Homework 4 Code Refactoring

1. (1 point) Read http://en.wikipedia.org/wiki/Code_refactoring first. Use your own word to briefly explain what is code refactoring?

2. (4 points) The following code has passed all test cases. First read the code and understand how the code works. Then refactoring the code little by little. While you are refactoring the code, write down the changes that you made. For example:

   1. The comment should not use vague word. It should not use "bubble down to its appropriate position". It should say "bubble down until both its children are greater"

   2. ...

   3. ...

   The bubbleDown method in Heap.java of Assignment 3.

```java
/**
 * The entry should be bubbled down to its appropriate position
 * @param p
 */
@TimeComplexity("O(lg n)")
private void bubbleDown(Position<Entry<K,V>> p)
{
// While the node still needs to be bubbled down, keep looping the following.
// TCJ:  Because the maximum amount of times this loop can run is determined by the height
// of the tree, which is equal to lg n, that is the time complexity of the function.
while(tree.isInternal(p) &&
    (tree.hasLeft(p) && (tree.left(p).element().getKey().compareTo(p.element().getKey()) > 0) ||
    (tree.hasRight(p) && tree.right(p).element().getKey().compareTo(p.element().getKey()) > 0)))
{
   boolean lessThanLeft = false;
   boolean lessThanRight = false;

   if(tree.hasLeft(p) && tree.left(p).element().getKey().compareTo(p.element().getKey()) > 0)
   {
      lessThanLeft = true;
   }

   if(tree.hasRight(p) && tree.right(p).element().getKey().compareTo(p.element().getKey()) > 0)
   {
      lessThanRight = true;
   }

   if(lessThanLeft && lessThanRight)
   {
      if(tree.left(p).element().getKey().compareTo(tree.right(p).element().getKey()) > 0)
      {
         tree.swapElements(p, tree.left(p));
         p = tree.left(p);
      }
      else
      {
         tree.swapElements(p, tree.right(p));
         p = tree.right(p);
      }
   }
   else if(lessThanLeft && !lessThanRight)
   {
      tree.swapElements(p, tree.left(p));
      p = tree.left(p);
   }
   else if(!lessThanLeft && lessThanRight)
   {
      tree.swapElements(p, tree.right(p));
      p = tree.right(p);
   }
}
}
```

   (a) List of changes

(b) The new source code

3. (4 points) Do the same as previous question for the following code.
The insert method in OrderPQ.java of Assignment 3.

```java
/** Adds an entry to the PriorityQueue in its proper order.
 *
 * @param key The key to find the entry added.
 * @param value The value that is being stored in the entry.
 * @return returns the new entry added to the PriorityQueue.
 * @throws InvalidKeyException
 */
@TimeComplexity("O(n)")
public Entry<K,V> insert(K key, V value) throws InvalidKeyException
{
    // Declare method's variables.
    PQEntry<K,V> ret = new PQEntry<K,V>(key, value);
    boolean added = false;

    // If the size of the Queue is 0, add the entry to the first index.
    if(_size == 0)
    {
        // Add the data and increment size.
        _data.add(0, ret);
        ++_size;

        // Return the entry that was added.
        return ret;
    }

    // TCJ:  Because we do not know where we are supposed to add the entry, we
    // need to iterate through the entire sequence to find it, causing it to run
    // in O(n) time.
    for(int index = 0; index < _data.size(); ++index)
    {
        // Determines if the entry has found its proper place.
        if(_data.atIndex(index).element().getKey().compareTo(ret.getKey()) > 0)
        {
            // if it has, add it in and break the loop.
            _data.add(index, ret);
            index = _data.size();
            added = true;
        }
    }

    // If the added variable is still false, this means
    // the data must go in the next empty index.
    if(!added)
    {
        // Add the data to the next empty index.
        _data.addLast(ret);
    }

    // Increment size and return the new entry.
    _size++;
    return ret;
}
```

(a) List of changes

(b) The new source code

4. (1 point) Print and sign all team member's names below to acknowledge that you have carefully reviewed the code line by line with your team and refactored the code to your best. Each team needs to have two or three team members. Each student may work in multiple teams.

_____

_____

_____