Assignment 3

Betty Zhang - 30040611
William Chan - 30041834
Umair Hassan - 30047693
CPSC 331

December 5, 2018

1. Proof the following claim: If the precondition for the MaxHeap Restoration after Deletion problem is satisfied and the bubbleDown algorithm is executed (with the input node x, as described) then this execution of the algorithm eventually ends, and the postcondition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

Proof: Assuming the preconditions are satisfied, and the bubbleDown algorithm is executed with a non-null input node that is in H. We will proof the claim by induction on the height of x, with the base case height = 0.
Note (from lecture 17): H is already a Max Heap if (i) x is a leaf, or (ii) x is not a leaf but the value stored at x is greater than or equal to any values stored at all children of x.
Basis: Suppose x has height = 0, then x does not have any children. The test at line 1 is checked and failed since x has no right child; we will move on to the next execution to line 13. The test at line 13 is then checked and failed as well since x has no left child. There is no more execution after line 13, so the program terminates. Since x is a leaf and the preconditions were satisfied with no changes made H, H is a MaxHeap and the postconditions are satisfied as well.
Inductive Step: Let k be an integer such that $k \geq 0$, it is necessary and sufficient to use the following:

Inductive hypothesis: Suppose algorithm is executed with a non-null input node x where x has height k, then this execution of the algorithm eventually ends, and the post condition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

To prove the following

Inductive claim: If the algorithm is executed with a non-null input node x where x has height k + 1 , then the execution of the algorithm eventually ends, and the postcondition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

Since $k \geq 0$, $k+1 \geq 1$, so k has at least one child, we split this into two cases: (1) x has two children, or (2) x only has one child.
Case (1): Suppose x has two children. Let y be the left child of x and z be the right child

of z. Since x does have a right child, the test at line 1 is checked and passed. We now have two sub cases:

Subcase (1.a): Suppose the value of y (left child of x) is greater than or equal to the value of z(right child of z). Then the test at line 1 is checked and passed, the test at line 2 is also checked and passed. The test at line 3 is then checked.

- If the value of y is not greater than the the value of x, the test at line 3 fails, and there is no more lines to executed, so the algorithm terminates. Since no changes were made and value(x) $\geq$ value (y) $\geq$ value(z), the value of x is greater or equal to the values of both its children. H is a Max Heap and the post condition are satisfied.

- If the value of y is greater than the value of x, the test at line 3 is passed and we enter the body. Line 4, 5, 6, is executed, swapping the value of x and its left child y. Since the y is now the new parent of x and z, the value of y is greater than or equal to both of its child. The recursive call at line 7 is then executed with left(x) as input. Since x previously had height k + 1 and it is now swapped with the its left child y, x becomes the new input left(x) in the recursive call with height k. Since x has height k in this recursive call, it now follows by the inductive hypothesis that execution of the algorithm eventually ends, and the post condition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

Subcase (1.b): Suppose the value of z (right child of x) is greater than the value of y(right child of z). After passing the test at line 1, the test at line 2 is checked and failed, then the test at line 8 is checked :

- If the value of z is not greater than x, the test at line 8 fails. Since there is no more lines to execute after the test, the program terminates. Since value(x) $\geq$ value(z) $>$ value (y) and changes are made to the tree, the post conditions are satisfied.

- If the value of z is greater than x, and test at line 8 passed. Line 9, 10, 11 is then executed, swapping the value of x and its right child z. Since the z is now the new parent of x and y, the value of z is greater than both of its child. Line 12 is then executed with a recursive call of the algorithm with the input right(x). Since x previously had height k + 1 and it is now swapped with the its right child z, x becomes the new input right(x) in the recursive call with height k. Since x has height k in this recursive call, it now follows by the inductive hypothesis that execution of the algorithm eventually ends, and the post condition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

Case (2): Suppose x only has one child, it follows by the the definition of a binary heap that x has a left child. After failing the test at line 1, line 13 is checked and passed. The test at line 14 is then checked:

- If the value of y is not greater than x (less than or equal to x), the test at line 14 fails and the program terminates. Since the value of x is greater than its only child and no changes are made to the tree, the post conditions are satisfied.

- If the value of y is greater than x, the test at 14 is passed. Line 15, 16, 17 is then executed, swapping the value of x and its left child y. Since the y is now the new parent of x, the value of y is greater than its child. The recursive call at line 18 is then executed with left(x) as input. Since x previously had height k + 1 and it is now swapped with the its left child y, x becomes the new input left(x) in the recursive call with height k. Since x has height k in this recursive call, it now follows by the inductive hypothesis that execution of the algorithm eventually ends, and the post condition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

Since the program terminates with the post conditions satisfied in all cases, we can now conclude that, by induction on the height of x, this execution of the algorithm eventually ends, and the postcondition for the MaxHeap Restoration after Deletion problem is satisfied on termination.

To prove correctness:
One can see by inspection of code that the bubbleDown algorithm does not have any undocumented side-effects. That is, if does not use extra inputs, change any inputs it should not, create an undocumented outputs, or access or modify any undocumented global data.
Since we also proved the claim If the precondition for the MaxHeap Restoration after Deletion problem is satisfied and the bubbleDown algorithm is executed (with the input node x, as described) then this execution of the algorithm eventually ends, and the postcondition for the MaxHeap Restoration after Deletion problem is satisfied on termination by induction, it follows that the the algorithm is correct.

4. As noted in Tutorial Exercise 17, it is useful to maintain a reference to the node, in the MinHeap being represented, that was most recently added (and has not yet been deleted). The exercise described a process that can be used to update this reference after a deleteMin operation has been performed.
Describe the changes you would need to make in order update this reference as a new node is being added during an insert operation, instead.

If the min heap is implemented without an array representation:
Cases:

(a) The reference to be u was null before the insertion

- In this case, the heap was empty before the insertion, the reference will be updated to reference the new element inserted (in this case, the root)

(b) The reference was a reference to a node that is the left child of its parent

- The newly inserted node becomes right child of the reference's parent, update the reference to the right child of its parent

(c) The reference was a right child of its parent.
Let m be the current level of the referenced node before the insertion

(c.1) If the reference was the right-most node at level m

- This is true if every ancestor of the referenced node was a right child of a parent (except for the root, which has no parent at all)
- This means that, after insertion, the latest inserted node would the the left-most node at the next level (level m + 1). Update the reference to be a reference of the left most node at level m + 1

(c.2) If the reference was a right child but not the right-most node

- This is true if there exist an ancestor of the referenced node such that the ancestor node is a left child of a parent. let y be the located ancestor node and z be the parent y.
- This means that the referenced node was the right-most node in level m of the subtree where y is the root. After the insertion, update the reference to be the left-most node (at level m) of right subtree of z.

If an array A is used to represent a bounded MinHeap:

If the reference was a non-null node and heap size is less than the length of the array(heap not full)

- Update reference to A[size -1] where size is an integer representing the size of the heap and A is the array used for representing the MinHeap

The reference was a reference to a non-null node in the tree and the size of the heap is equal to the size of the array represented (heap full)

- HeapFullException is thrown in insertion and the reference is unchanged