4999406

Problem 2:

Part A:

If the method to print keys used preorder traversal the output will not be in order for example.
A binary heap with 4 as the root, the right child being 1 and the left 10, the output would be 4, 10, 1.
In in order traversal it will also not be sorted and provide a result of: 10, 1, 4. There is no particular order given to the children of any focal node from the root down to every level of the binary heap.
So as with preorder inorder will only have a proper sort if the keys entered happened to appear in an order that appealed to the traveral's pattern.

Part B:

(Pseudo-code implementation for finding all nodes in a min-heap less than a specified value x):

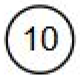
```
FindNodesLessThanX (Value X, MinHeap HeapPosition):

NodesFound = 0

If HeapPosition >= HeapSize:
      Return NodesFound as result.

If MinHeap's node at HeapPosition is <= X :
      Increment NodesFound.
      Do FindNodesLessThanX for X on left child of HeapPosition.
      Add result to NodesFound.
      Do FindNodesLessThanX for X on right child of HeapPosition.
      Add result to NodesFound.

Return NodesFound as result.
```

Declaring a variable initialized to 0 (NodesFound) is O(1) complexity.
The statements to check and then return if HeapPositon is greater than or equal to Heap size is O(2).
The third set of statements regarding the situation where a node is found in a valid heap position, that is less than the value used as a reference: (X), is O(A + 4).
Where A is the number of nodes left in the recursive route of a child of the current HeapPosition's node (That could be less than X in worst case).
The last statement is O(1)
The total complexity is O(n + 5); Resolving to a complexity of O(n);
Where n is the total nodes, as A's possible nodes worst case can converge to a total of n.

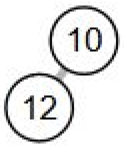
Part C:

Min-Heap Array: 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2

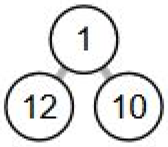
Insert 10:



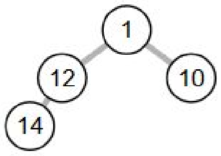
Insert 12:



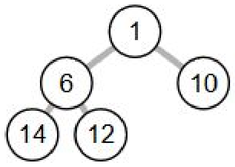
Insert 1:



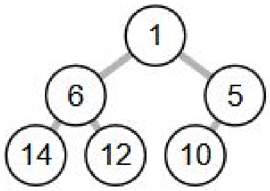
Insert 14:



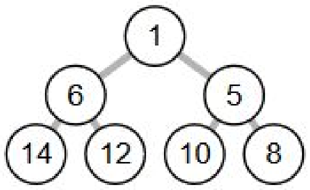
Insert 6:



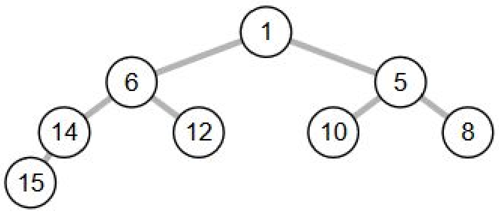
Insert 5:



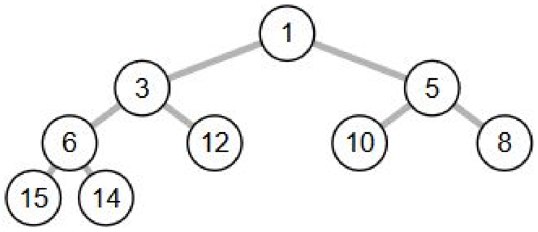
Insert 8:



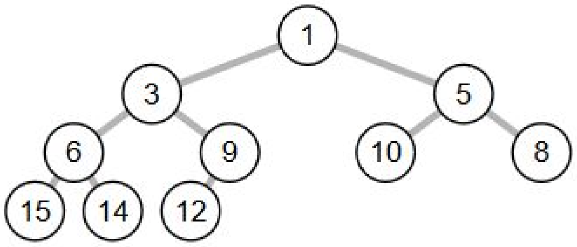
Insert: 15:



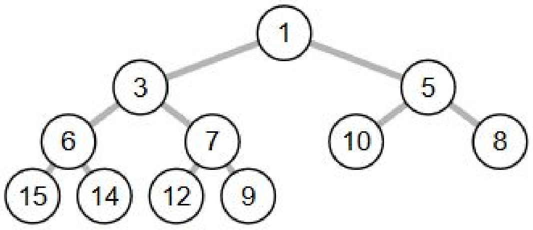
Insert 3:



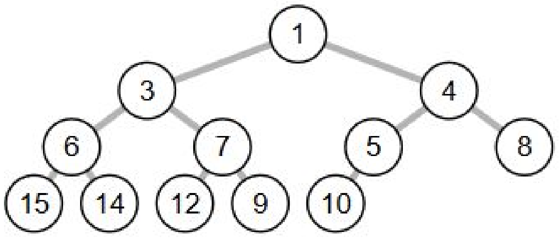
Insert 9:



