CSE 373: Heaps (Priority Queues)

Chapter 6

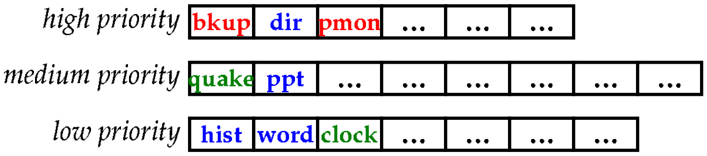
Motivation

We'd like a data structure that stores the programs currently running on a computer

- a queue provides a "fair" data structure since it has FIFO ordering
- but, sometimes things shouldn't be exactly fair
  - system administrator may need to run something of high priority
  - user may have job that isn't urgent
  - interactive applications should perhaps run more often than long numerical computations
  - run short applications first to get them out of the way

One Approach

Use an array of queues



high priority	bkup	dir	pmon	...	...	...		
medium priority	quake	ppt	...	...	...	...	...	...
low priority	hist	word	clock	...	...	...	...	

But what if there were 100 priority levels rather than just three?

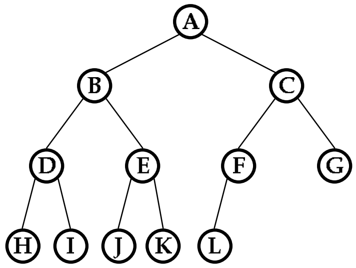
Priority Queue Goals

- We'd like a data structure that allows us to find its lowest (highest) stored value quickly
- Inserts should also be fast
- Current Approaches:

	`findMin()`	`insert()`
– simple list		
– sorted list		
– binary search tree		
– hash table		

(Binary) Heap Structure

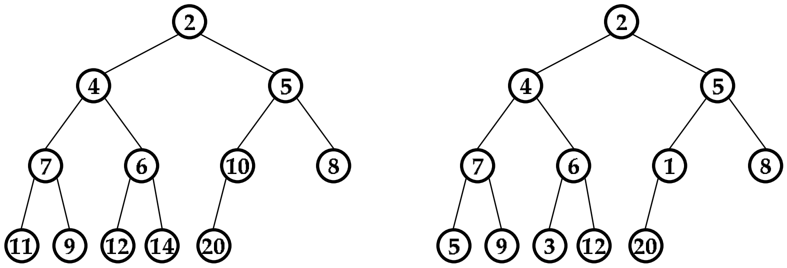
Heaps will always be stored as a *complete* binary tree:



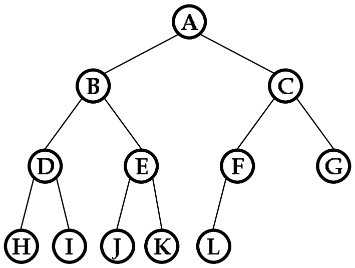
Note that a complete tree's bottom level need not be completely full – but it must fill left to right

Heap Order

Each node must be smaller than its descendants

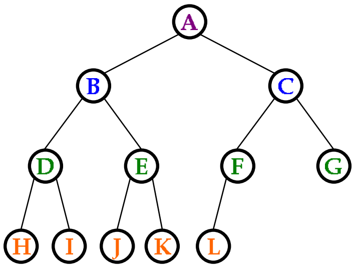


Binary Heap: Array Implementation



–	A	B	C	D	E	F	G	H	I	J	K	L	–	–	–	…
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…

