When Priority Values Change

Suppose one item in a Max-Heap changes its value. If the value increased, it may be higher than its parent. In which case, run:

```c
void fixUp(index j) { // index j of table away
    while (parent(j) exists & & table.at(j).value > table.at(parent(j)).value)
        swap(table.at(j), table.at(parent(j)));
    //ENS: table is a heap again, provided no other value changed!
}
```

If the value decreased, instead run:

```c
void fixDown(index j) { // if j is not in the bottom & &
    while (j is not in the bottom & & table.at(j).value is not >= both children)
        swap(j with the larger child)
    //ENS: table is a heap again, if no other changes.
}
```

Both routines run in \(O(\log n)\) time, and are just a re-conceptualization of the ideas for insert and pop:

```c
void insert(I & newIem) {
    table.at (firstFree++) = new Item;
    fixUp (firstFree - 1); // O(log n) time but often "lucky". Note n = firstFree
}
```

```c
I top() { //REQ: firstFree > 0
    return table.at(0);
}
```

```c
void pop() { // same REQ - test checks first, throws exception if violated.
    table.at(0) = table.at(-- firstFree);
    fixDown(0); // O(log n) time, really \(8\) since rarely "lucky."
}
```

The STL priority_queue class, part of <queue>, provides the above methods, but not a public fixUp or fixDown that can be used with an index. However, in <algorithm> the STL provides void make_heap (RA iterator first, RA iterator last, comp) which executes the following loop when \{first == 0 = table.height;

last = firstFree as in

\}