Heap Algorithms

PARENT(A, i)

// *Input*: A: an array representing a heap, i: an array index
// *Output*: The index in A of the parent of i
// *Running Time*: $O(1)$
if $i == 1$ **return** NULL
return $\lfloor i/2 \rfloor$

LEFT(A, i)

// *Input*: A: an array representing a heap, i: an array index
// *Output*: The index in A of the left child of i
// *Running Time*: $O(1)$
if $2 * i \leq$ *heap-size*$[A]$
    return $2 * i$
else return NULL

RIGHT(A, i)

// *Input*: A: an array representing a heap, i: an array index
// *Output*: The index in A of the right child of i
// *Running Time*: $O(1)$
if $2 * i + 1 \leq$ *heap-size*$[A]$
    return $2 * i + 1$
else return NULL

MAX-HEAPIFY(A, i)

// *Input*: A: an array where the left and right children of i root heaps (but i may not), i: an array index
// *Output*: A modified so that i roots a heap
// *Running Time*: $O(\log n)$ where $n =$ *heap-size*$[A] - i$
$l \leftarrow$ LEFT(i)
$r \leftarrow$ RIGHT(i)
if $l \leq$ *heap-size*$[A]$ and $A[l] > A[i]$
    largest $\leftarrow l$
else *largest* $\leftarrow i$
if $r \leq$ *heap-size*$[A]$ and $A[r] < A[largest]$
    largest $\leftarrow r$
if *largest* $\neq i$
    exchange $A[i]$ and $A[largest]$
    MAX-HEAPIFY(A, LARGEST)

BUILD-MAX-HEAP(A)

// *Input*: A: an (unsorted) array
// *Output*: A modified to represent a heap.
// *Running Time*: $O(n)$ where $n =$ *length*$[A]$
heap-size$[A] \leftarrow$ *length*$[A]$
for $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1
    MAX-HEAPIFY(A, i)

```
HEAP-INCREASE-KEY(A, i, key)
  // Input: A: an array representing a heap, i: an array index, key: a new key greater than A[i]
  // Output: A still representing a heap where the key of A[i] was increased to key
  // Running Time: O(log n) where n = heap-size[A]
if key < A[i]
    error("New key must be larger than current key")
A[i] ← key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] and A[PARENT(i)]
    i ← PARENT(i)
```

```
HEAP-SORT(A)
  // Input: A: an (unsorted) array
  // Output: A modified to be sorted from smallest to largest
  // Running Time: O(n log n) where n = length[A]
BUILD-MAX-HEAP(A)
for i = length[A] downto 2
    exchange A[1] and A[i]
    heap-size[A] ← heap-size[A] − 1
    MAX-HEAPIFY(A, 1)
```

```
HEAP-EXTRACT-MAX(A)
  // Input: A: an array representing a heap
  // Output: The maximum element of A and A as a heap with this element removed
  // Running Time: O(log n) where n = heap-size[A]
max ← A[1]
A[1] ← A[heap-size[A]]
heap-size[A] ← heap-size[A] − 1
MAX-HEAPIFY(A, 1)
return max
```

```
MAX-HEAP-INSERT(A, key)
  // Input: A: an array representing a heap, key: a key to insert
  // Output: A modified to include key
  // Running Time: O(log n) where n = heap-size[A]
heap-size[A] ← heap-size[A] + 1
A[heap-size[A]] ← −∞
HEAP-INCREASE-KEY(A[heap-size[A]], key)
```