

Chapter 6: Heapsort

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

PARENT(i)

1. return $(i - 1)/2$ // integer division

LEFT(i)

1. return $2i + 1$

RIGHT(i)

1. 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

1. $l = \text{LEFT}(i)$

2. $r = \text{RIGHT}(i)$

3. if $l < A.\text{heap-size}$ and $A[l] > A[i]$

4. $\text{largest} = l$

5. else $\text{largest} = i$

6. if $r < A.\text{heap-size}$ and $A[r] > A[\text{largest}]$

7. $\text{largest} = r$

8. if $\text{largest} \neq i$

9. exchange $A[i]$ with $A[\text{largest}]$

10. MAX-HEAPIFY($A, \text{largest}$)

BUILD-MAX-HEAP(A)

// $A[0:n-1]$ is an unsorted array

1. $A.\text{heap-size} = n$

2. for $i = n/2 - 1$ downto 0 //skip the leaves

3. MAX-HEAPIFY(A, i)

HEAPSORT(A)

// array $A[0:n-1]$ is unsorted

1. BUILD-MAX-HEAP(A)

2. for $i = n-1$ downto 2

3. exchange $A[1]$ with $A[i]$

4. $A.\text{heap-size} = A.\text{heap-size} - 1$

5. 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, i, newKey)

//O(lg n) time

1. if newKey < A[i].key
2. then error "new key must be larger than current key"
3. A[i].key = newKey
5. while i > 0 and A[PARENT(i)].key < A[i].key
6. exchange A[i] and A[PARENT(i)]
7. i = PARENT(i)

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[A.heap-size - 1] = x
7. MAX-HEAP-INCREASE-KEY(A, A.heap-size - 1, k)