Quicker make_heap than looping over insert, which is all the text would say to do:

```cpp
void make_heap (with arguments O and firstFree)
for (int j = firstFree-1; j >= 0; j--) {
  fixed (table, at (j));
}
```

(not even in Ch.10 Heapsort)

**Fact (not in text; 😞)** This way of making an arbitrary vector into a heap runs in \(O(n)\) time, more precisely \(\Theta(n)\) time, with a pretty small constant. Hence better than:

```cpp
void fix PA (priority_queue &pq) {
  priority_queue other;
  while (!pq.empty()) {
    other, insert (pq, top(l));
    pq, pop();
  } // still need to swap or copy other back into pq!
}
```

**NOT literal code—see next page**

Takes \(\Theta(n \log n)\) time

Conclusion: STL <algorithm>'s make_heap provides an efficient way to "re-heapify" that works also when many priorities have changed. "Dynamic."

It also gives the speediest general method for compiling "Top K" lists from initially-unsorted data:

1. Put or Keep data in a vector, any order.
2. Call make_heap.
3. Pop (and save) K elements
4. Insert them back into heap, if needed

STL algorithm also has pop_heap and push_heap, which work with the same iterator args.

When \(n\) is the overall \# of elements, the time is

\[\Theta(n)\] for \(1\) & \(2\)

\[\Theta(K \log n)\] for \(3\) & \(4\).

When \(K\) is relatively small—formally, \(K = \Theta(n/\log n)\),

this is \(\Theta(n)\), and hardly beats the \(\Theta(n \log n)\)

You'd get if you'd fully sorted all the data, then read off top \(K\).

Importantly, STL's make_heap has a third parameter Comp (comp),

which can take any function object that does a "bool greater-than"

style comparison. This also is templated, but you can just call it!

Whereas literal code for "fix PA" would need a template header.