CSC 225: Fall 2022: Lab 6

Inplace Heapsort

You are to implement heapsort in an array given as a max-heap. You may use the programming language of your choice. The heapsort algorithm takes a max-heap input array $A[1 \dots n]$, where $n = A.length$. Since the maximum element of the array is stored at the root $A[1]$, we can put it into its correct final position by exchanging it with $A[n]$. If we now discard node n from the heap—and we can do so by simply decrementing $A.heap\_size$—we observe that the children of the root remain max-heaps, but the new root element might violate the max-heap property. All we need to do to restore the max-heap property, however, is call MaxHeapify $(A, 1)$, which leaves a max-heap in $A[1 \dots n-1]$. The heapsort algorithm then repeats this process for the max-heap of size $n-1$ down to a heap of size 2.

Algorithm Heapsort(A):
Input: An n-element max-heap in an array, A.
Output: Array A sorted.

for $i = A.length$ **to** 2 **do**
swap(A[i],A[1])
$A.heap\_size = A.heap\_size - 1$
MaxHeapify($A, 1$)

In order to maintain the max-heap property, we call the procedure MaxHeapify. Its inputs are an array A and an index i into the array. When it is called, MaxHeapify assumes that the binary trees rooted at $2i$ and $2i + 1$ are max-heaps, but that $A[i]$ might be smaller than its children, thus violating the max-heap property. MaxHeapify lets the value at $A[i]$ "bubble down" in the max-heap so that the subtree rooted at index i obeys the max-heap property.

Algorithm MaxHeapify (A,i):
Input: An array, A, and index, i.
Output: Maxheap A rooted at i.

$l \leftarrow 2i$
$r \leftarrow 2i + 1$
if $l \leq A.heap\_size$ **and** $A[l] > A[i]$ **then**
$largest \leftarrow l$
else
$largest \leftarrow r$
if $larget \neq i$ **then**
swap($A[i]$,$A[largest]$)
MaxHeapify(A,$largest$)
end

Testers: You can use the following two max-heaps as input, $A1 = [, 6,5,4,1,2,3]$ and $A2 = [, 99,19,9,7,11,3,3,2,2,1]$.