CS 180 - Heaps

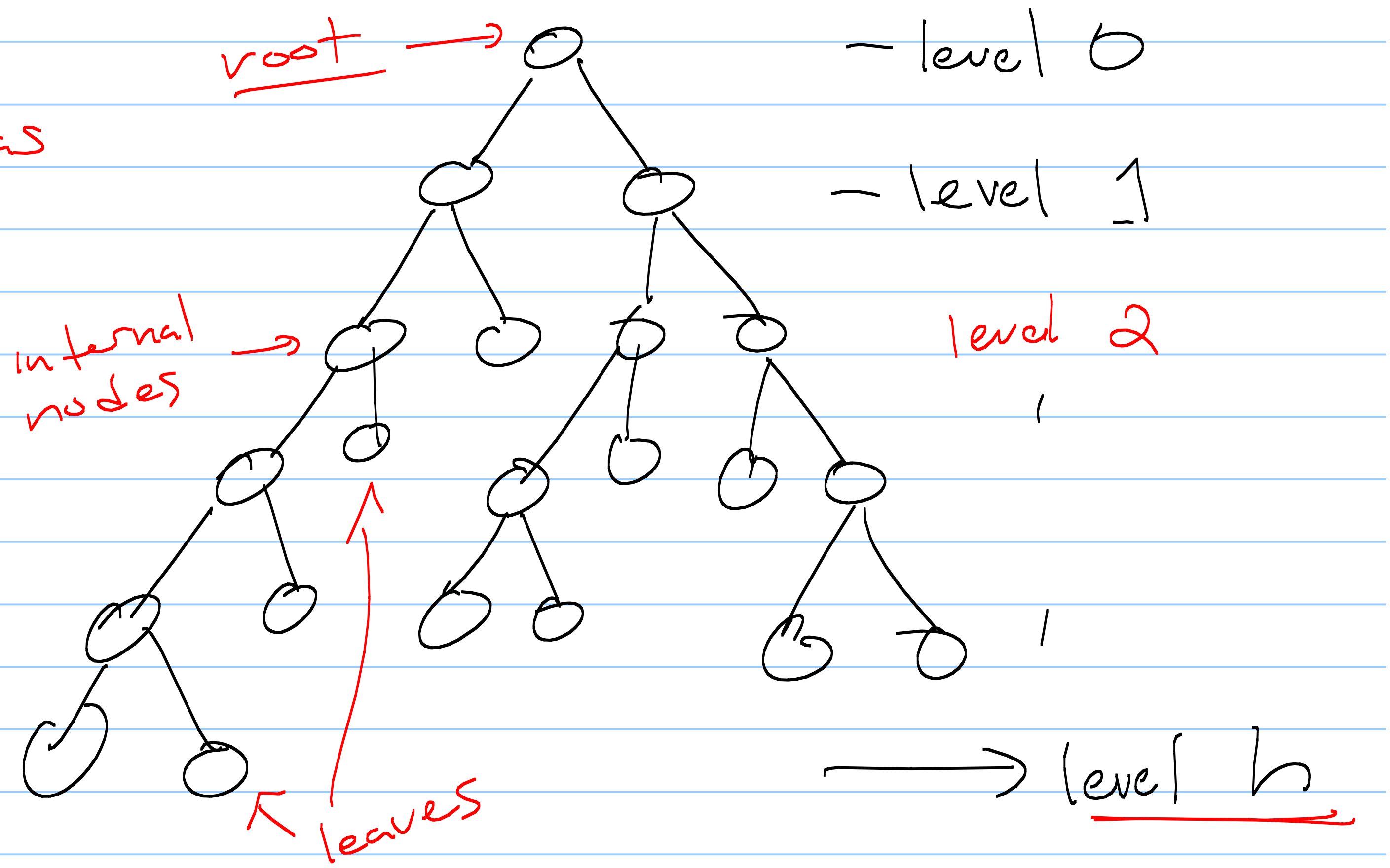
Note Title 11/2/2009

Announcements

- HW due Wednesday
- Next program will come out over the break sometime (due 1 week from Wed.)
- Ritter is closed this weekend (open again Monday) (no office hours Monday)

