O(log n)	O(1)
insert N	O(n^2)	O(n)	O(n log n)	O(n)**

(**) assuming we know the sequence in advance (**buildHeap**)

35

buildHeap

36

Problem

Build a heap by inserting a sequence of **n** elements

'easy': call insert **n** times **O(n log n)**

Can we do it in **linear time**?

37

buildHeap

Place **n** items into the tree (array) in any order

keeps structure property

Perform **Down-Heap** on each internal node

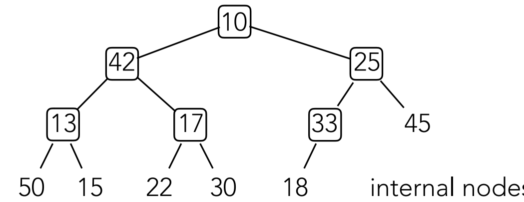
from parent(n) to 1

keeps heap-order property

38

buildHeap example

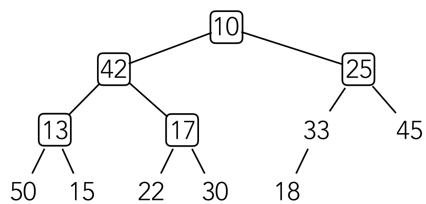
input: 10 42 25 13 17 33 45 50 15 22 30 18



39

buildHeap example

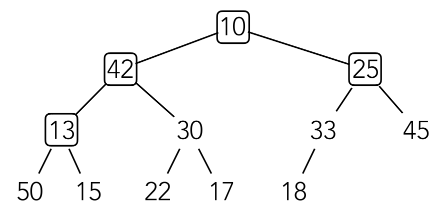
input: 10 42 25 13 17 33 45 50 15 22 30 18



40

buildHeap example

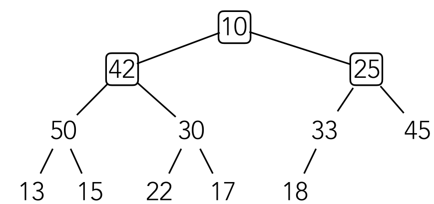
input: 10 42 25 13 17 33 45 50 15 22 30 18



41

buildHeap example

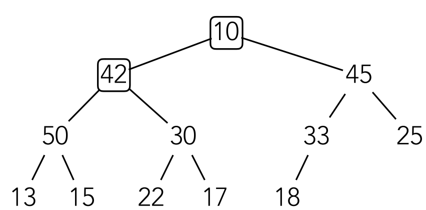
input: 10 42 25 13 17 33 45 50 15 22 30 18



42

buildHeap example

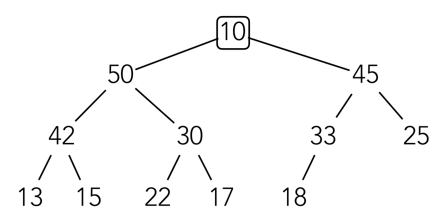
input: 10 42 25 13 17 33 45 50 15 22 30 18



43

buildHeap example

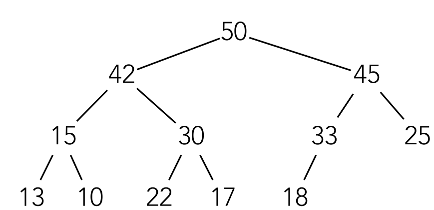
input: 10 42 25 13 17 33 45 50 15 22 30 18



44

buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



45

Analysis

Cost is **sum of the heights** of all internal nodes

assume tree is full and complete, thus **n=2^{h+1}-1**

$$
\begin{aligned}
T(n) &= h + 2(h-1) + 4(h-2) + 8(h-3) + \cdots + 2^h(0) \\
&= \sum_{i=0}^{h} 2^i(h-i) = h\sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i2^i \\
&= h\left[2^{h+1}-1\right] - \left[2 + (h+1-2)2^{h+1}\right] \\
&= 2^{h+1} - h - 2 = n - (h+1) \\
&= O(n)
\end{aligned}
$$

👍