Priority Queues - Moar Heaps

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Making priority queues out of heaps

Announcements

- Next MA should be out tonight.
- The leadership of Emsi is doing a presentation and student meeting session on Oct 17th, 4-5pm in Sloan 150 with pizza
  - https://www.facebook.com/events/849380565221322
- I’ve got a Heaps PA coming together, but we’ll see about timing after PA3

My brief Reddit fame:
A Linux Cake

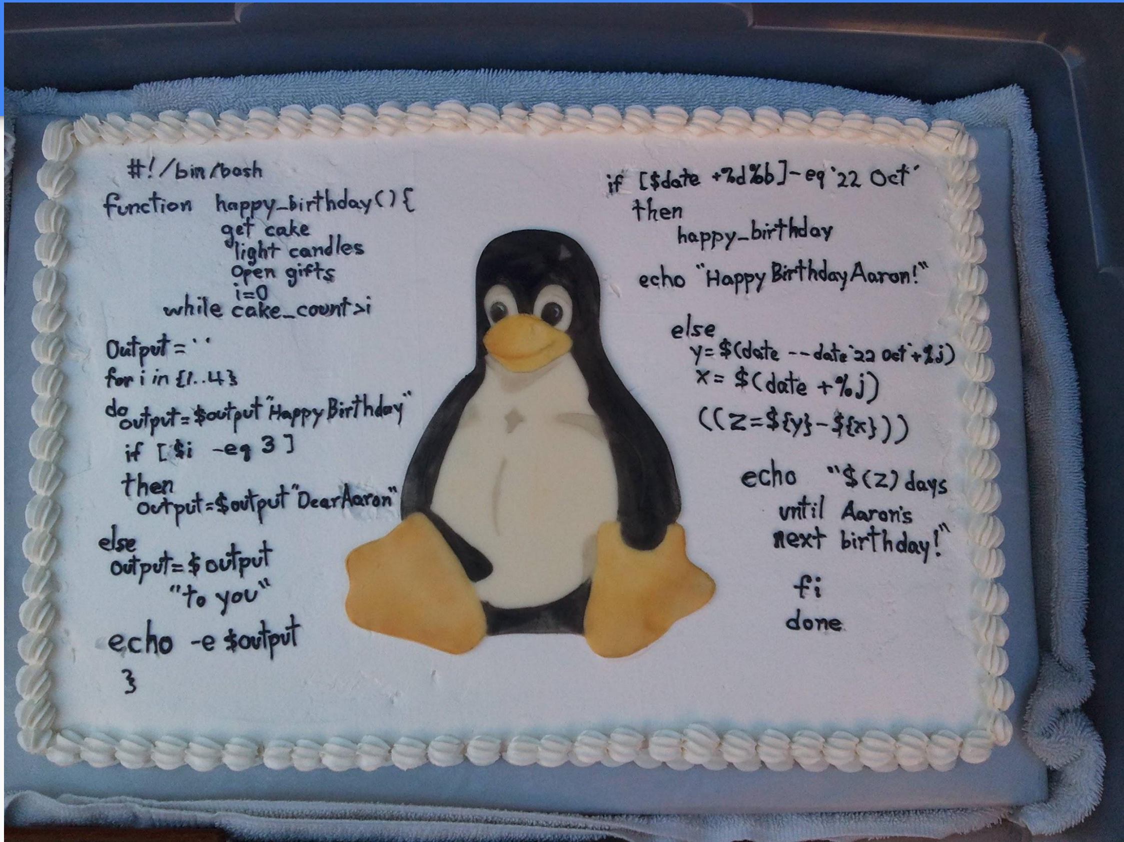
My birthday cake from a few years ago.

Posted on Reddit

4196 points, FP

The comments are geek humor galore.

Mom got it made at a baker’s in Moscow.



```
#!/bin/bash
function happy_birthday(){
                get cake
                light candles
                Open gifts
                i=0
        while cake_count>i

Output=''
for i in {1..4}
do output=$output"Happy Birthday"
  if [ $i -eq 3 ]
  then
    output=$output"DearAaron"
else
 output=$output
         "to you"
echo -e $output
 }
```

```
if [$date +%d%b]-eq'22 Oct'
  then
     happy_birthday
  echo "Happy BirthdayAaron!"

     else
       y=$(date --date'22 Oct'+%j)
       x= $(date +%j)
       ((z=${y}-${x}))

          echo "$(z) days
            until Aaron's
            next birthday!"
              fi
              done
```

Reminder: heap property

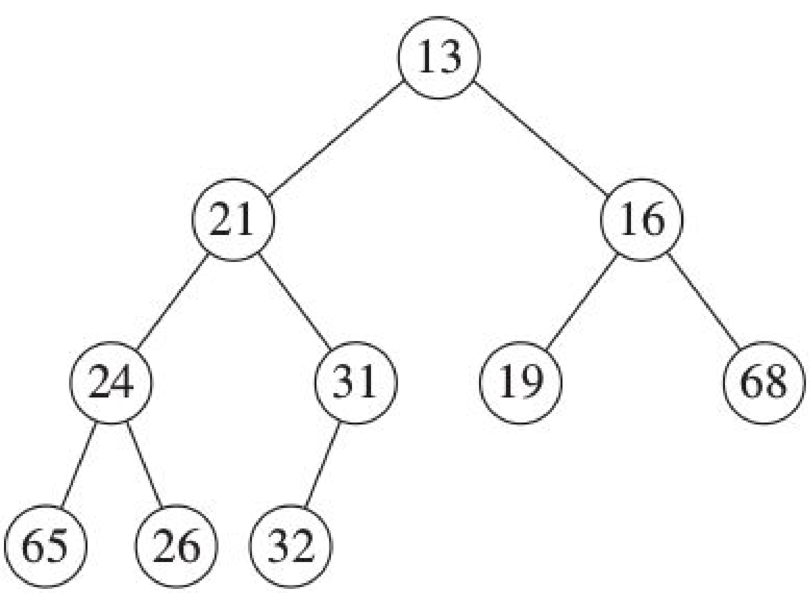
- Binary tree
- Parent key <= Children’s keys
- Height is log N
- Nodes stored in an array, not on The Heap and linked by pointers
    - Still drawn as trees to make operations clear

Remember! This is NOT a BST!

