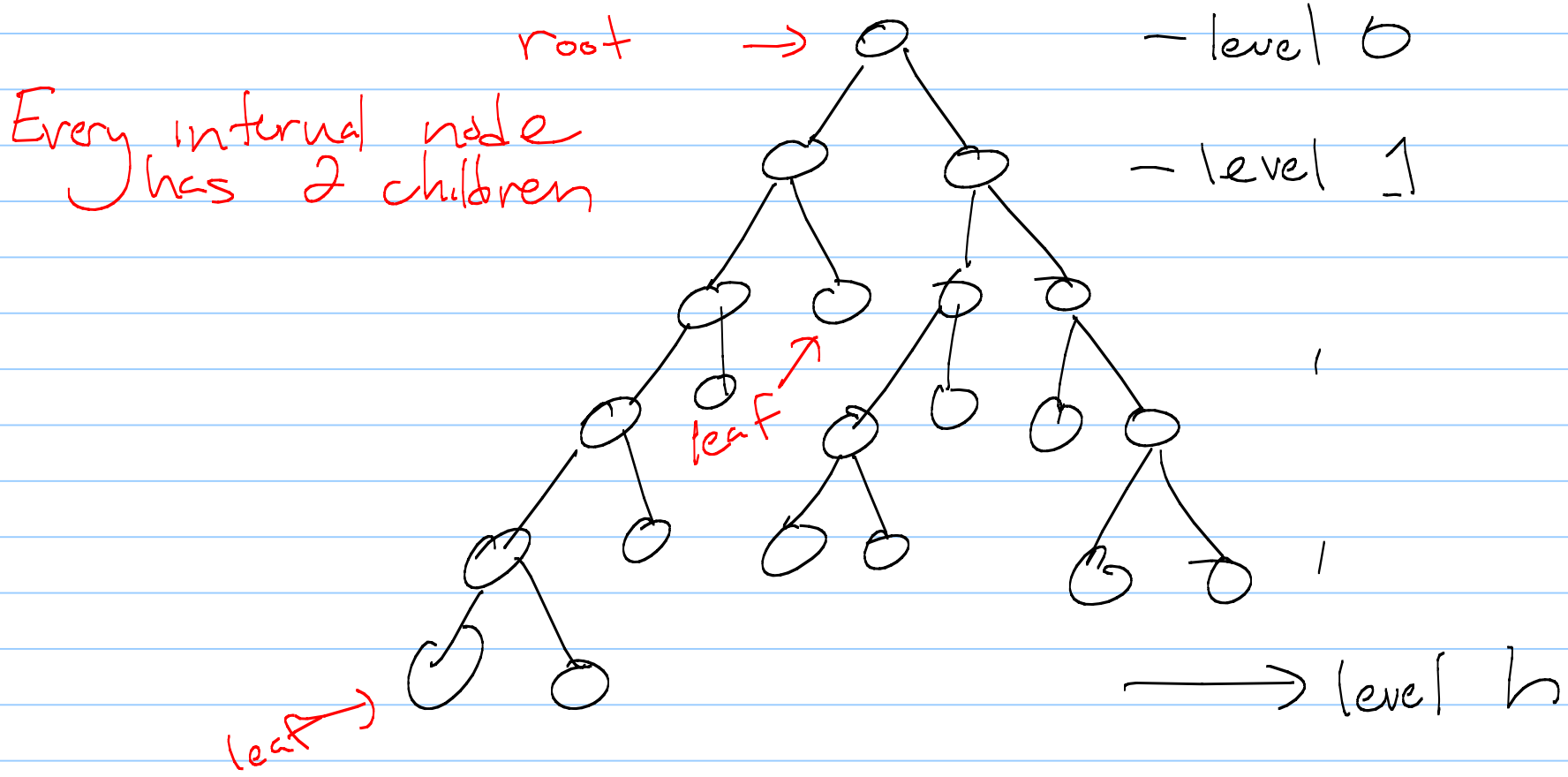
CS 180 - Heaps

Note Title 11/2/2009

Announcements

- Program due this Thursday.
- Checkpoint tomorrow.
  See me after class
- Midterm next week
  (Wed or Thurs)

