CS211 ALGORITHMS & DATA STRUCTURES II

LAB 3

Dr. Phil Maguire

QUICKSORT – HUFFMAN ENCODING

Underlined indicates a pivot
Two red numbers indicates a swap
Green numbers are sorted

7	8	2	5	1	9	3	<u>6</u>
3	8	2	5	1	9	7	<u>6</u>
3	1	2	5	8	9	7	<u>6</u>
3	1	2	5	6	9	7	8
3	1	2	<u>5</u>	6	9	7	8
3	1	2	5	6	9	7	8
3	1	<u>2</u>	5	6	9	7	8
1	3	<u>2</u>	5	6	9	7	8
1	2	3	5	6	9	7	8
<u>1</u>	2	3	5	6	9	7	8
1	2	<u>3</u>	5	6	9	7	8
1	2	3	5	6	9	7	<u>8</u>
1	2	3	5	6	7	9	<u>8</u>
1	2	3	5	6	7	8	9

1 2 3 5 6 7 8 9

1 2 3 5 6 7 8 9

1 2 3 5 6 7 8 9

MEDIAN OF THREE

(assume that when there is no exact middle element it takes the middle-left element)

7 8 2 5 1 9 3 6

5 8 2 6 1 9 3 7

5 8 2 3 1 9 6 7

5 1 2 3 8 9 6 7

5 1 2 3 6 9 8 7

5 1 2 3 6 9 8 7

1 3 2 5 6 9 8 7

1 2 3 5 6 9 8 7

1 2 3 5 6 9 8 7

1 2 3 5 6 9 8 7

1 2 3 5 6 7 8 9

1 2 3 5 6 7 8 9

1 2 3 5 6 7 8 9

Huffman.java

```
PriorityQueue < Tree >  PQ = new PriorityQueue < Tree >() ;

//make a priority queue to hold the forest of trees


       for(int i=0; i<array.length; i++){
//go through frequency array
           if(array[i]>0){
//print out non-zero frequencies - cast to a char
               System.out.println("'"+(char)i+"' appeared "+array[i]+((array[i] == 1)
? " time" : " times"));
               Tree myTree = new Tree();    //create a new Tree
               myTree.frequency = array[i]; //set the cumulative frequency of Tree
               myTree.root=new Node();      //insert the letter as the root node
               myTree.root.letter = (char)i;//insert the letter as the root node
               PQ.add(myTree);
//add the Tree into the PQ
           }
        }


        while(PQ.size()>1){
//while there are two or more Trees left in the forest
            Tree firstTree = PQ.poll();
//get the two trees
            Tree secondTree = PQ.poll();
            Tree comboTree = new Tree();
//combine them into a new tree
            comboTree.frequency=firstTree.frequency+secondTree.frequency;
//add the cumulative frequency of both trees
            comboTree.root=new Node();
//insert a default root node (or else you get a null pointer exception)

//insert a default root node (or else you get a null pointer exception)
            comboTree.root.leftChild=firstTree.root;
//the two trees are the left and right children of the combo tree
            comboTree.root.rightChild=secondTree.root;
            PQ.add(comboTree);
//add the combo tree back into the PQ
        }

        Tree HuffmanTree = PQ.poll();
//now there's only one tree left - get its codes
        int totalLength=0;
//keeps track of the length of the new compressed version
        String theCode;
        for(int i=0; i<sentence.length(); i++){
            theCode=HuffmanTree.getCode(sentence.charAt(i));
            System.out.print(theCode+" ");
//get the code for the letter
            totalLength+=theCode.length();
//track the length of the solution
        }
        //print out all the info
        System.out.println("\nCompressed size is "+totalLength+" bits /
"+sentence.length()*7+" bits = "+(int)((totalLength*100)/(sentence.length()*7))+"
%\n");
```

```
}
```

Tree.java

```
   private void inOrder(Node localRoot, char letter, String path){ //the path
variable tracks the current search path
      if(localRoot != null){ //if root is null we've gone off the edge of the
tree - back up
          if(localRoot.letter==letter){
            this.path=path;     //if we've found the letter, note the path -
final path = current search path
          }else{
            inOrder(localRoot.leftChild, letter, path+"0"); //go left and add
"0" to the current search path
            inOrder(localRoot.rightChild, letter, path+"1");    //go right
and add "1" to the current search path
          }
      }
      return; //quit searching this branch of the tree
   }
```