Quiz 05: Heaps Implementation

You will still be able to edit your answers up until the due date: **Jun. 23 12:59 AM**

Submit

```
# you should NOT modify the Heap implementation! (scroll down for merge)

class Heap:
    def __init__(self, keyfn=lambda x: x):
        self.data = []
        self.keyfn = keyfn

    @staticmethod
    def _left(idx):
        return idx*2+1

    @staticmethod
    def _right(idx):
        return idx*2+2

    @staticmethod
    def _parent(idx):
        return (idx-1)//2

    def add(self, x):
        self.data.append(x)
        idx = len(self.data)-1
        while idx > 0:
            pidx = Heap._parent(idx)
            if self.keyfn(self.data[idx]) > self.keyfn(self.data[pidx]):
                self.data[idx], self.data[pidx] = self.data[pidx], self.data[idx]
                idx = pidx
            else:
                break

    def max(self):
        assert len(self) > 0
        return self.data[0]

    def _heapify(self, idx):
        while idx < len(self):
            lidx = Heap._left(idx)
            ridx = Heap._right(idx)
            maxidx = idx
            if lidx < len(self) and self.keyfn(self.data[lidx]) > self.keyfn(self
```

```
                .data[maxidx]):
                maxidx = lidx
            if ridx < len(self) and self.keyfn(self.data[ridx]) > self.keyfn(self
                .data[maxidx]):
                maxidx = ridx
            if maxidx != idx:
                self.data[idx], self.data[maxidx] = self.data[maxidx], self
                    .data[idx]
                idx = maxidx
            else:
                break

    def pop_max(self):
        assert len(self) > 0
        rval = self.data[0]
        self.data[0] = self.data[-1]
        del self.data[-1]
        self._heapify(0)
        return rval

    def __bool__(self):
        return len(self.data) > 0

    def __len__(self):
        return len(self.data)

    def __repr__(self):
        return repr(self.data)

def merge(*lsts):
   merged = []
   heap = Heap()
   for i in range(len(lsts)):
       heap.add(lsts[i][-1])
   while len(heap) != 0:
       max_value = heap.pop_max()
       merged.append(max_value)
       for i in range(len(lsts)):
           if lsts[i][-1] == max_value:
               del lsts[i][-1]
               if len(lsts[i]) != 0:
                   heap.add(lsts[i][-1])
               break
   merged.reverse()
   return merged
```