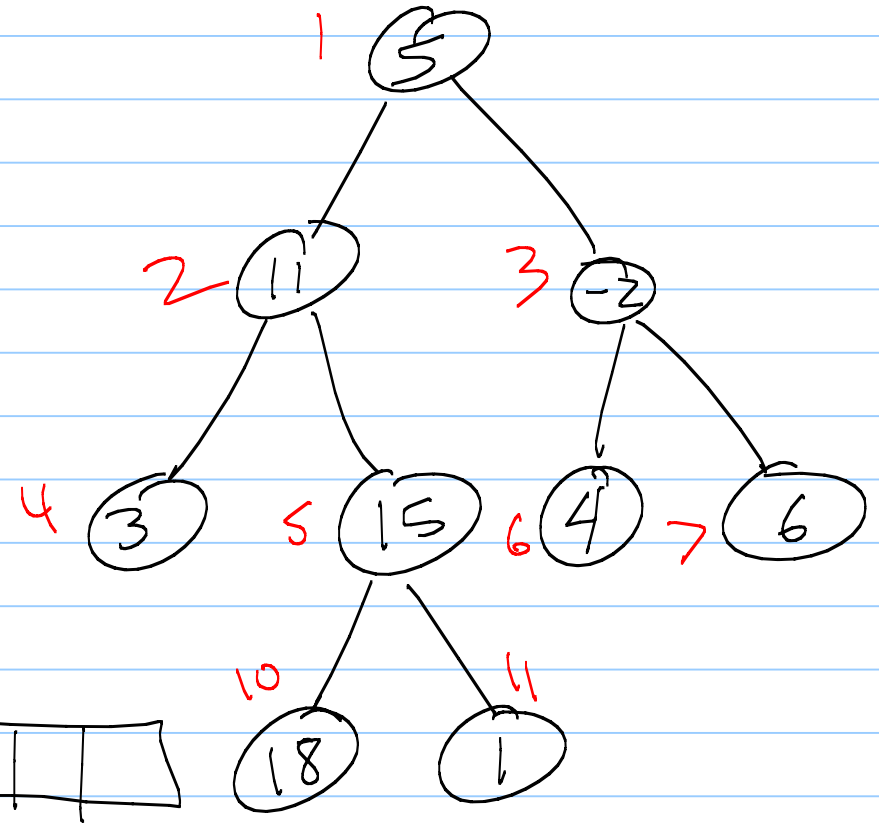
Last time – binary trees

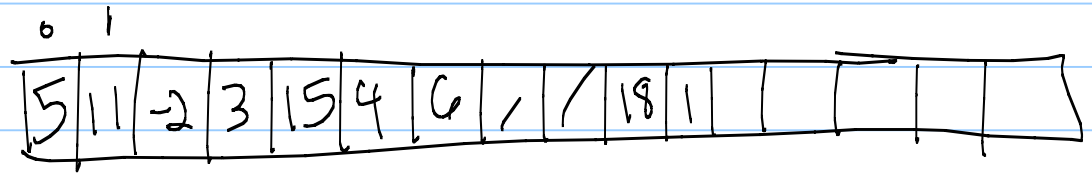


Array Based implementation:

Root is #1

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





Priority Queue:

Supports the following operations:

insert(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.