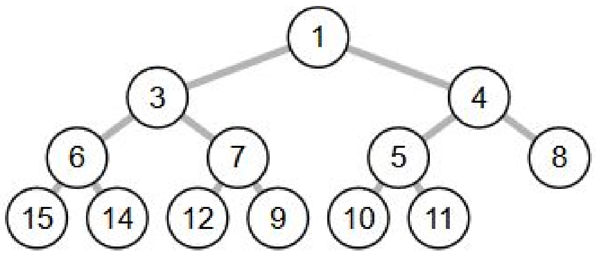
Insert 7:



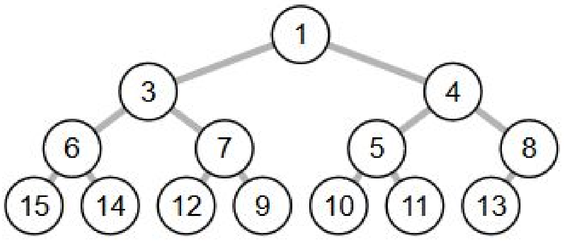
Insert 4:



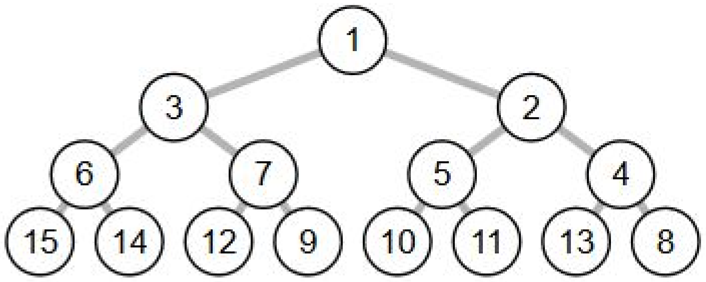
Insert 11:



Insert 13:



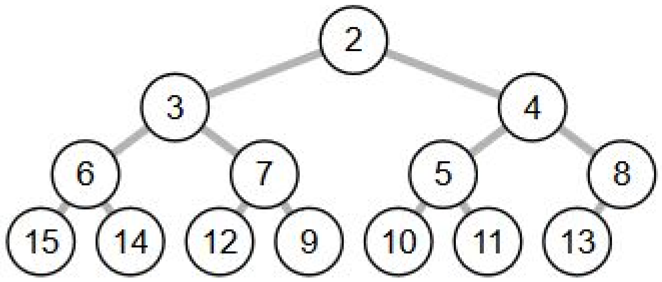
Insert 2:



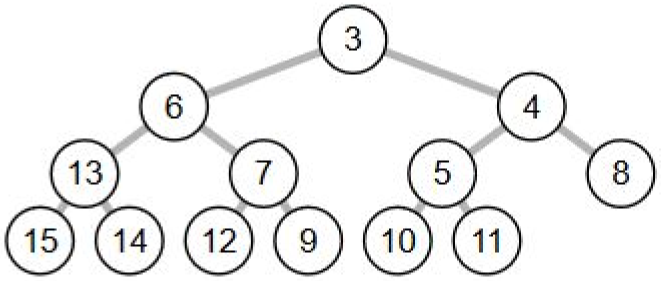
Part D:

Min-Heap Array: 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2

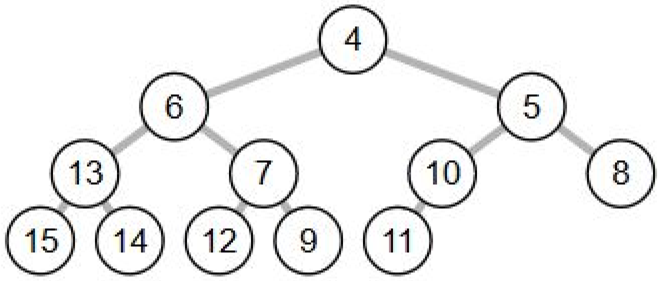
deleteMin(1st):



deleteMin(2nd):



deleteMin(3rd):



Part E:

```
public void replaceKey(Integer oldKey, Integer newKey)
{
        Integer position = 0;

        for ( ;position < HeapArray.length && !HeapArray[position].equals(oldKey); position++);

        if ( position == (HeapArray.length - 1) )
        {
                System.out.println("The key: " + oldKey + " is not within the heap.");

                return;
        }
        else
        {
                HeapArray[position] = newKey;

                if ((position > 0) && (HeapArray[position].compareTo(HeapArray[position/2]) < 0))
                {
                        perlocateUp(position);
                }
                else
                {
                        perlocateDown(position);
                }
        }
  }
```

Part F:

Declaring an integer with “position” as indentifier is O(1).
For loop to check every element of heap for oldKey is O(n).
Outer conditionals after is O(InnerConditonal + 4).
Where InnerConditional is O(Log(n)) for the perlocations.
Total is O(n + Log(n) + 5), this resolves to a complexity of O(n) toward infinity.