- Both children are larger than their parents in a Heap!
- Children are NOT sorted in any way
- Structure is NOT a BST

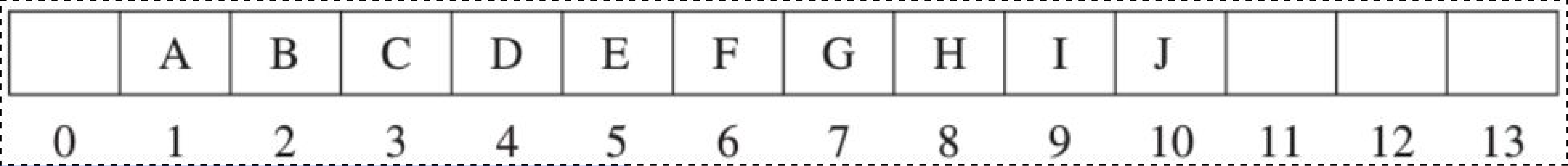
This is a valid Heap! ⇒

It's ***not*** a valid BST.



Heap-order property

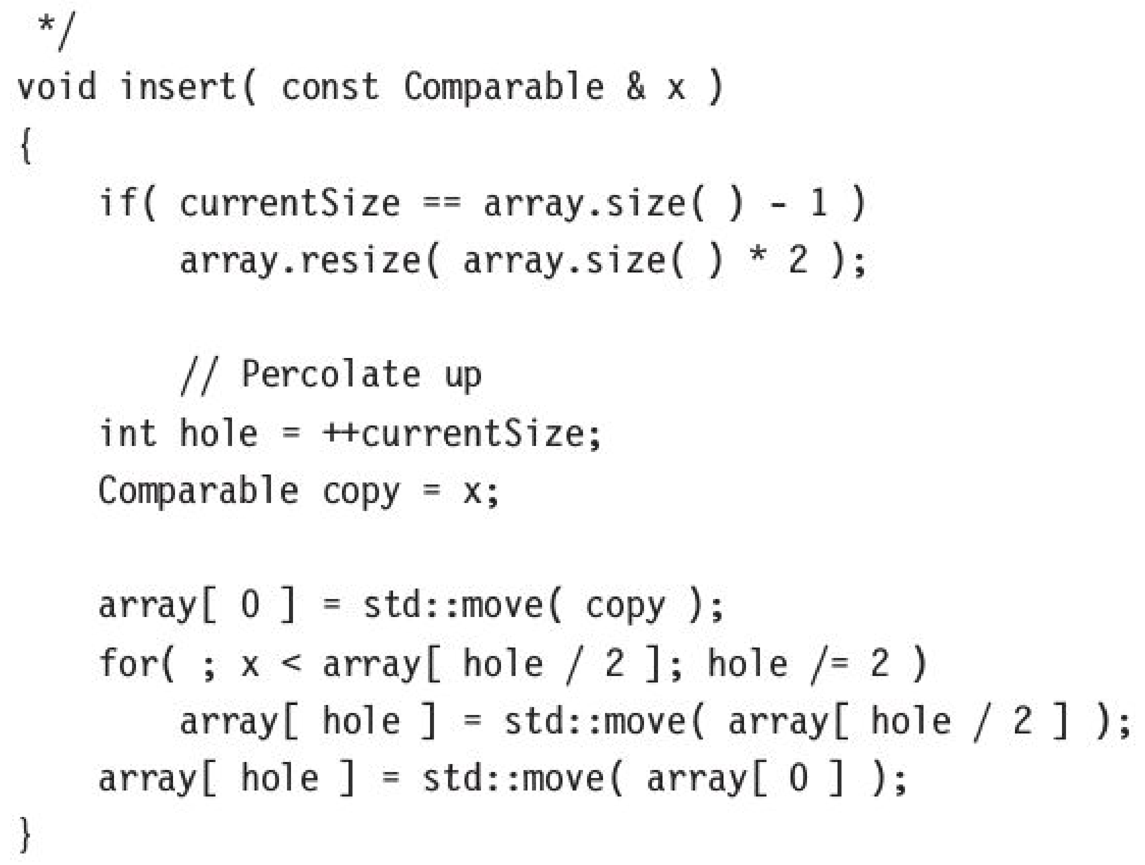
- Parents <= Children
- Root will always be smallest value
- findMin will always be in constant time



