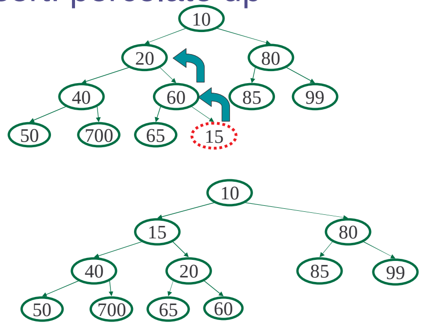
Heap – Insert(val)

Basic Idea:

1. Put val at “next” leaf position
2. Repeatedly exchange node with its parent if needed

Insert: percolate up



Insert pseudo/C++ Code (optimized)

```
void insert(Object o) {
  assert(!isFull());
  size++;
  newPos =
    percolateUp(size,o);
  Heap[newPos] = o;
}
```

```
int percolateUp(int hole,
                Object val) {
  while (hole > 1 &&
         val < Heap[hole/2])
    Heap[hole] = Heap[hole/2];
    hole /= 2;
  }
  return hole;
}
```

runtime:

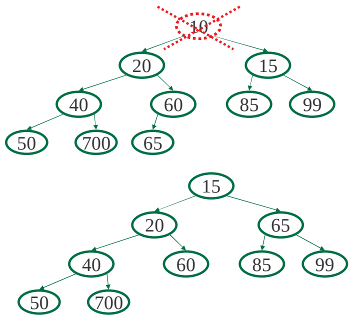
(Java code in book)

Heap – Deletemin

Basic Idea:

1. Remove root (that is always the min!)
2. Put “last” leaf node at root
3. Find smallest child of node
4. Swap node with its smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

DeleteMin: percolate down



DeleteMin pseudo/C++ Code (Optimized)

```
Object deleteMin() {
  assert(!isEmpty());
  returnVal = Heap[1];
  size--;
  newPos =
    percolateDown(1,
        Heap[size+1]);
  Heap[newPos] =
    Heap[size + 1];
  return returnVal;
}
```

```
int percolateDown(int hole,
                  Object val) {
while (2*hole <= size) {
    left = 2*hole;
    right = left + 1;
    if (right ≤ size &&
        Heap[right] < Heap[left])
      target = right;
    else
      target = left;

    if (Heap[target] < val) {
      Heap[hole] = Heap[target];
      hole = target;
    }
    else
      break;
  }
  return hole;
}
```

runtime:

(Java code in book)

Insert: 16, 32, 4, 69, 105, 43, 2

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Other Priority Queue Operations

- **decreaseKey(objPtr, amount)**
  - given a pointer to an object in the queue, reduce its priority value

Solution: change priority and ______________________________

- **increaseKey(objPtr, amount)**
  - given a pointer to an object in the queue, increase its priority value

Solution: change priority and _______________________________

Why do we need a *pointer*? Why not simply data value?

More Priority Queue Operations

- **Remove(objPtr)**
  - given a pointer to an object in the queue, remove it

Solution: set priority to negative infinity, percolate up to root and deleteMin

- **buildHeap**

Naïve solution:

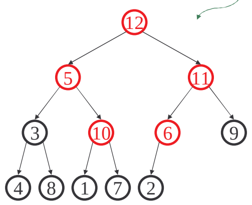
Running time:

Can we do better?

BuildHeap: Floyd's Method

12	5	11	3	10	6	9	4	8	1	7	2

Add elements arbitrarily to form a complete tree.
Pretend it's a heap and fix the heap-order property!

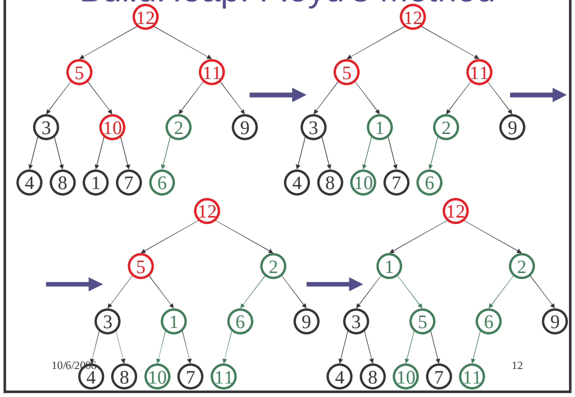


Buildheap pseudocode

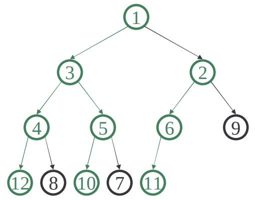
```
private void buildHeap() {
  for ( int i = currentSize/2; i > 0; i-- )
     percolateDown( i );
}
```

runtime:

BuildHeap: Floyd's Method



Finally…



runtime:

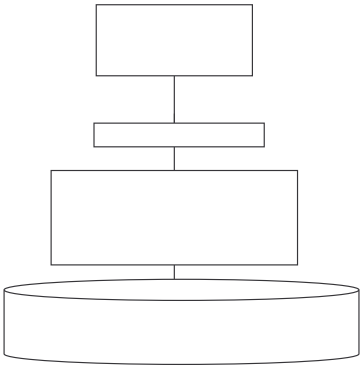
Facts about Heaps

Observations:

- finding a child/parent index is a multiply/divide by two
- operations jump widely through the heap
- each percolate step looks at only two new nodes
- inserts are at least as common as deleteMins

Realities:

- division/multiplication by powers of two are equally fast
- looking at only two new pieces of data: bad for cache!
- with huge data sets, disk accesses dominate



Cycles to access:

CPU

Cache

Memory

Disk