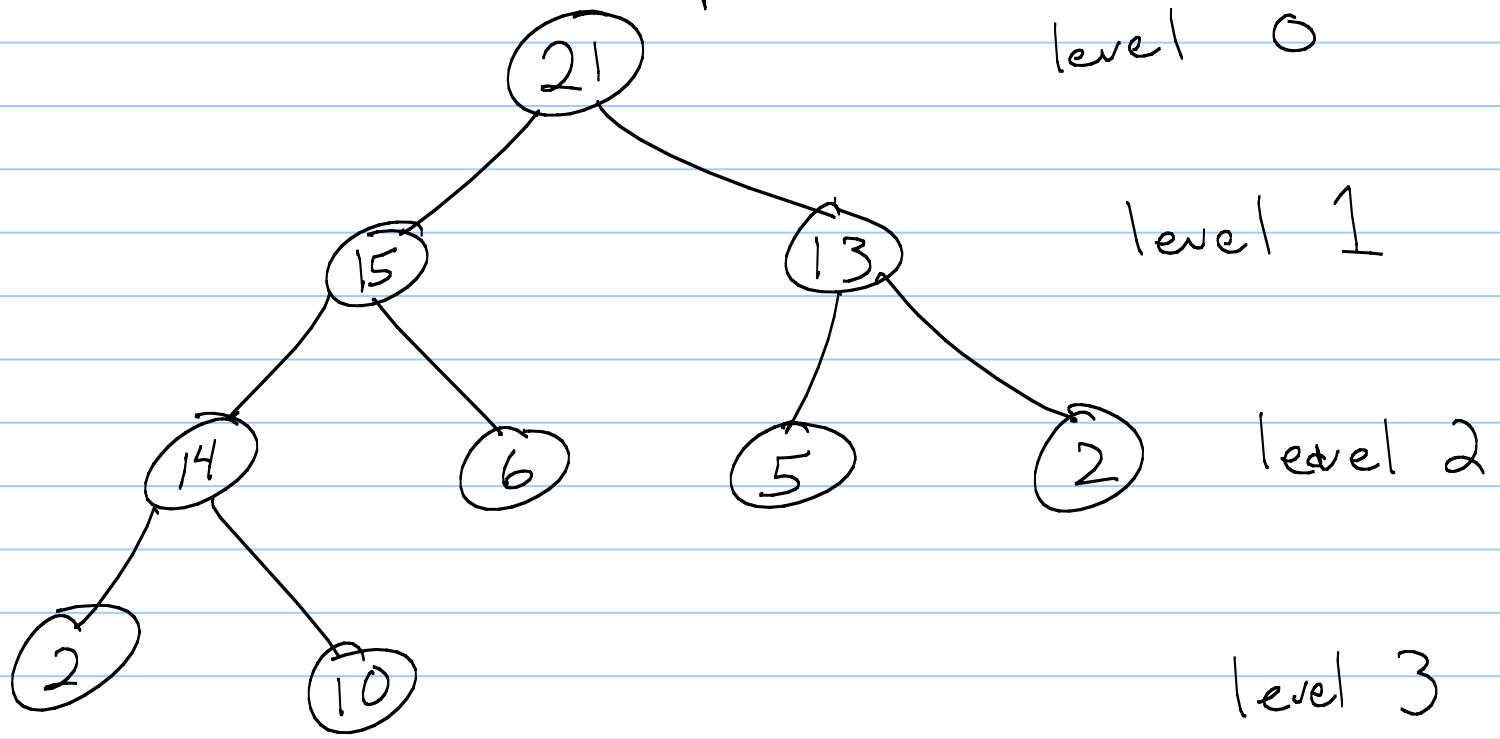
Implementing a priority queue with a heap:

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

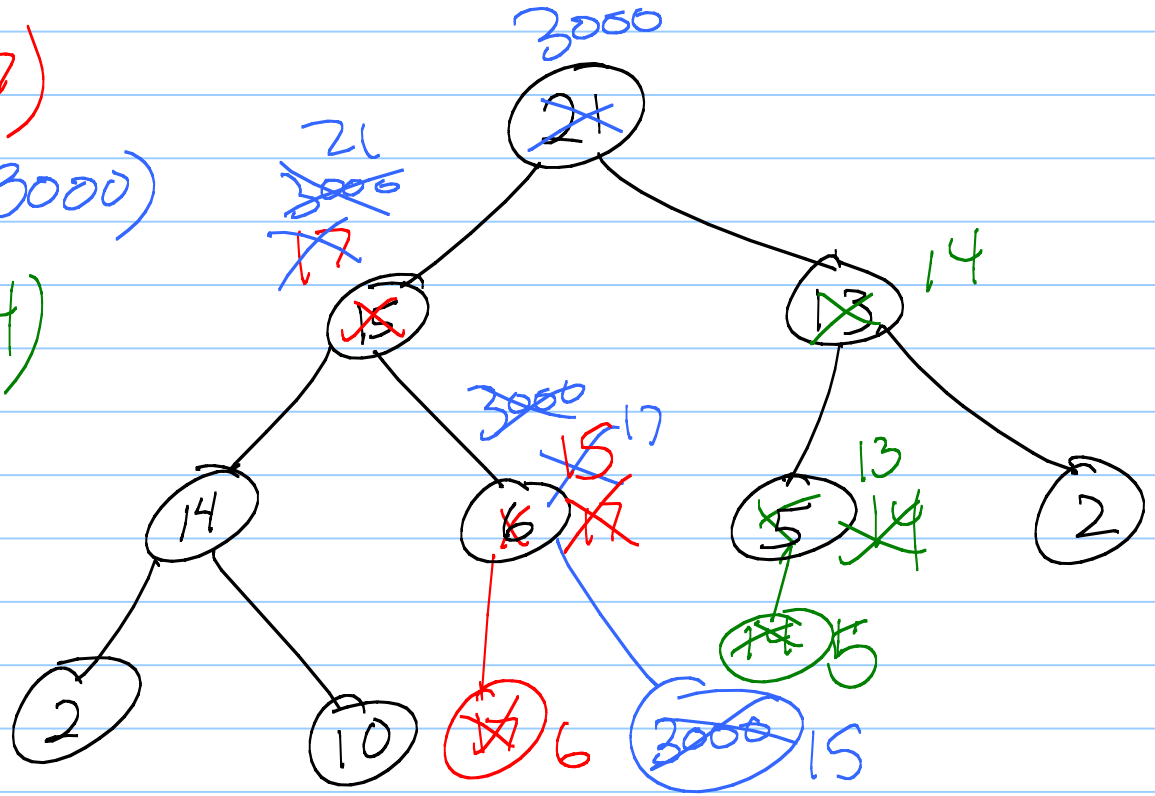


So: insert

insert(17)

insert(3000)