Last time - binary trees

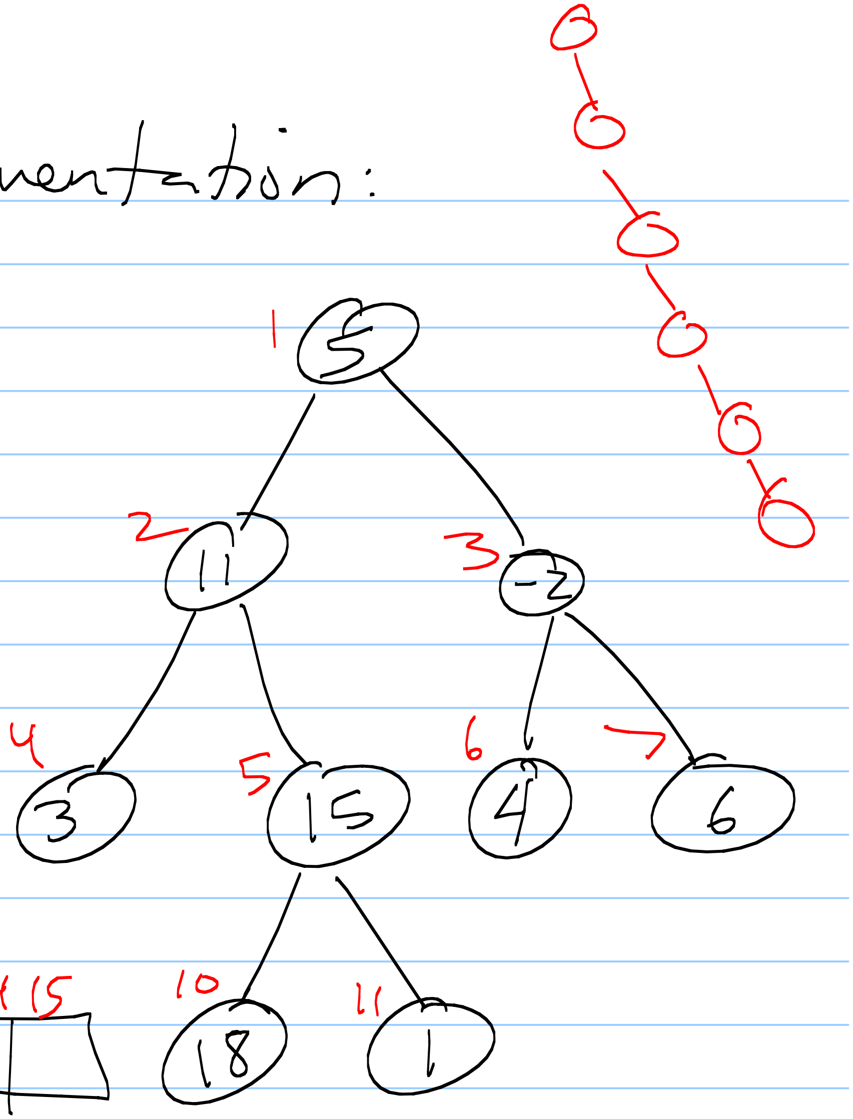
Every node has 0 or 2 children.



Array Based implementation:

Root is #1

For any node v with number n, left child gets number 2·n and right child get 2·n+1



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	11	-2	3	15	4	6	/	/	18	1					

Priority Queue:

if I use List or :
Vector + sorting,
need $O(n)$ ← do better

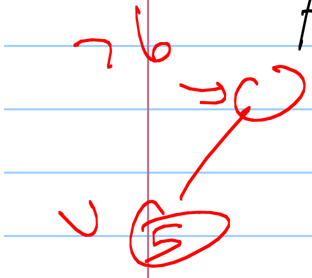
Supports the following operations:

- insert($\overset{\text{priority}}{e}$) : adds element e to the data structure
- removeMax() : removes the maximum element
- maxItem() : returns a reference to the maximum item in P.Q

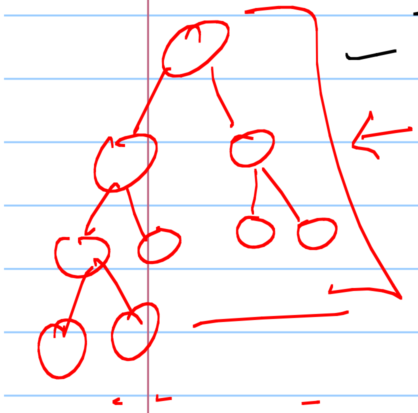
Also: size, empty, etc.

