6.5-5.

Argue the correctness of HEAP-INCREASE-KEY using the following loop invariant:

> At the start of each iteration of the **while** loop of lines 4–6, the subarray $A[1..\,A.\textit{heap-size}]$ satisfies the max-heap property, except that there may be one violation: $A[i]$ may be larger than $A[\textsc{Parent}(i)]$.

You may assume that the subarray $A[1..\,A.\textit{heap-size}]$ satisfies the max-heap property at the time HEAP-INCREASE-KEY is called.

Proof.

Initialization. The subarray $A[1..\,A.\textit{heap-size}]$ satisfies the max-heap property at the time HEAP-INCREASE-KEY is called. In line 3 just prior to the first iteration of the loop, $A[i]$ is increased to *key*, which is not smaller than $A[i]$ itself. This may cause $A[i]$ violate the max-heap property, for it's possible for $A[i]$ to paramount its parent after increment.

Maintenance. In each iteration, elements except $A[i]$ and $A[\textsc{Parent}(i)]$ remain unchanged, thus still satisfiy the max-heap property. If $A[i]$ is larger than $A[\textsc{Parent}(i)]$, the procedure exchanges their keys, making $A[i]$ smaller than $A[\textsc{Parent}(i)]$ and it again satisfies the max-heap property, while $A[\textsc{Parent}(i)]$ now might be larger than its parent and violates the max-heap property. Replacing $\textsc{Parent}(i)$ by i in line 6 inside the **while** loop reestablishes the loop invariant for the next iteration.

Termination. When i declines to 1 or $A[i]$ percolates up to a position where it is no longer greater than $A[\textsc{Parent}(i)]$, the iteration terminates. In the first case, as $A[1]$ is the root of the heap and has no parent, by the loop invariant, the subarray $A[1..\,A.\textit{heap-size}]$ satisfies the max-heap property. In the second case, since $A[i]$ no longer larger than $A[\textsc{Parent}(i)]$, all elements in $A[1..\,A.\textit{heap-size}]$ including $A[i]$ satisfy the max-heap property. □

Email address: informlarry@gmail.com