Final Exam

CMP-338 | 110 Points Total | Spring 2018

1. (20 Points) The corrected methods:

```java
// add the given element at the given index
// return true if successful, false otherwise
public boolean add(I element, int index) {
   boolean rc = false;
   Node<I> newNode = new Node<I>(element);
   Node<I> curNode = head;

   if (index == 0) {
      if (head == null) {
         tail = newNode;
      }
      newNode.setNext(head);
      head = newNode;
      listSize++;
      rc = true;
   } else if (index == listSize) {
      add(element);
      rc = true;
   } else  if (index < listSize) {
      for ( int j = 1  ; j < index ; j++ ) {
         curNode = curNode.getNext();
      }
      newNode.setNext(curNode.getNext());
      curNode.setNext(newNode);
      listSize++;
      rc = true;
   }
   return rc;
}
```

```java
// remove the element at the specified index
// return true if successful, false otherwise
public boolean remove(int index) {
   boolean result = false;
   Node<I> curNode = head;
   Node<I> prevNode = null;

   if (index < listSize) {
      if (index == 0) {
         head = head.getNext();
         result = true;
      } else {
         for ( int i = 0 ; i < index ; i++ ) {
            prevNode = curNode;
            curNode = curNode.getNext();
         }
         prevNode.setNext(curNode.getNext());
         if (index == (listSize - 1)) {
            tail = prevNode;
         }
         result = true;
      }
      listSize--;
   }
   return result;
}
```

2. (30 Points) The method to reverse the contents of a `List`:

```
public List<I> reverseList(List<I> list) {
    List<I> reversed = new List<I>();
    Stack<I> stack = new Stack<I>();

    int index = 0;
    while (index < list.size()) {
        I element = list.get(index);
        stack.push(element);
        index++;
    }

    while (!stack.isEmpty()) {
        I element = stack.pop();
        reversed.add(element);
    }

    return reversed;
}
```

3. (20 Points) The method to create a balanced `BinarySearchTree`:

```
public BinarySearchTree<K,V> createBalancedTree(TreeItem<K,V>[] items) {
    BinarySearchTree<K,V> tree = new BinarySearchTree<K,V>();

    TreeNode<K,V> root = balanceTree(items, 0, items.length - 1);

    tree.setRoot(root);

    return tree;
}

private TreeNode<K,V> balanceTree(Object[] arr, int first, int last) {
    TreeNode<K,V> node = null;

    if (first <= last) {
        int mid = (first + last) / 2;
        node = new TreeNode<K,V>((TreeItem<K,V>)arr[mid]);
        node.setLeftChild(balanceTree(arr, first, (mid - 1)));
        node.setRightChild(balanceTree(arr, (mid + 1), last));
    }

    return node;
}
```

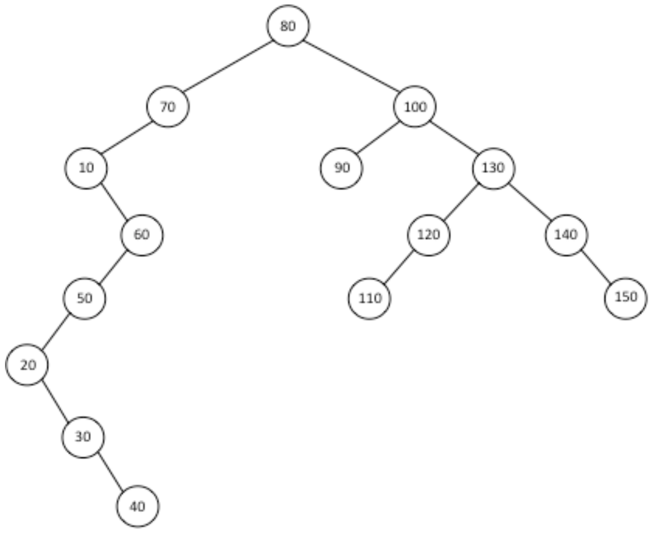
4. (20 Points) The correct heapify method:

```
private void heapify() {
   int last = this.heapArray.length - 1;
   int parent = (last - 1) / 2;

   while (parent >= 0) {
      siftDown(parent);
      parent = parent - 1;
   }
}

private void siftDown(int node) {

   while (node < this.heapArray.length) {
      int leftChild = (2 * node) + 1;
      int rightChild = (2 * node) + 2;
      int swap = node;

      if ((leftChild < this.heapArray.length) &&
          (this.heapArray[node].compareTo(this.heapArray[leftChild]) < 0)) {
         swap = leftChild;
      }

      if ((rightChild < this.eapArray.length) &&
          (this.heapArray[swap].compareTo(this.heapArray[rightChild]) < 0)) {
         swap = rightChild;
      }

      if (swap == node) {
         return;
      } else {
         T temp = this.heapArray[node];
         this.heapArray[node] = this.heapArray[swap];
         this.heapArray[swap] = temp;
         node = swap;
      }
   }
}
```

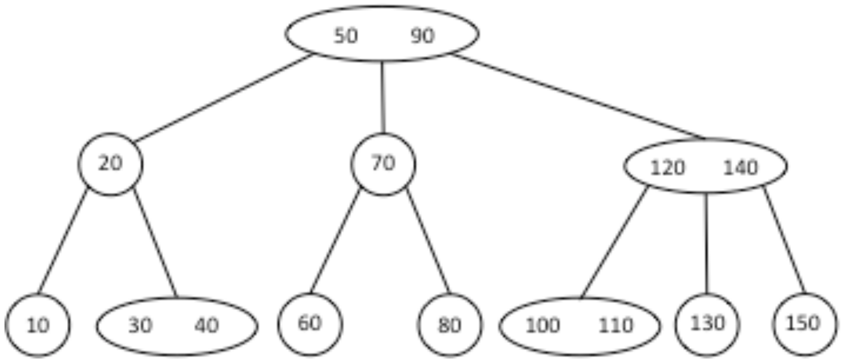
5. (20 Points) The following list of numbers:

80, 70, 10, 100, 90, 130, 60, 120, 50, 110, 20, 30, 140, 150, 40

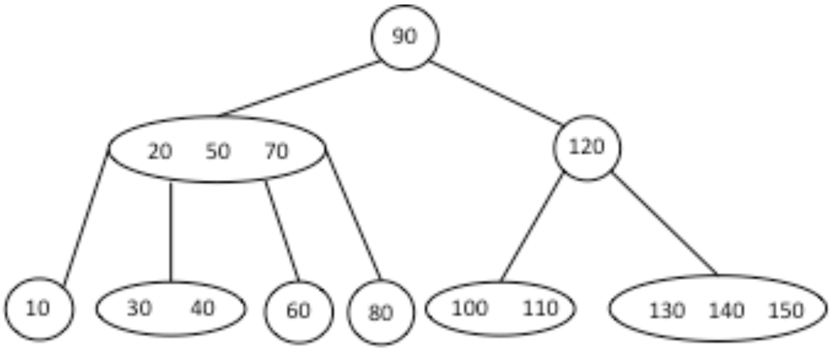
a. (5 Points) Inserted in the given order into a Binary Search Tree produces the following tree:



b. (5 Points) Inserted in the given order into a 2-3 Tree produces the following tree:



c. (5 Points) Inserted in the given order into a 2-3-4 Tree produces the following tree:



d. (5 Points) The numbers in the following order produces a full Binary Search Tree:

80, 40, 20, 10, 30, 60, 50, 70, 120, 100, 90, 110, 140, 130, 150