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

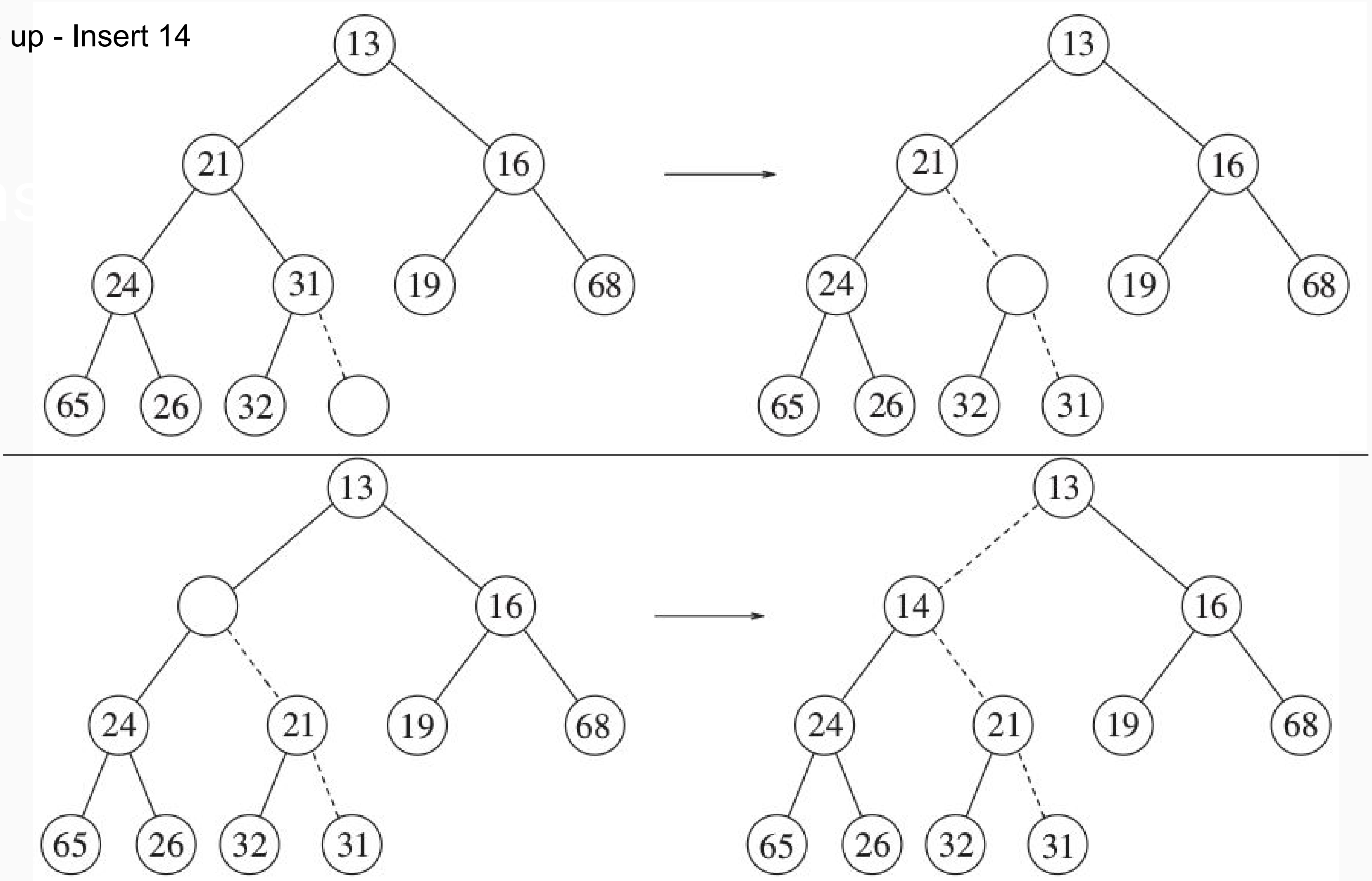
Short code!

- Move new item up tree until it fits heap-property



```
 */
void insert( const Comparable & x )
{
    if( currentSize == array.size( ) - 1 )
        array.resize( array.size( ) * 2 );

        // Percolate up
    int hole = ++currentSize;
    Comparable copy = x;

    array[ 0 ] = std::move( copy );
    for( ; x < array[ hole / 2 ]; hole /= 2 )
        array[ hole ] = std::move( array[ hole / 2 ] );
    array[ hole ] = std::move( array[ 0 ] );
}
```