More on Array Implementation

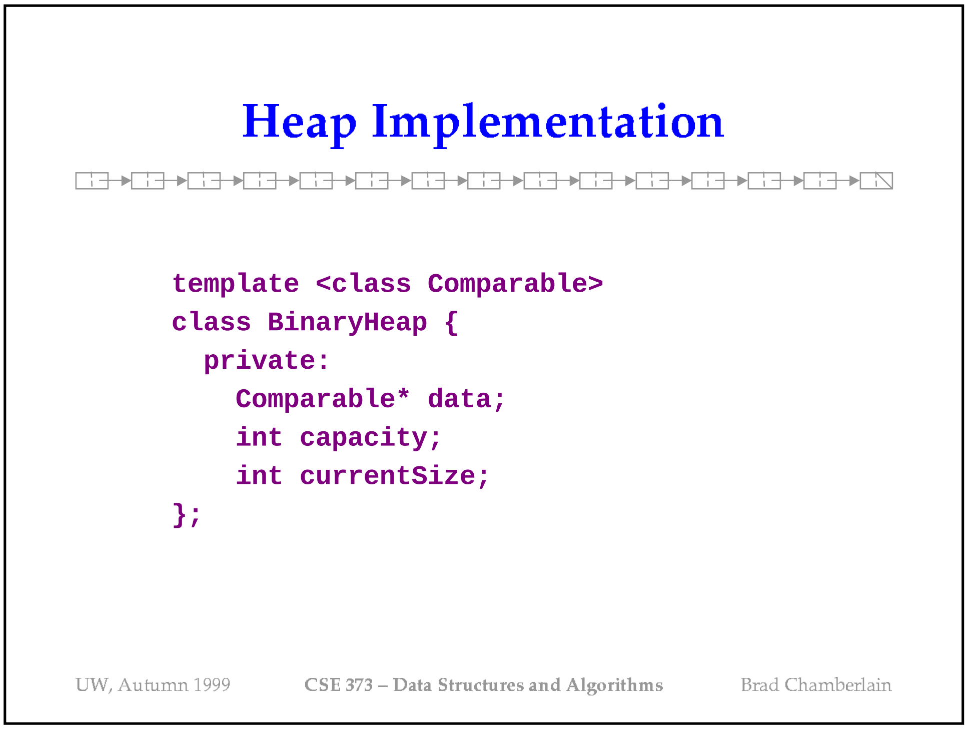


left(i) = $2i$

right(i) = $2i + 1$

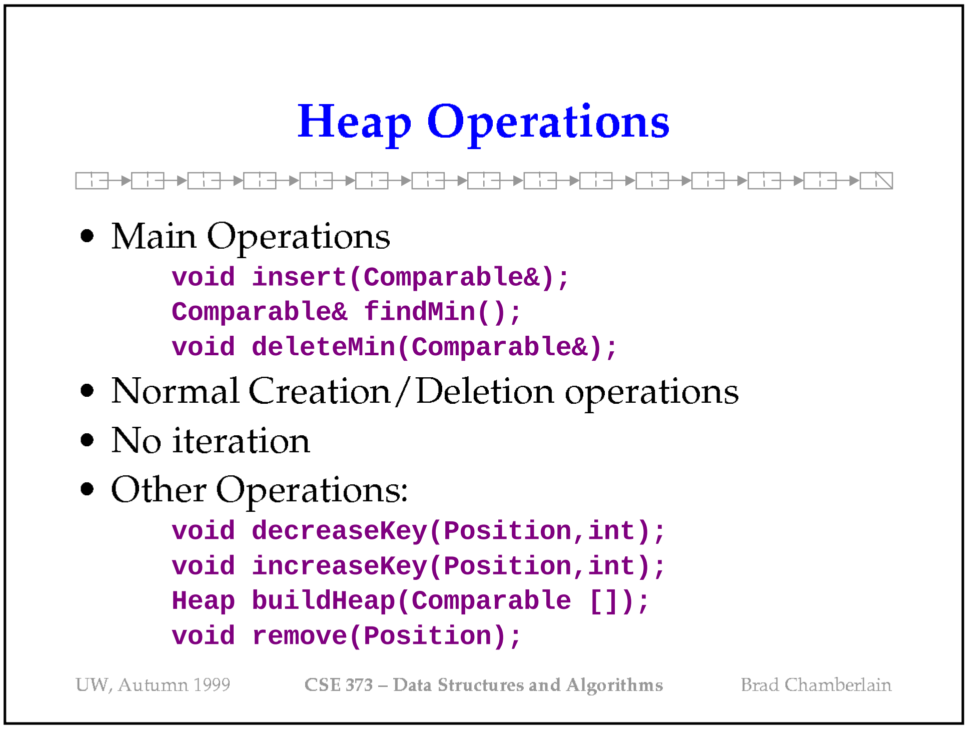
parent(i) = $\lfloor i/2 \rfloor$

–	A	B	C	D	E	F	G	H	I	J	K	L	–	–	–	…
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…



Heap Implementation

```
template <class Comparable>
class BinaryHeap {
  private:
    Comparable* data;
    int capacity;
    int currentSize;
};
```



Heap Operations

- Main Operations

```
void insert(Comparable&);
Comparable& findMin();
void deleteMin(Comparable&);
```

- Normal Creation/Deletion operations
- No iteration
- Other Operations:

```
void decreaseKey(Position,int);
void increaseKey(Position,int);
Heap buildHeap(Comparable []);
void remove(Position);
```