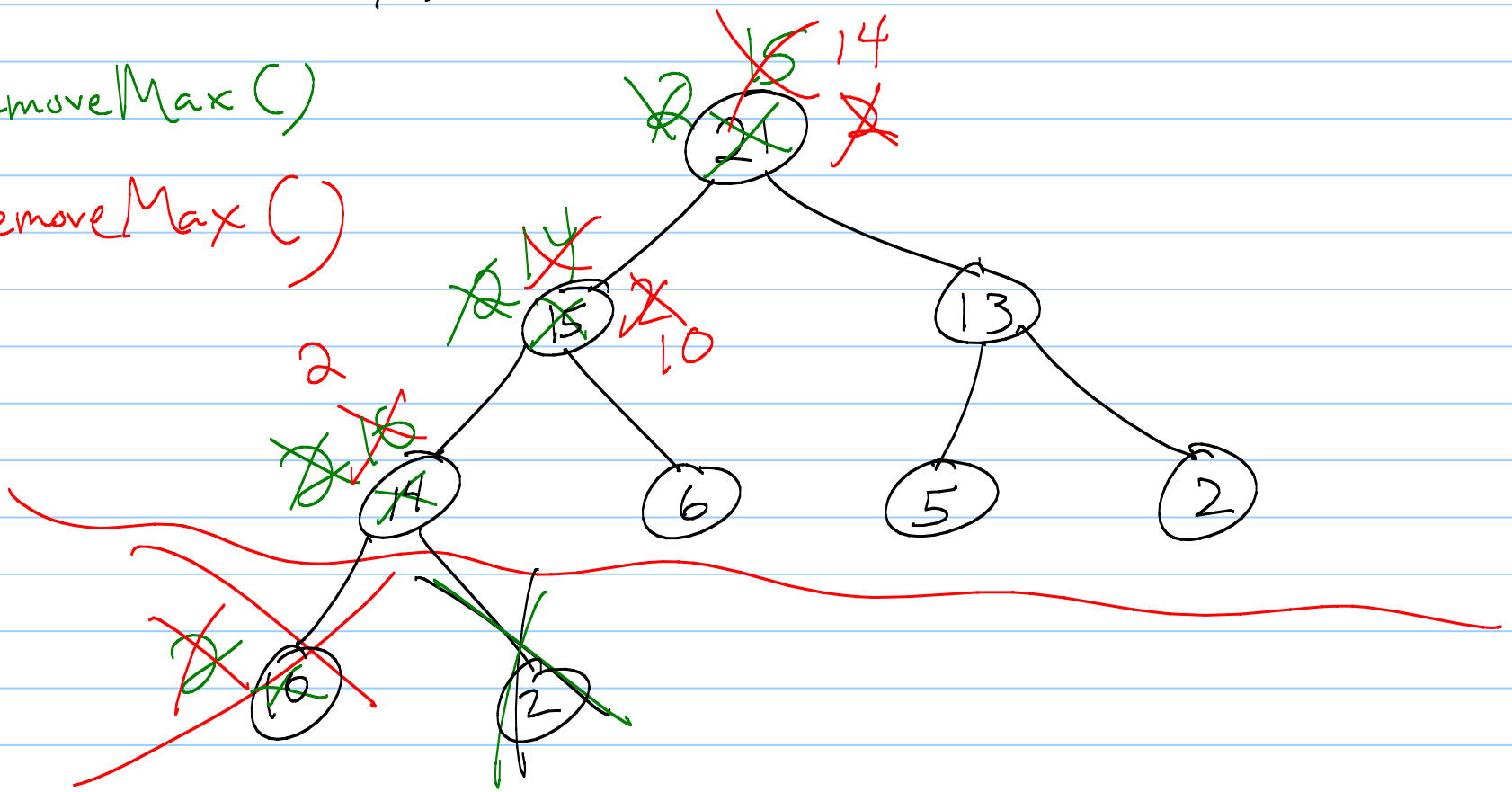
insert(14)



Remove Max:

removeMax()

removeMax()



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

$$\underbrace{2^0 + 2^1 + 2^2 + \ldots + 2^{h-1} + 2^h}_{2^{h+1} - 1} \geq n$$

$$2^{h+1} - 1 \geq n$$

$$2^0 + 2^1 + \ldots + 2^{h-1} \leq n$$

$$2^h - 1 \leq n$$

$$\Rightarrow h = \lceil \lg n \rceil$$

Running Times

Operation	Time
size, empty	$O(1)$
insert	$O(\log_2 n)$
removeMax	$O(\log_2 n)$
maxItem	$O(1)$

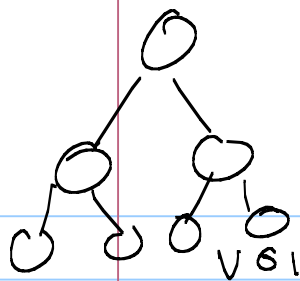
Now - some code

```
template <typename Itemtype>
class Heap {

  private:

    Itemtype* data;
    int size;
    int capacity;

  public:
    Heap () {
      size = 0;
      capacity = 1;
      data = new Itemtype[capacity];
    }
```



```
void insert(const ItemType& value) {
    if (size == capacity) {
        capacity = 2*capacity + 1;
        ItemType * newData = new ItemType[capacity];
        for (int i=0; i<size; i++)
            newData[i] = data[i];
        delete[] data;
        data = newData;
    }
    data[size] = value;
    size++;
```

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

while ( data[parent] < data[current] ) {
    ItemType temp = data[current];
    data[current] = data[parent];
    data[parent] = temp;
    current = parent;
    parent = (current-1)/2;
}

} //end of insert
```