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)

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

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

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)

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

```cpp
int percolateDown(int hole, Object val) {
  int target = left;
  if (right <= size && Heap[right] < Heap[left])
    target = right;
  if (Heap[target] < val) {
    Heap[hole] = Heap[target];
    hole = target;
    if (Heap[target] > val) {
      returnVal = Heap[hole];
      Heap[hole] = Heap[target];
      hole = target;
    }
  }
  return hole;
}
```

runtime: (Java code in book)
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**
  
  Naive solution:
  
  Running time: Can we do better?

BuildHeap: Floyd's Method

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

Buildheap pseudocode

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

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