Implementing a priority queue with a heap:

A binary tree where:

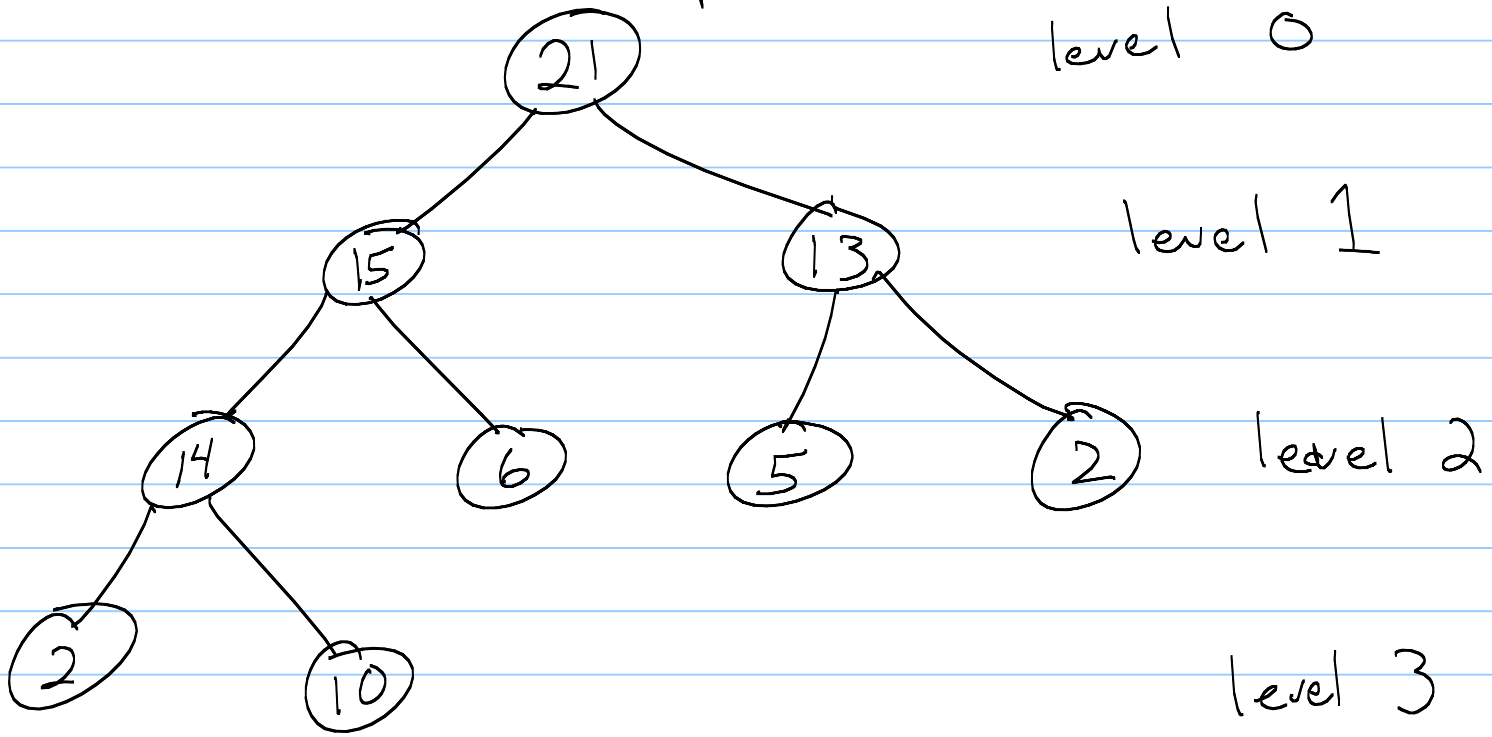


- For every node v (other than the root), the key stored at v is less than or equal to the key stored at v's parent



- The tree is complete - levels 0 to h-1 have all possible nodes, and all internal nodes in level h-1 are on the left

