2/

Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is Ω(lg n). (Hint: For a heap with n nodes, give node values that cause MAX-HEAPIFY to be called recursively at every node on a simple path from the root down to a leaf.)

Answer:

The worst-case occurs when the A[root] is the least value in a max-heap A. Since then, A[i] needs to recursively traverse on a longest path from the top down-to it becomes a leaf. In addition, the longest path is a continuous connection of h levels (whereas h = *lg* n is the height of the heap). So, the function MAX-HEAPIFY is needed to call *lg* n times, which is Ꝋ(*lg* n). Hence, the worst-case running time is Ω(*lg* n)

3/

Provide a tight bound for the running time of finding the **biggest** element in a binary **min-heap** with n elements.

findBiggestElement (A, n) Cost # of executions

biggest = A[n] C1 1

for i = + 1 to (n – 1) C2 n – 1 – – 1 + 1 = – 1

if A[i] ≥ biggest C3 1

biggest = A[i] C4 1

return biggest C5 1

T(n) = C\*1 + C2\*( – 1) + C3\*1 + C4\*1 + C5\*1 = O() + C = O(n)

5/

The operation HEAP-DELETE (A, i) deletes the item in node i from heap A. Give

an implementation of HEAP-DELETE that runs in O(*log* n) time for an *n*-element

max-heap.

1. Pseudocode Cost # of executions

HEAP-DELETE (A, A.heap-size, i)

*(1)* A[i] = A[A.heap-size] C1 1

/\* *overwrites the value of A[i] by the value of the smallest leaf* \*/

*(2)* A.heap-size -= 1 C2 1

*(3)* while (i > 1 and parent of i < A[i]) C3 1

*(4)* swap (A[parent(i)]) with A[i] C4 1

*(5)* i = parent(i) C5 1

*(6)* MAX-HEAPIFY(A, i) C6 1

(2) correctness justification

Done in the answer (1)

1. provide an upper bound of your procedure and give an explanation

*Explanation*: as of A[i] has been removed and replaced by the smallest leaf, A[i] must be iteratively compared with its parent then swap if the parent is ≤ A[i] to ensure A[parent(i)] ≥ A[i] (line 3, 4, 5).

In case of A[i] is small, thus it recursively swaps with its child and traverses the longest path until it becomes a leaf.

T(n) = C1\*1 + C2\*1 + C3\* + C4\* + C5\* + C6\*1

= C1 + C2 + (C3 + C4 + C5)\*(h-2+1) + C6

= C\*(h-1) + D (C and D are positive constants)

= O(h) + D (provided h = *lg* n)

= O(*lg* n) = > proved