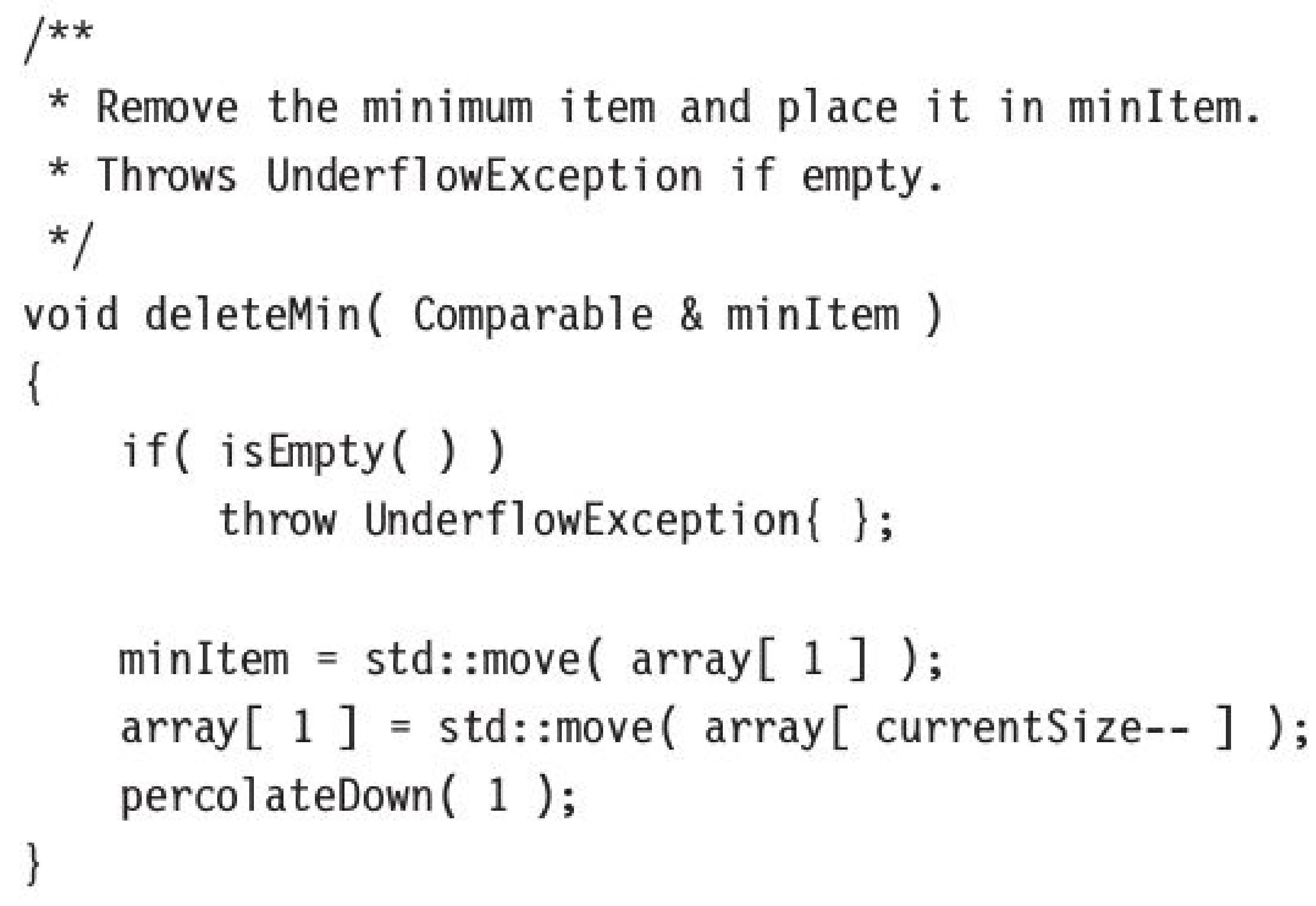
Percolate up - Insert 14



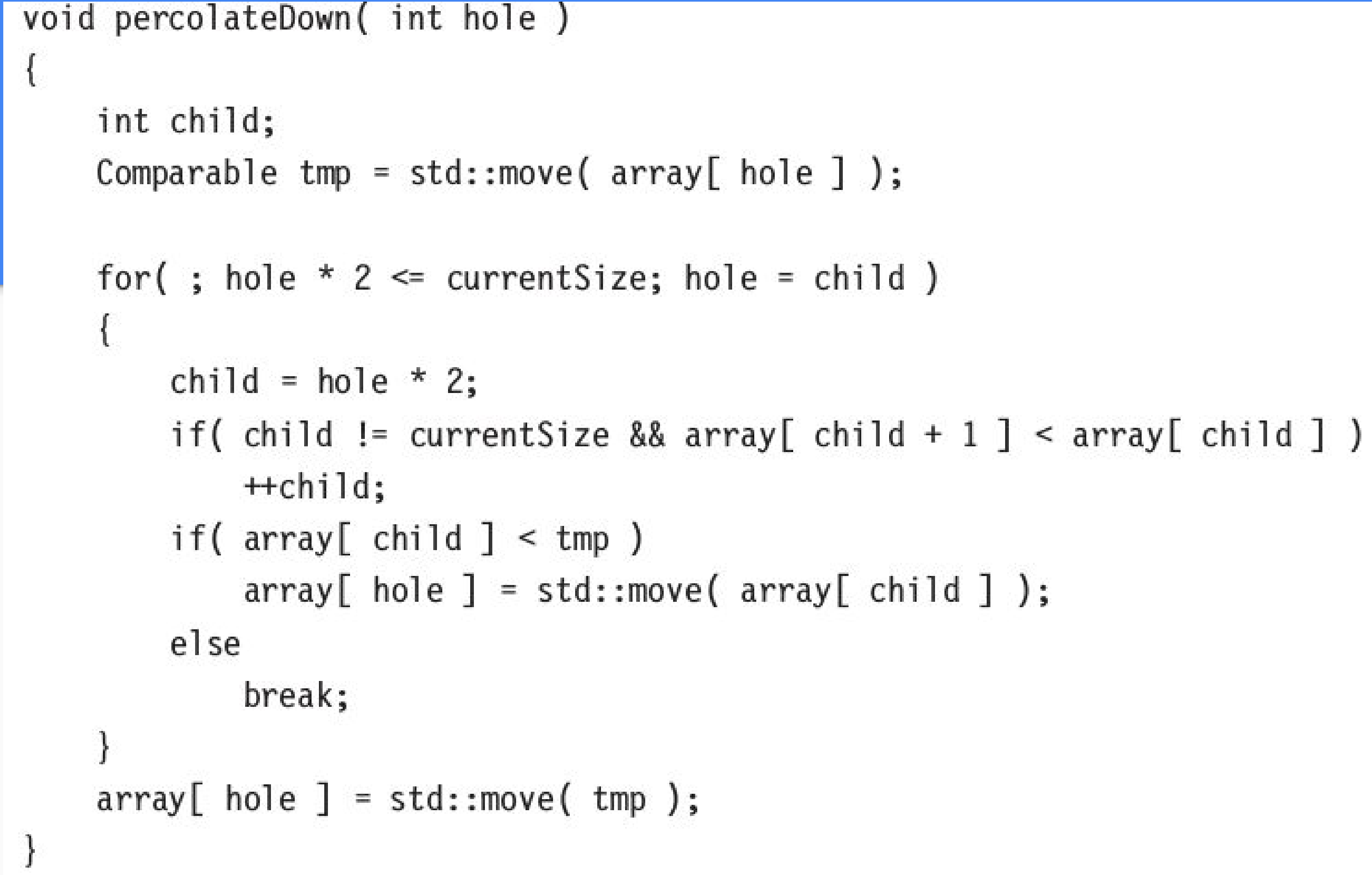
DeleteMin - Get lowest key and fix tree

- Move off the root of the tree for returning later
- Take hole at root of tree and fill it with the last item in the tree
    - This is the last filled element in our array
- Iteratively (or recursively) try to move the root of the tree down until it satisfies the heap-order property