Picture - Max heap



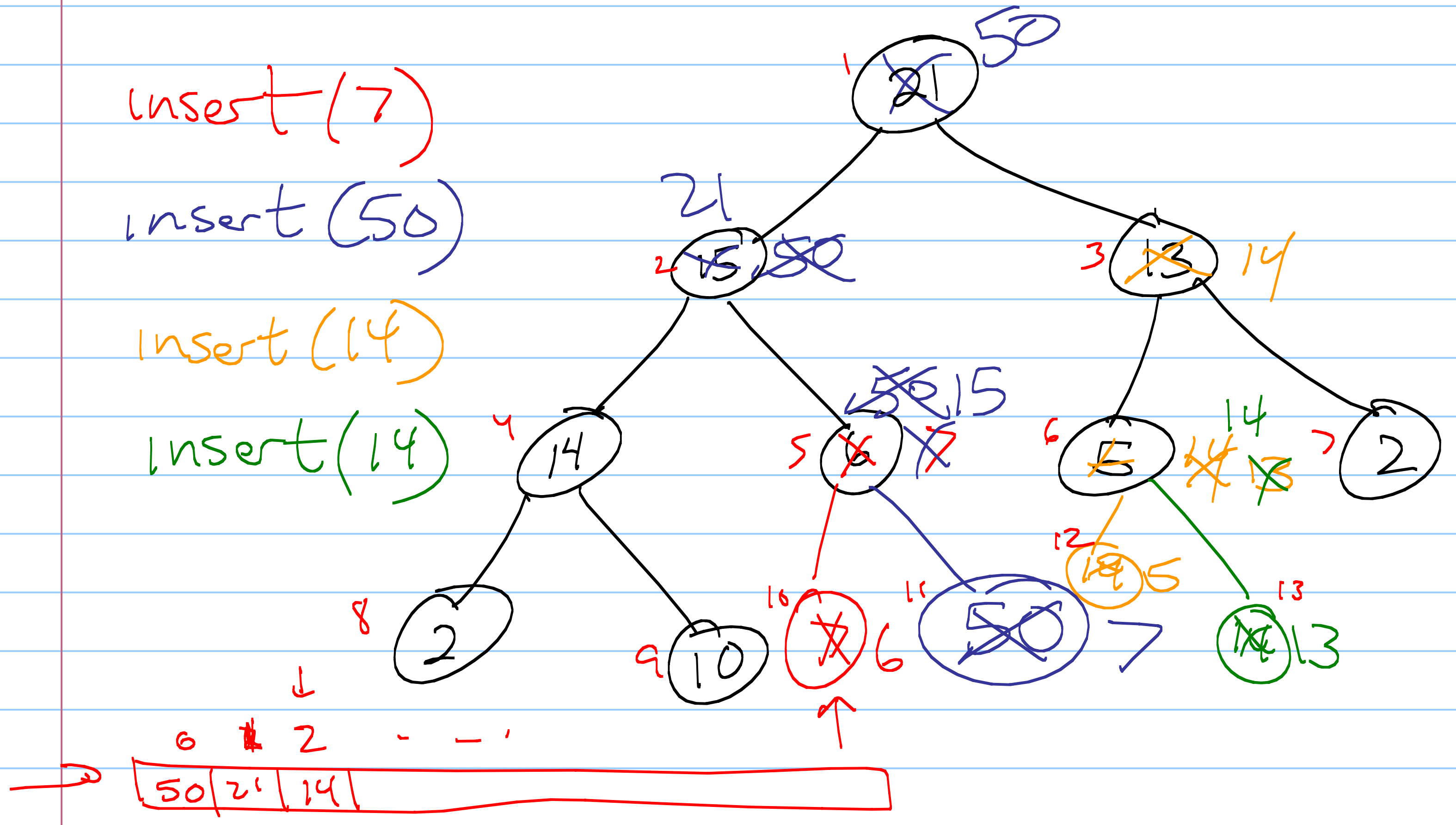
So: insert

insert(7)

insert (50)

