CS 180 - More Trees + Heaps

Note Title 10/30/2009

Announcements

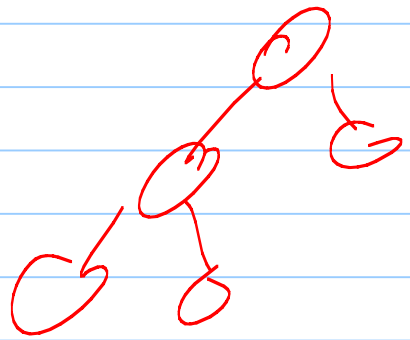
- HW due tomorrow
- Program 1 is graded
- Next program will be posted tomorrow

Trees: Traversals

How to display or check information stored in a tree?

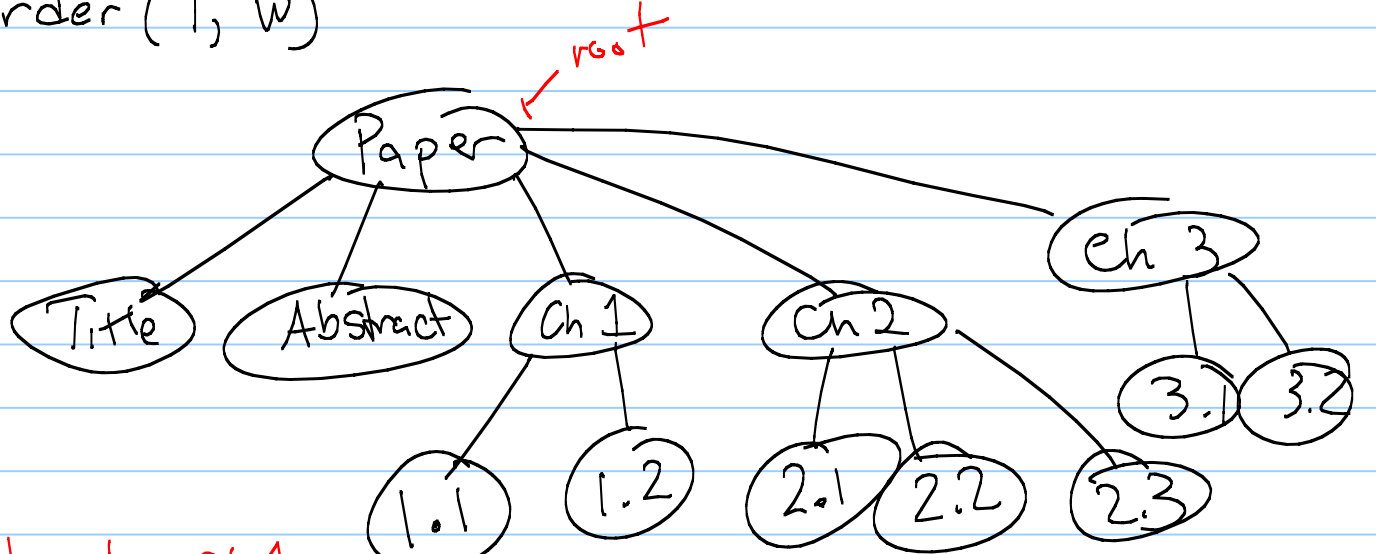
Different ways depending on what is stored in it.

- Inorder
- Preorder
- Postorder



Preorder (T, v):

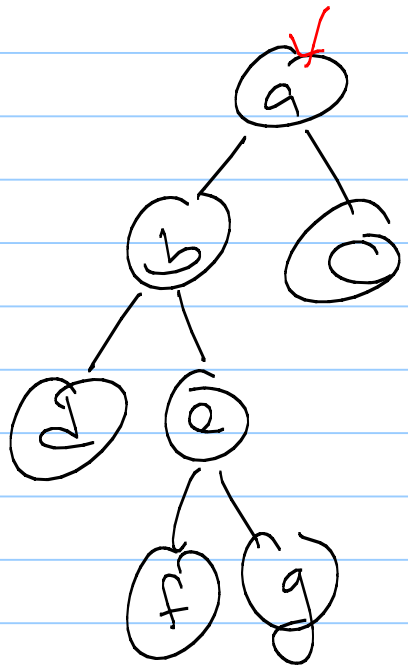
Perform "visit" at v
for each child w of v:
Preorder (T, w)



Paper, Title, Abstract, Ch1, 1.1, 1.2

Postorder (T, v):

```
for each child w of v:
    postorder (T, w)
perform action a v
```