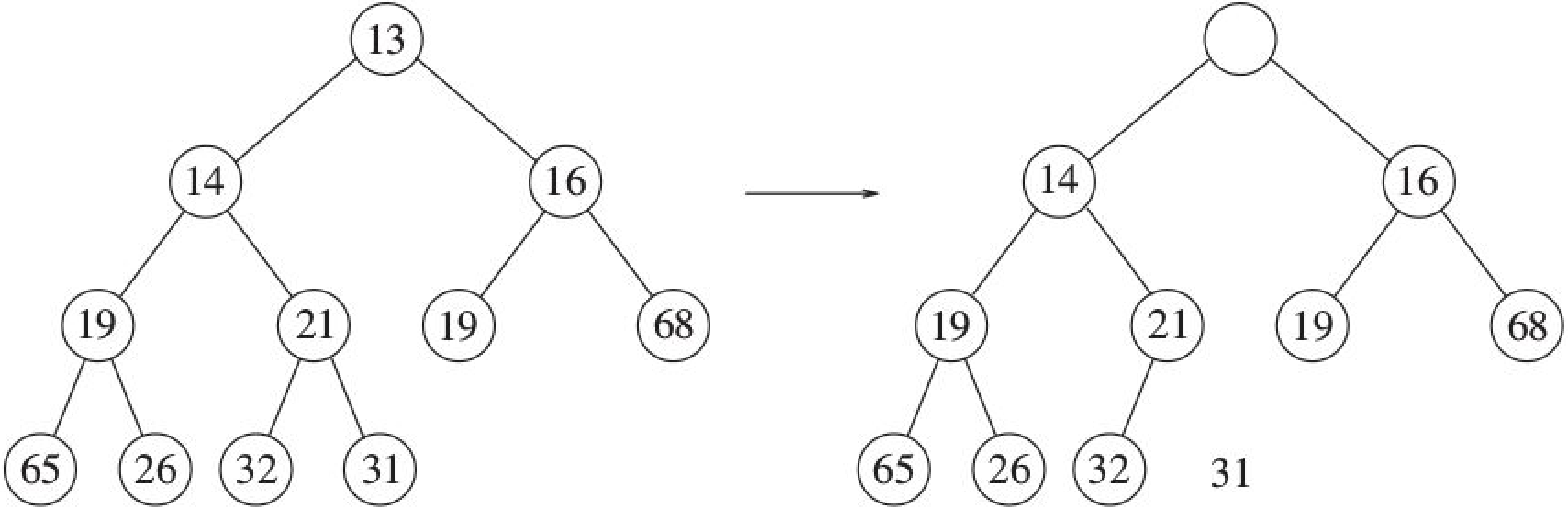
deleteMin

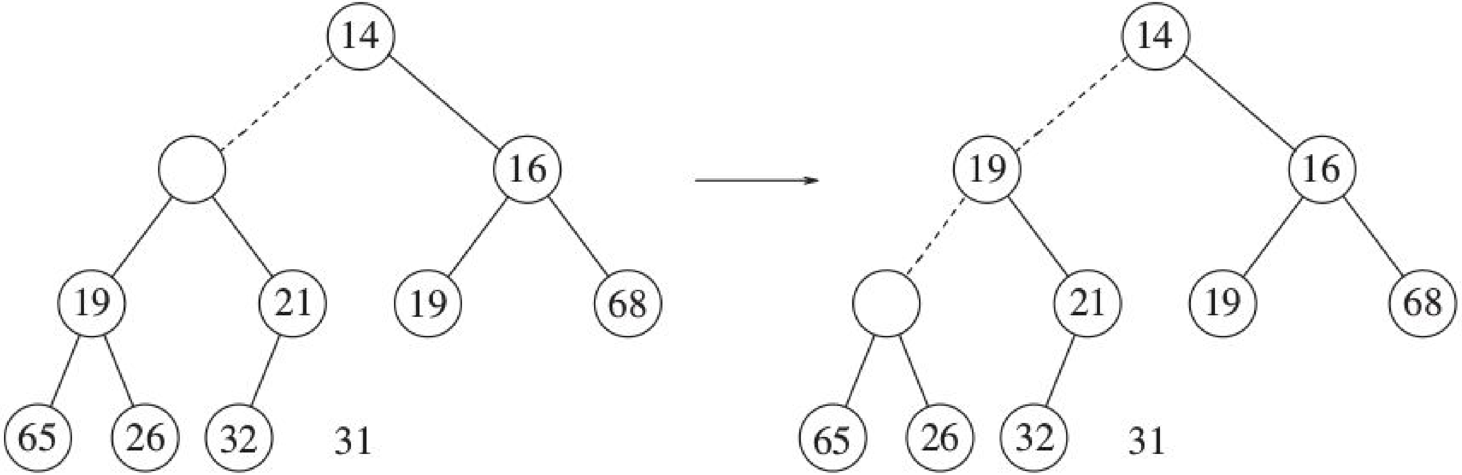


```
/**
 * Remove the minimum item and place it in minItem.
 * Throws UnderflowException if empty.
 */
void deleteMin( Comparable & minItem )
{
    if( isEmpty( ) )
        throw UnderflowException{ };

    minItem = std::move( array[ 1 ] );
    array[ 1 ] = std::move( array[ currentSize-- ] );
    percolateDown( 1 );
}
```