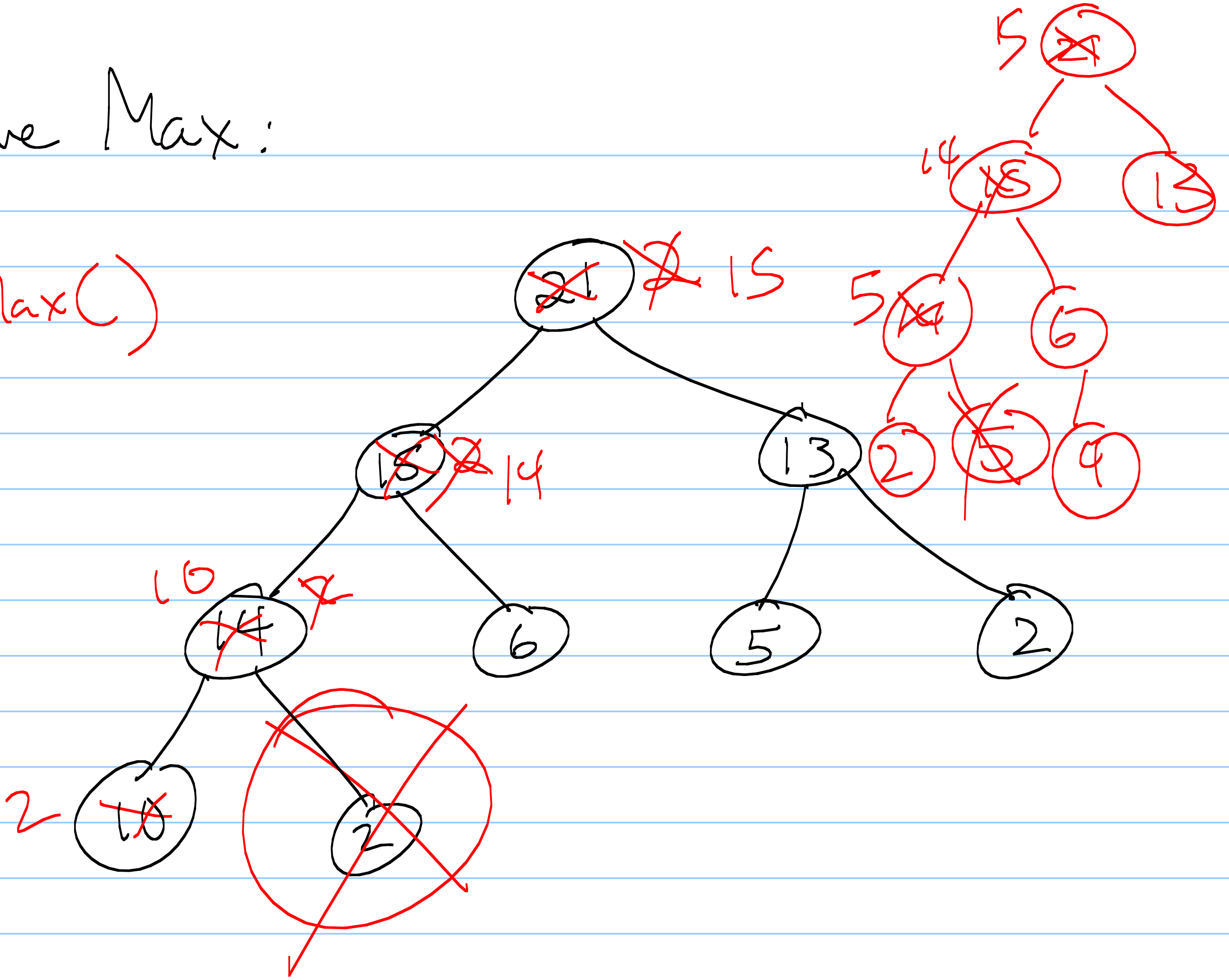
insert (14)

insert(14)



remove Max:

removeMax()



$1+2=3$ ←

$1+2+4=7$ ←

$1+2+4+8=15$ ←

Running times:

insert & removeMax will both run in time $O(h)$, where h is the height of the tree.

How big can h be?

How many nodes are on level i?

2^i nodes on level i

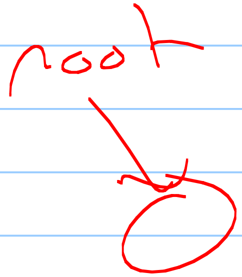
total # nodes $= n = \sum_{i=0}^{h} 2^i = 2^0+2^1+2^2+\dots+2^h = 2^{h+1}-1$

$\log_2 n \approx \log_2 2^{h+1} = h+1$

$\Rightarrow h = \underline{O(\log_2 n)}$

Running Times

Operation	Time
size, empty	$O(1)$
insert	$O(\log n)$
removeMax	$O(\log n)$
maxItem	$O(1)$



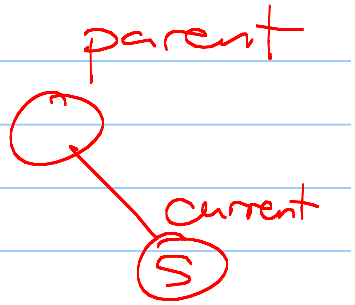
Now - to code it:

```
template <typename ItemType>
class Heap {
    private:
        ItemType*   _data;
        int _size;
        int _capacity;
    public:
        Heap() : _data(new ItemType[1]),
            _size(0), _capacity(1) {}
```

```
void insert(const ItemType& val) {
    if (_size == _capacity) {
        _capacity = 2*_capacty;
        ItemType newdata * = new ItemType[_capacity];
        for (int i=0; i<_size; i++)
            newdata[i] = _data[i];
        delete data;
        data = newdata;
    }
    _data[_size] = val;
    _size ++;
```

```
int current = _size - 1;
int parent = (current - 1)/2;
```

bubble it up


parent
current
5