Assignment 3

> Author: Isaac Archer
>
> Due Date: July 21, 2015 13:00

Part 1

> Description: MULTIPLE-ELEMENT HEAP (MEH) EXTRACTION
>
> Problem Description:

```
< H; ⊑: E x E; new: P(E),
                    insert: E x H → H,
                    extract-best H → (E, H),
                    empty: H → B,
                    extract-best-m (H, m) → (a, H) >

extract-best-m (H, m) → (a, H) | a is a set of Elements | ∃ m ≤ |H|, |a| = m
```

Brute Force Algorithm Provided

```
def brute_force
      a ← m
      for i ← 1..|m|
            (a[i], H) ← extract-best(H)

      ↑ a, H
```

Algorithm: Multiple Element Heap Extraction

> input: Priority Heap H of n Elements. An extraction count m ≤ n
>
> output: An array containing the best m elements extracted from H, and modified H.

```
def extract_best_m ← H, m

      a ← []

      if m > |H| then ↑ a, H

      for i ← 1..m
            R ← H.root

            remove H.root
```

```
            H.root ← rightmost bottom element of H

            downheap ← H.root

            // append R to a
            append R → a

      ↑ a, H
```

> input: Root Node in a Priority Heap
>
> output: None, it does an in place re-ordering

```
def downheap ← R

      if is_leaf ← R then ↑

      if R ⊑ R.left and R ⊑ R.right then ↑

      R.good ← better_of ← R.left, R.right

      swap ← R, R.good

      downheap ← R.good
```

> input: Node in a Priority Heap
>
> output: True if the input is a leaf and False if it is not

```
def is_leaf ← R
      if not R.left and not R.right then ↑ true

      ↑ false
```

> input: Two Nodes in a Priority Heap
>
> output: The better option of the Two Nodes

```
def better_of ← a, b
      if a ⊑ b then ↑ a

      if b ⊑ a then ↑ b

      ↑ a
```

Correctness

> - Returning the Correct Value
>
> > The algorithm pulls the topmost element, or best, from H and then reorders H each time. So it cannot re-organize H incorrectly as downheap always pushes elements back down into the correct sorted location, however it does suffer a performance drop which can be solved by grabbing the best m elements and then re-ordering H. however re-ordering becomes more complicated.
>
> - Halting
>
> > downheap is the only function that is recursive that has to meet some requirement to return up and it has a termination condition that will always check when there is no more of H to shuffle down assuming a worst case for relocating R. if H is finite and $m \le n$ then the number of times this can be run is finite, ans since each version will always halt because H is finite, the algorithm will halt.

Complexity

> Since my algorithm does not attempt to pull multiple elements and then reorder it will perform the same as the brute force algorithm and that complexity is covered below.

Show Worst Case of BRUTE-FORCE MEH is O(m lg n)

> since the number of iterations is m; and the worst case for extract-best is lg n due to it being a binary tree implementation of a Heap assuming $|H| = n$; we get m iterations of lg n or $O(m \lg n)$.

Part 2

Worst Case for Merge Sort is

```
T[0] = 1
T[1] = 1
T[n] = T[n/2] + T[n/2] + O(n)
```

Master Theorem with this Recurrence