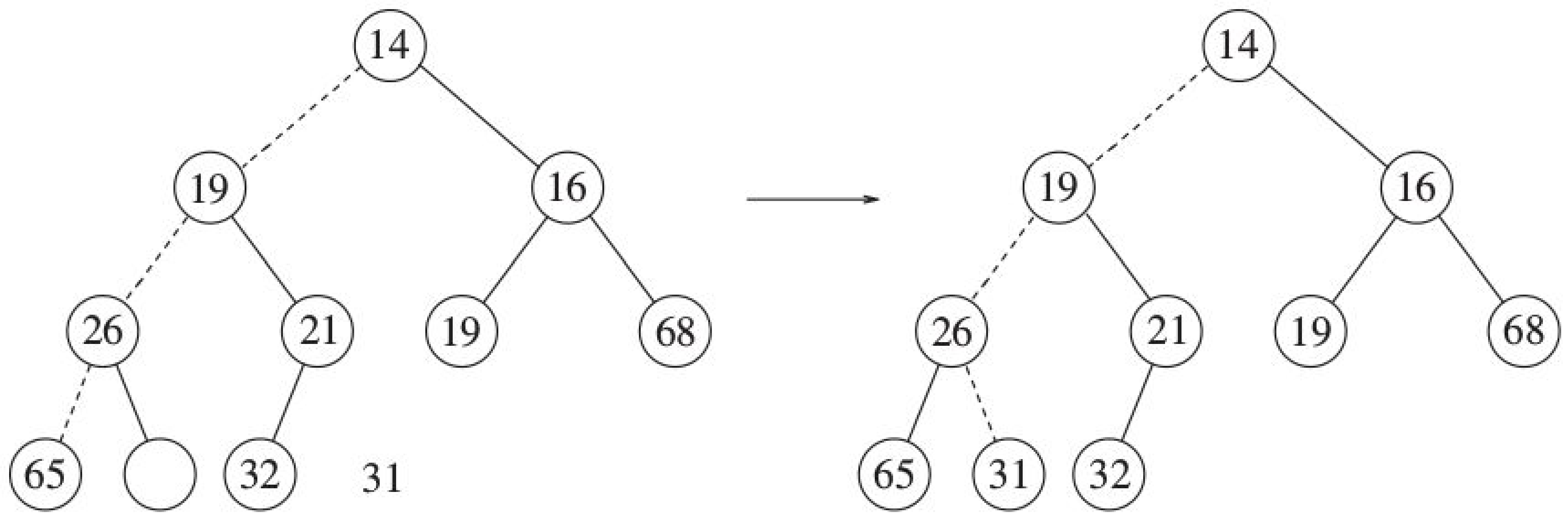


```
void percolateDown( int hole )
{
    int child;
    Comparable tmp = std::move( array[ hole ] );

    for( ; hole * 2 <= currentSize; hole = child )
    {
        child = hole * 2;
        if( child != currentSize && array[ child + 1 ] < array[ child ] )
            ++child;
        if( array[ child ] < tmp )
            array[ hole ] = std::move( array[ child ] );
        else
            break;
    }
    array[ hole ] = std::move( tmp );
}
```