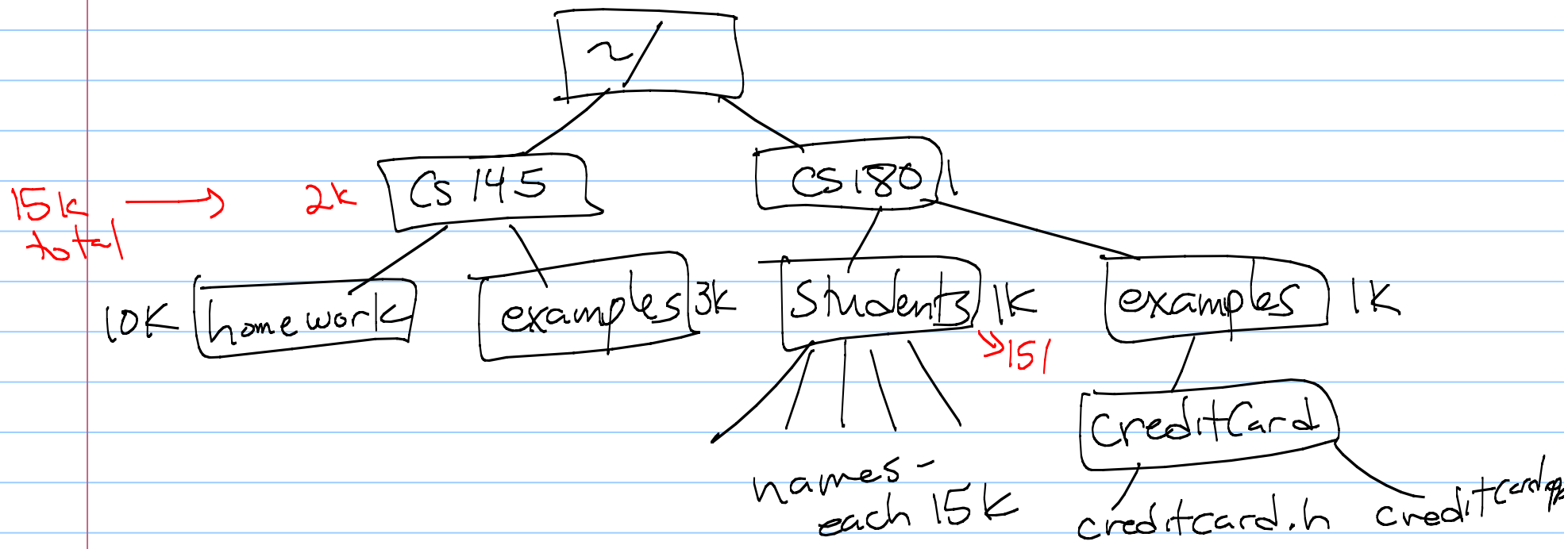


Print:

d, f, g, e, b, c, a

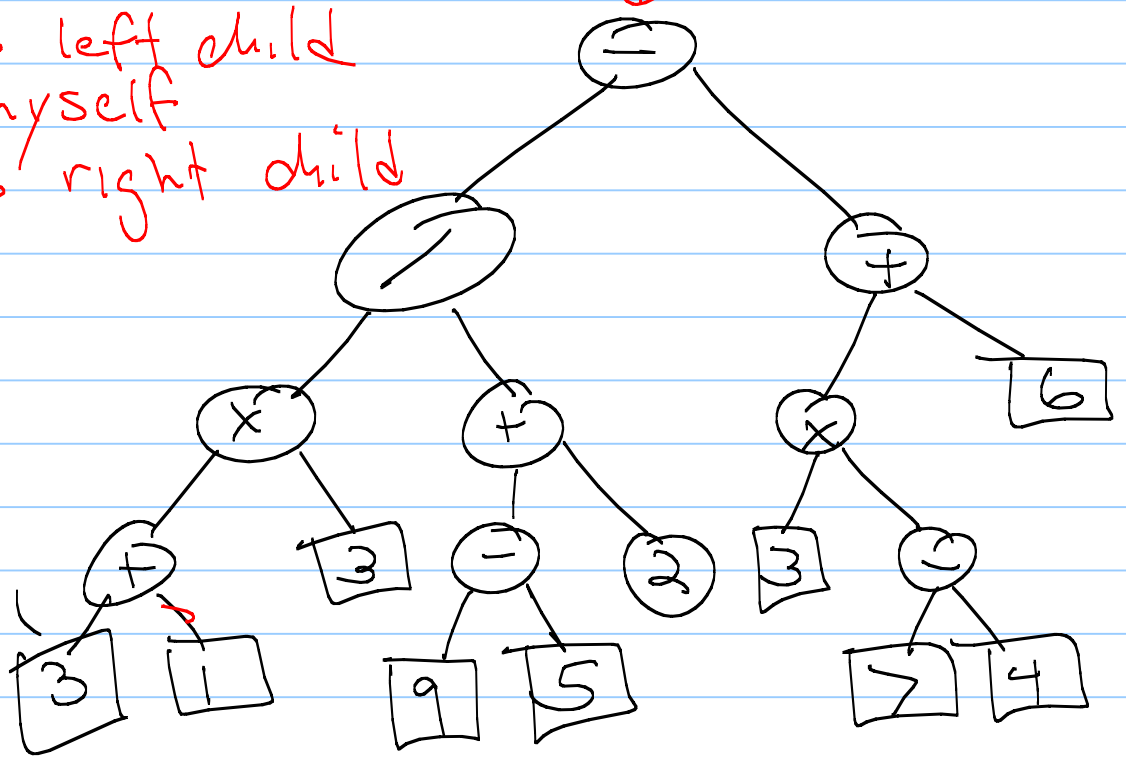
Example: size of a directory

- need to know size of children before we can compute current directory's size:



Inorder: ← only for binary trees

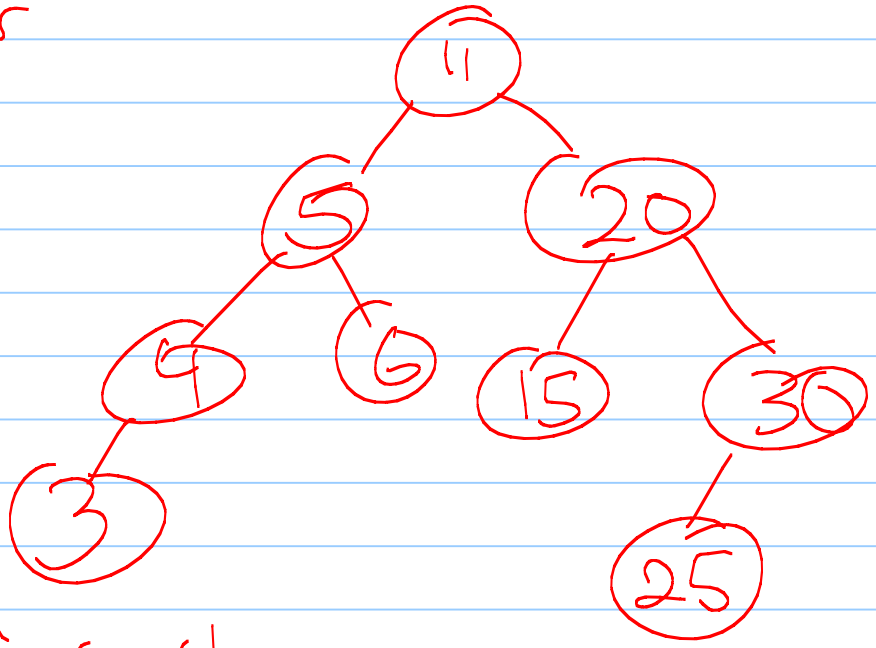
do left child
myself
do right child



((3 + 1) x 3) / ((9 - 5) + 2) -

Binary Search Trees

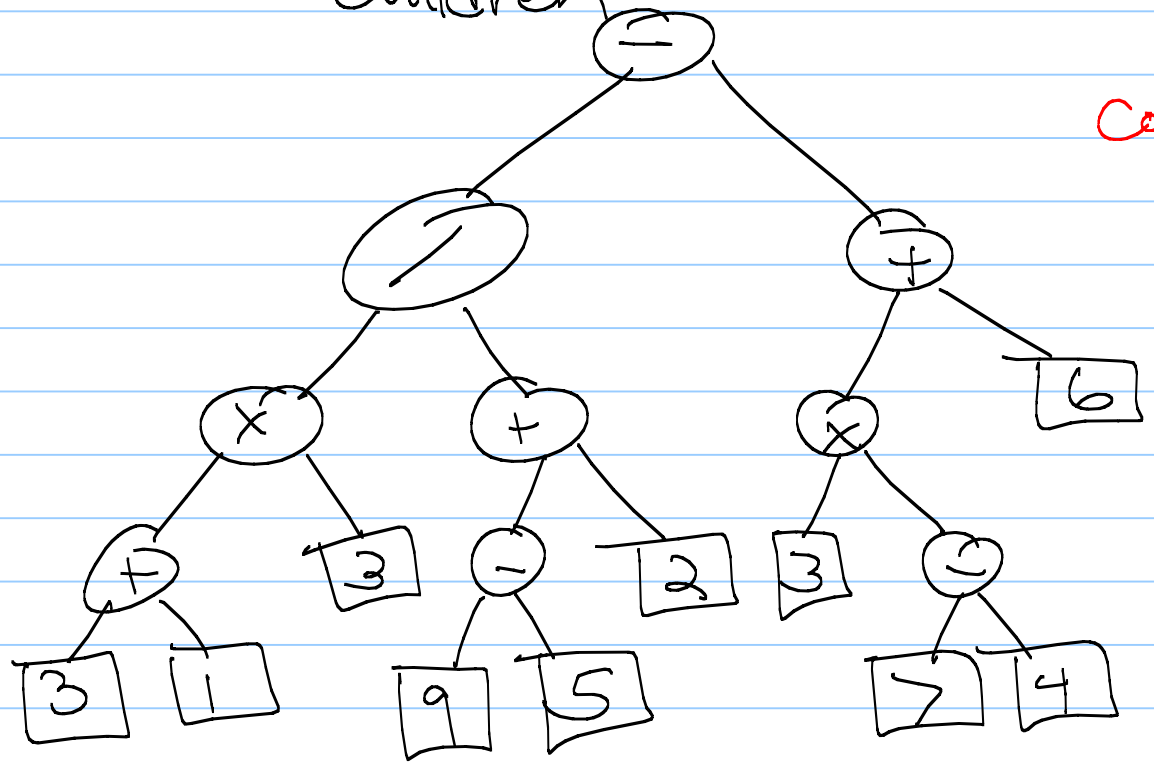
inorder



3, 4, 5, 6, 11

Binary Trees

- each internal node has exactly 2 children



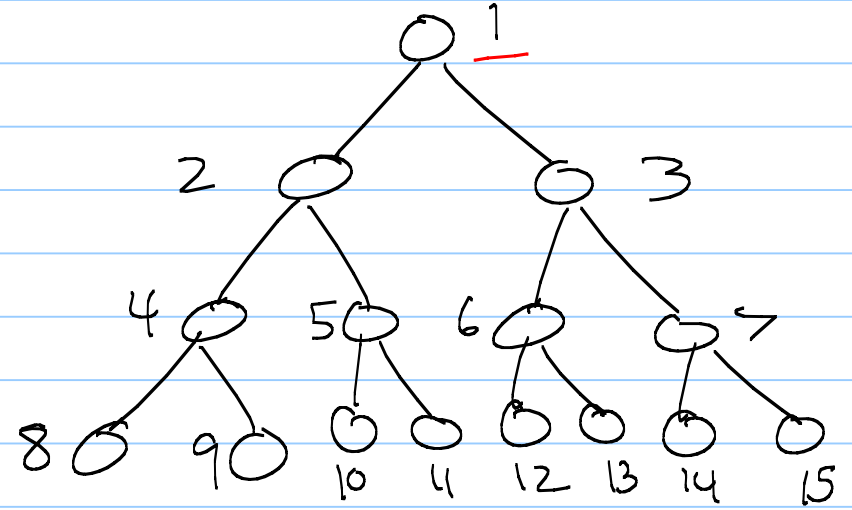
complete: all leaves have same depth

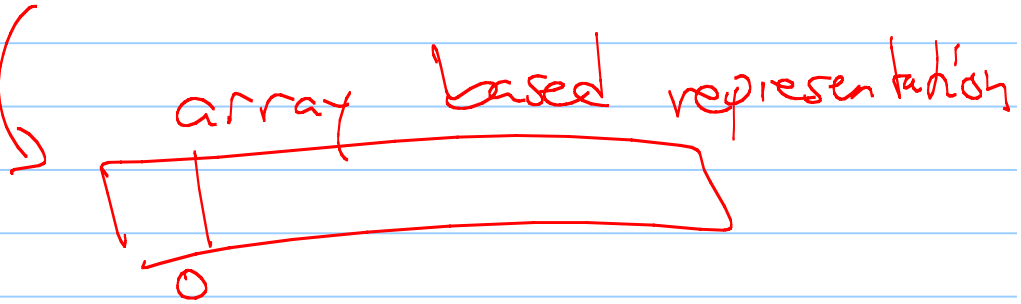
← not complete

Representations of binary trees:

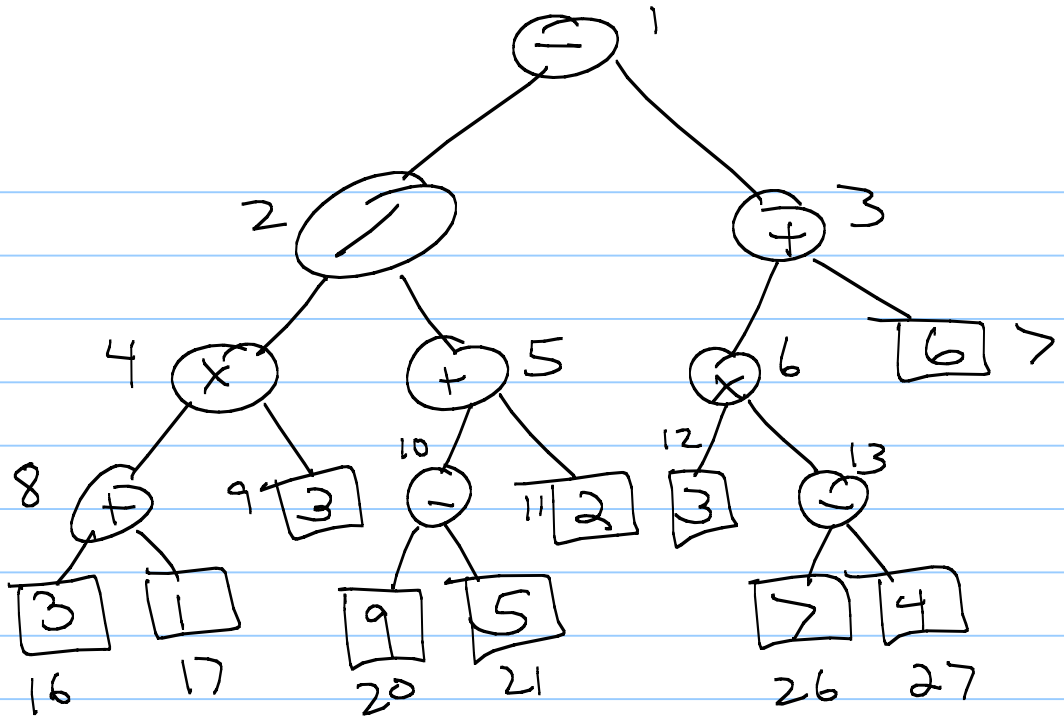
Level numbering

- If v is the root, $p(v) = 1$
- If v is left child of u, $p(v) = 2p(u)$
- If v is right child of u, $p(v) = 2p(u) + 1$





Ex:



1	2	3	4	5	6	7	8	9
-	/	+	x	+	x	6	+	3

Priority Queue ADT (Ch. 7)

Keys versus values

- Keys: sort based on these
- values: data stored

Ex: Standby list for a flight

values = names of people

key = calculated based on freq. flyer, order of request, + price

A note about keys:

Properties: need to be able to compare them

operator <=
operator ≤
operator ==

- reflexive property: $k \leq k$
- transitive property: if $k_1 \leq k_2$ and $k_2 \leq k_3 \Rightarrow k_1 \leq k_3$
- anti-symmetric: if $k_1 \leq k_2$ and $k_2 \leq k_1 \Rightarrow k_1 = k_2$

P. Q. ADT:

Methods:

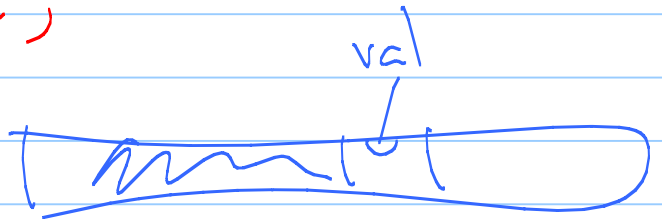
- insertItem(k, e)
- ~~min~~ max Element(): returns the element with the smallest key
- removeMin(): removes element with minimum key