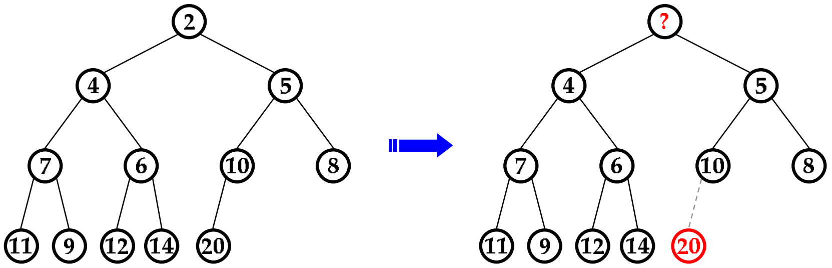
findMin()

- Trivial…

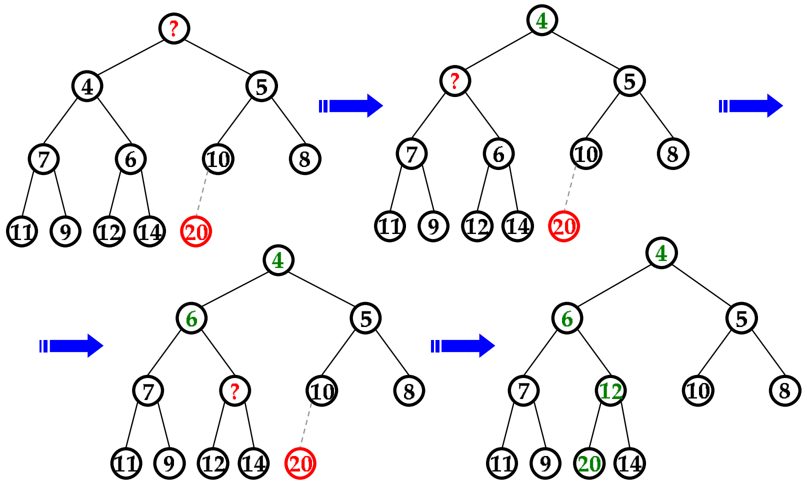
```
Comparable BinaryHeap::findMin() {



}
```

deleteMin()

```
H.deleteMin();
```

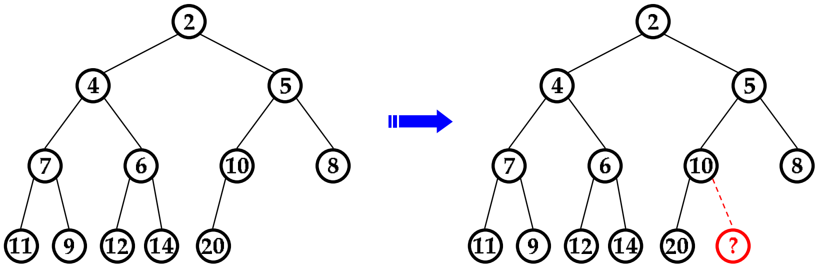


deleteMin() – Continued

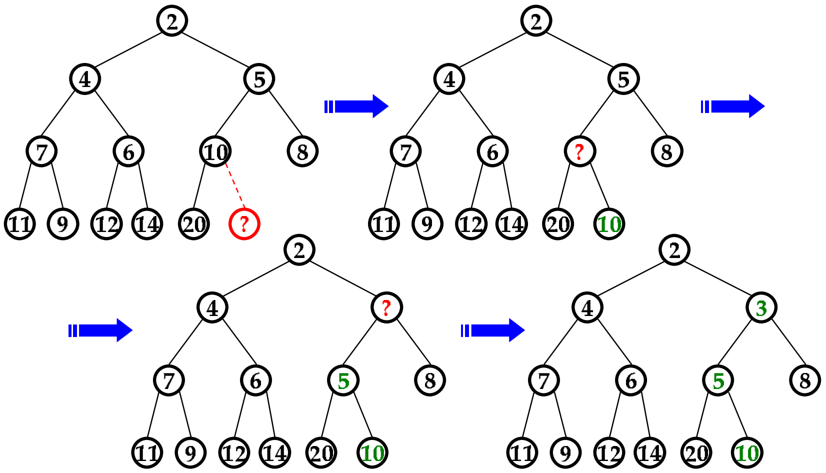


insert()

```
H.insert(3);
```



insert() – Continued



Heap Operator Summary

problem size

space

`findMin()`
`deleteMin()`
`insert()`