13
14
16
19
21
19
68
65
26
32
31
14
16
19
21
19
68
65
26
32
31


14
16
19
21
19
68
65
26
32
31
14
19
16
21
19
68
65
26
32
31

Wrap up deleteMin



Other operations

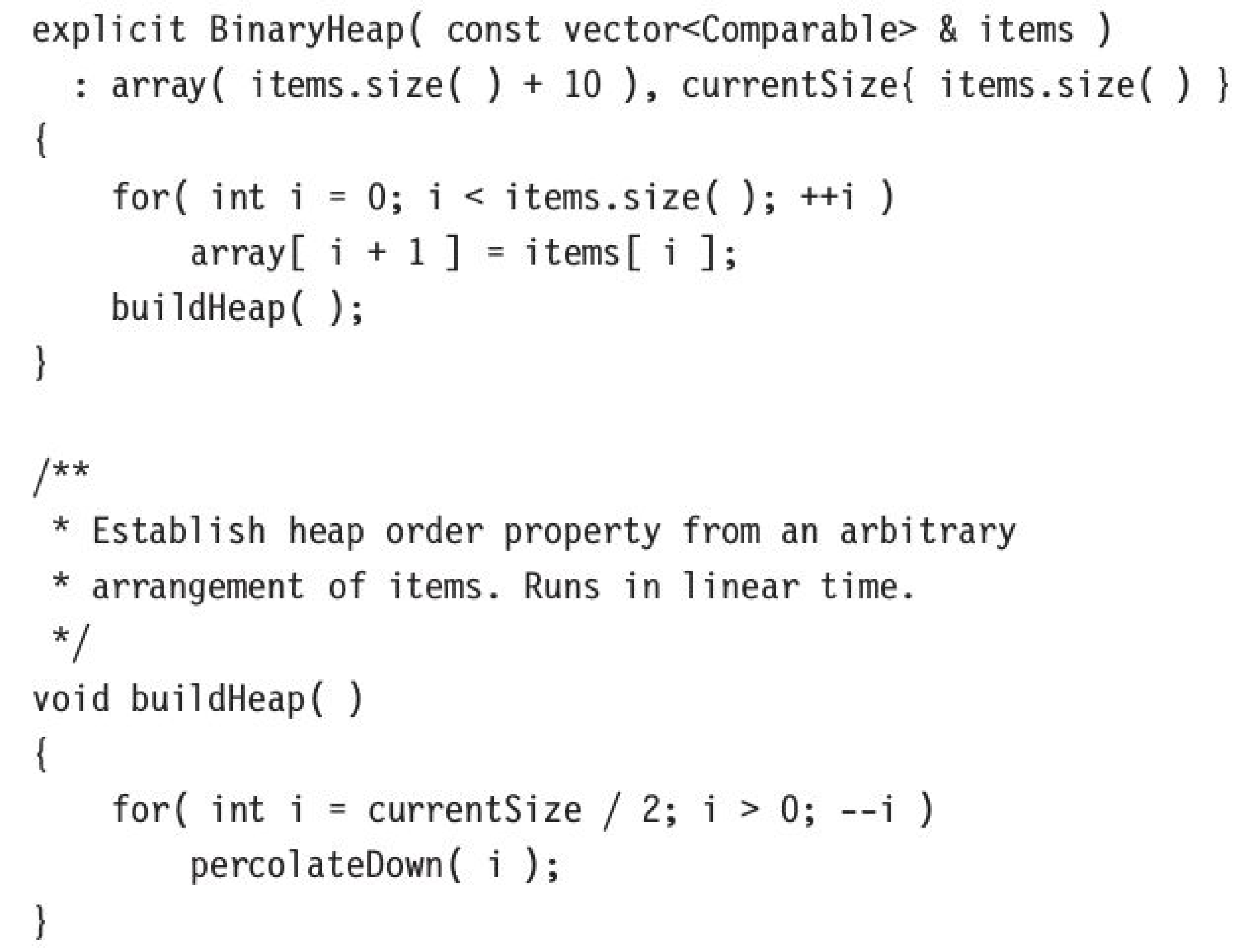
- decreaseKey(p, Δ)
  - Moves a node up the tree. Node @ p has it's key lowered by Δ. Then percolate Up(p)
  - So: heap[p].key -= Δ; percolateUp(p);
- increaseKey(p, Δ)
  - Moves a node down the tree. Node @ p has it's key raised by Δ. Then percolate Down(p)
  - So: heap[p].key += Δ; percolateDown(p);
- remove(p)
  - Removes node @ p. Set key @ p down by ∞. Percolate Up(p), then deleteMin()
  - So: decreaseKey(p, ∞); deleteMin();

Build Heap - O(N) creation of a heap!

- Actually quite simple:
  1) Start with an unordered list of nodes, based at 1 not 0 in array
  2) Start with (size of heap)/2, call percolateDown(), loop and decrement
- Inserts during build into array take O(1) average each,
  with O(log N) worst case
- Average time is: O(1) * N -> O(N)
- Worst case: O(log N) * N -> O(N log N)
- Building the heap gives us an average build of O(N) time!

Build Heap code

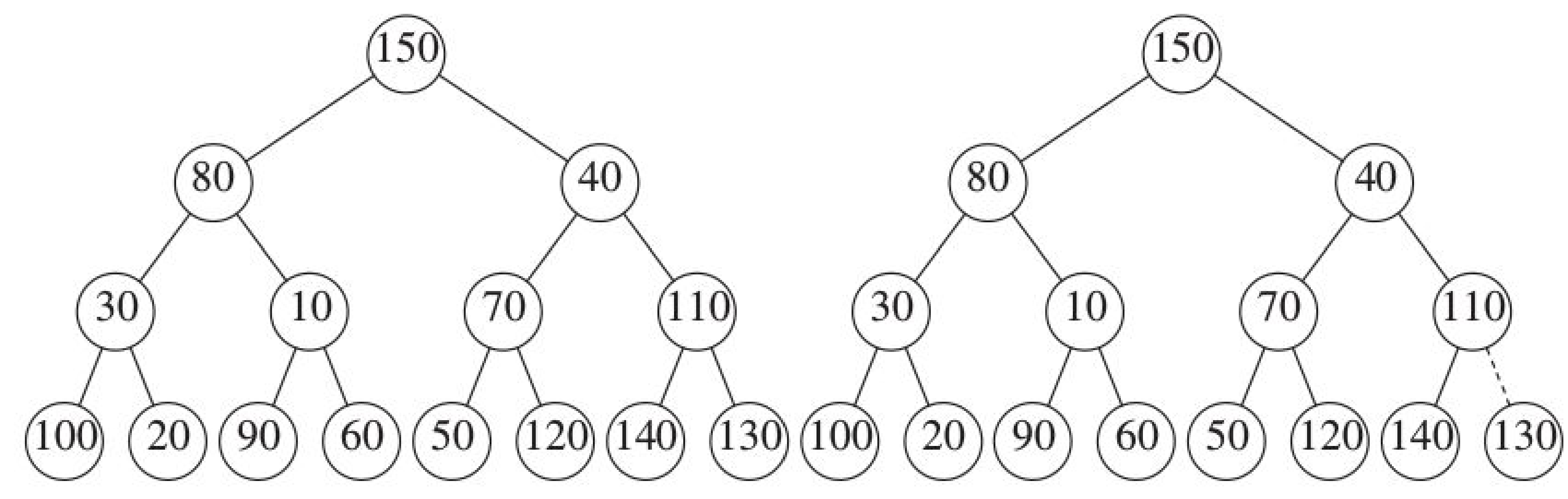
- Can merge two arrays?
- How long to do a merge?
    - MAGIC!