Last time →

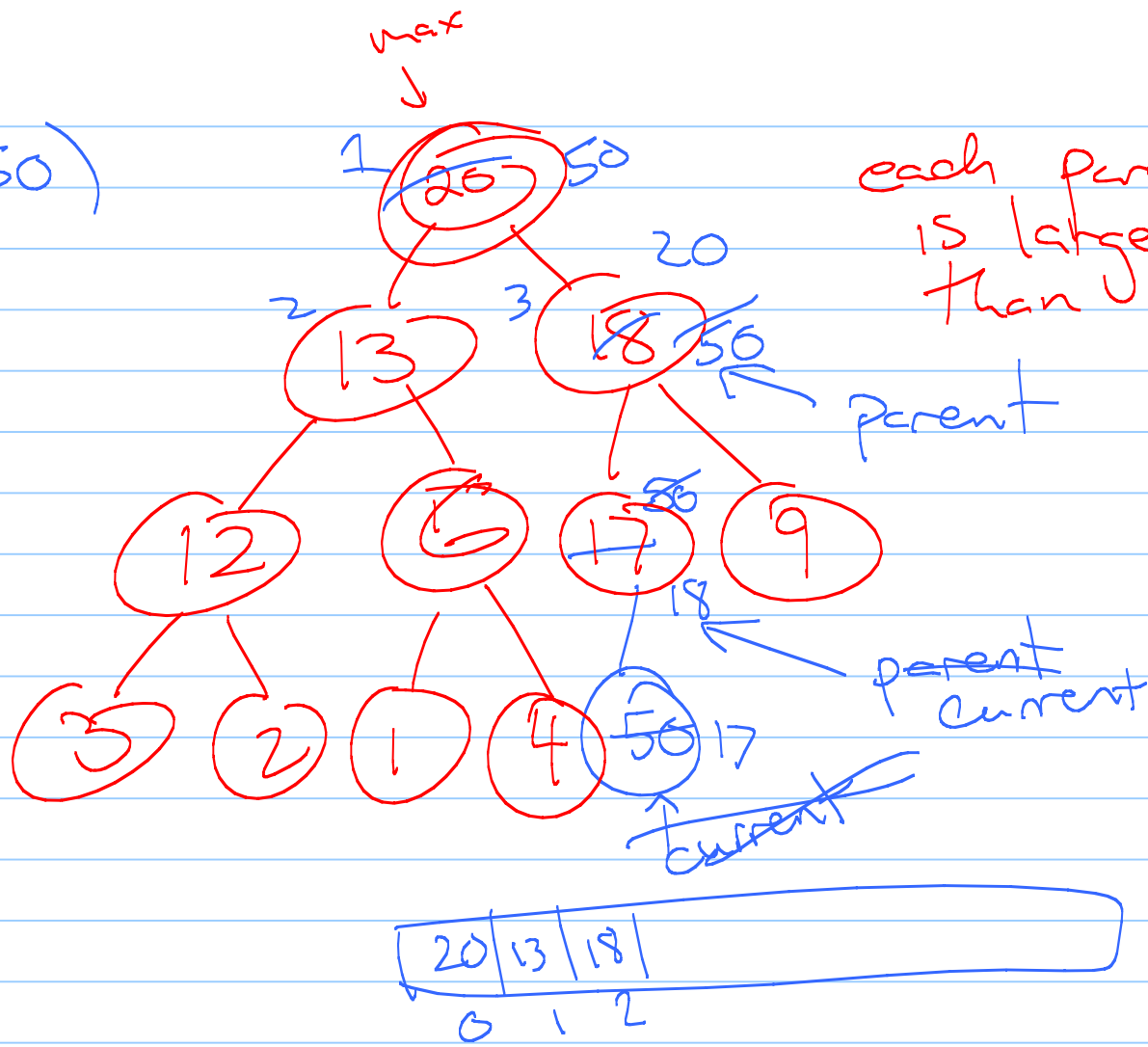
Code

```
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 * _capacity;
    ItemType* newdata = new ItemType[capacity];
    for (int i=0; i < _size; i++)
      newdata[i] = _data[i];
    delete data;
    _data = newdata;
  }
  _data[_size] = val;
  _size++;
```



insert(50)



each parent is larger than children

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

while (_data[current] > _data[parent] && current > 0) {

    ItemType temp(_data[current]);
    _data[current] = _data[parent];
    _data[parent] = temp;

    current = parent;
    parent = (current - 1) / 2;
}
}
```

removeMax();