Zijian Gao
26319560
2/11/2016
CS311 HW1

6-1)

a. BUILD-MAX-HEAP(A) has deterministic behavior, the same input array yields the same output heap every time it is run. This is because its loop invariant is that at node i, each i + 1, i + 2 … n element is the root of a sub-heap. The only node in question is node i, which is max-heapified and so each loop iteration preserves the invariant, until the node 1 is the root of the entire heap. max-heapify(A, i), because of how it is written(lines 3-6) will left-justify node items so, if both child nodes of the i'th node have the same values, and the i'th node is less than the child nodes, then the i'th node will always be swapped with the left child. Because of this invariant, BUILD-MAX-HEAP always gives left-justified lesser values in cases where two child nodes have the same value. Since max-heapify does not perform any swaps at random and BUILD-MAX-HEAP has a proven loop invariant, it is the case that the same input array yields the same output heap every time it is run.

   BUILD-MAX-HEAP'(A) builds a max heap from the top down, bubbling up elements that are greater than their parents. Because of how the loop is written, it will always visit the left child(2i), then the right child(2i + 1), since it starts on the even index position 2. Taking the left and right sub-heaps of a heap, at no point does an element from the left ever cross over to the right, since no such operation exists in either the method, or MAX-HEAP-INSERT, only the swap operation for child to parent is done. Induction can be done on the sub-heaps.

   Hypothesis : At the i'th loop of BUILD-MAX-HEAP', the i'th element is swapped to its proper position

   Base case : A heap with 1 node trivially follows the heap rule. A heap with 3 nodes is either a heap, or the greatest node is swapped as the parent so that it is a heap.

   Inductive step : for i = 2n, MAX-HEAP-INSERT is running on the left child of a parent node and will correctly exchange parent nodes with the child node until the heap rule is maintained. for i = 2n + 1, MAX-HEAP-INSERT is running on the right child of a parent node and will correctly exchange nodes until the heap rule is maintained.

   Because BUILD-MAX-HEAP' is written such that the left child will be visited before the right child, if both the left and right children are the same value, greater than the parent, the left child will get swapped with the parent every time, which preserves the same output for the same input tree.

b. Proof that BUILD-MAX-HEAP' requires $\theta(n\log(n))$ worst-case time to build an n-element heap:

   The loop in BUILD-MAX-HEAP' iterates from 2 to n. Each iteration, it calls MAX-HEAP-INSERT, which has a worst-case run-time of $\theta(\log(n))$ since the most amount of swaps that can be done in the while loop of HEAP-INCREASE-KEY is ceiling(log(n)), the height of the heap. The other operations that are done are completed in constant time, yielding a worst-case runtime of $\theta(n\log(n))$. This runtime corresponds to a heap where every child element is greater than its parent, essentially the heap version of the worst-case insertion sort (where the inversions are maximized in an array from A[n..1]).