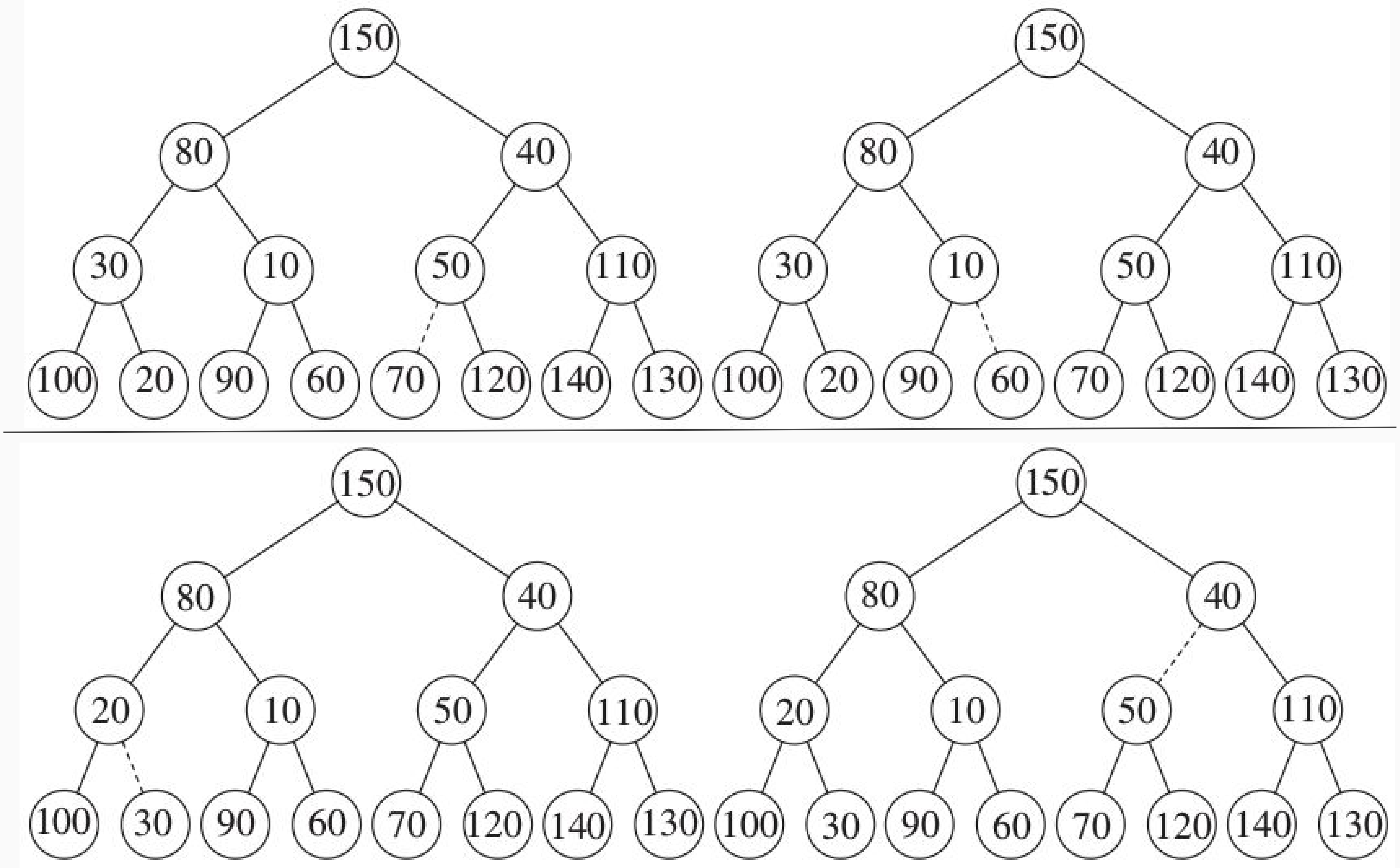


```
explicit BinaryHeap( const vector<Comparable> & items )
  : array( items.size( ) + 10 ), currentSize{ items.size( ) }
{
    for( int i = 0; i < items.size( ); ++i )
        array[ i + 1 ] = items[ i ];
    buildHeap( );
}


/**
 * Establish heap order property from an arbitrary
 * arrangement of items. Runs in linear time.
 */
void buildHeap( )
{
    for( int i = currentSize / 2; i > 0; --i )
        percolateDown( i );
}
```

Build Heap visually

Heap size = 15, so starting i = 15/2 = 7




150
80
40
30
10
50
110
100
20
90
60
70
120
140
130
150
80
40
30
10
50
110
100
20
90
60
70
120
140
130
150
80
40
20
10
50
110
100
30
90
60
70
120
140
130
150
80
40
20
10
50
110
100
30
90
60
70
120
140
130

VisuAlgo - great heap info

- VisuAlgo has some great heap examples

  https://visualgo.net/heap

For Monday

- Last of heaps
    - D-heaps
    - Leftist heaps
- Starting sorting