Chapter 6: Heapsort

Here is the psuedocode assuming arrays start at index 0. So an n element array has indices $0,\ldots,n-1$.

```
PARENT(i)
return (i - 1)/2 // integer division

LEFT(i)
return 2i + 1

RIGHT(i)
return 2i + 2

MAX-HEAPIFY(A, i) // heapification downward
   Pre-condition: Both the left and right subtrees of node i are max-heaps
                    and i is less than or equal to heap-size[A]
   Post-condition: The subtree rooted at node i is a max-heap
l = LEFT(i)
r = RIGHT(i)
if l < A.heap-size and A[l] > A[i]
   largest = l
else largest = i
if r < A.heap-size and A[r] > A[largest]
   largest = r
if largest != i
   exchange A[i] with A[largest]
   MAX-HEAPIFY(A, largest)

BUILD-MAX-HEAP(A)
//A[0:n-1] is an unsorted array
A.heap-size = n
for i = n/2 - 1 downto 0  //skip the leaves
    do MAX-HEAPIFY(A, i)

HEAPSORT(A)
// array A[0:n-1] is unsorted
BUILD-MAX-HEAP(A)
for i = n-1 downto 2
    do exchange A[1] with A[i]
       A.heap-size = A.heap-size - 1
       MAX-HEAPIFY(A, 0)
```

```
MAX-HEAP-MAXIMUM(A)
//O(1) time
1. if A.heap-size < 1
2.    error "heap underflow"
3. return A[0]

MAX-HEAP-EXTRACT-MAX(A)
//O(lg n) time
1. max = MAX-HEAP-MAXIMUM(A)
2. A[0] = A[A.heap-size - 1]
3. A.heap-size = A.heap-size - 1
4. MAX-HEAPIFY(A, 0)
5. return max

MAX-HEAP-INCREASE-KEY(A, x, key)
//O(lg n) time
1. if key < x.key
2.    then error "new key must be larger than current key"
3. x.key = key
4. find the index i in array A where object x resides
5. while i > 0 and A[PARENT(i)].key < A[i].key
6.       exchange A[i] and A[PARENT(i)] (and update the object to index map)
7.       i = PARENT(i)

MAX-MAX-HEAP-INSERT(A, x, n)
//O(log n) time
1. if A.heap-size == n
2.    error "heap overflow"
3. A.heap-size = A.heap-size + 1
4. k = x.key
5. x.key = -infinity //Integer.MIN_VALUE, for example
6. A[heap-size - 1] = x
7. map x to index heap-size - 1 in the array
8. MAX-HEAP-INCREASE-KEY(A, x, k)
```