CMSC 451 Homework 3

1. Shown below is the code for the insertion sort consisting of two recursive methods that replace the two nested loops that would be used in its iterative counterpart:

```
void insertionSort(int array[])
  {
    insert(array, 1);
  }
  void insert(int[] array, int i)
  {
    if (i < array.length)
    {
      int value = array[i];
      int j = shift(array, value, i);
      array[j] = value;
      insert(array, i + 1);
    }
  }
  int shift(int[] array, int value, int i)
  {
    int insert = i;
    if (i > 0 && array[i - 1] > value)
    {
      array[i] = array[i - 1];
      insert = shift(array, value, i - 1);
    }
    return insert;
  }
```

   Draw the recursion tree for `insertionSort` when it is called for an array of length 5 with data that represents the worst case. Show the activations of `insertionSort`, `insert` and `shift` in the tree. Explain how the recursion tree would be different in the best case.

2. Refer back to the recursion tree you provided in the previous problem. Determine a formula that counts the numbers of nodes in that tree. What is Big-Θ for execution time? Determine a formula that expresses the height of the tree. What is the Big-Θ for memory?

3. Provide a generic Java class named `SortedPriorityQueue` that implements a priority queue using a sorted list implemented with the Java `ArrayList` class. Make the implementation as efficient as possible.

4. Consider the following sorting algorithm that uses the class you wrote in the previous problem:

```
void sort(int[] array)
{
  SortedPriorityQueue<Integer> queue = new SortedPriorityQueue();
  for (int i = 0; i < array.length; i++)
    queue.add(array[i]);
  for (int i = 0; i < array.length; i++)
    array[i] = queue.remove();
```

```
}
```

Analyze its execution time efficiency in the worst case. In your analysis you may ignore the possibility that the array list may overflow and need to be copied to a larger array. Indicate whether this implementation is more or less efficient than the one that uses the Java priority queue.

Grading Rubric

Problem	Meets	Does Not Meet
Problem 1	**10 points**	**0 points**
	Recursion tree is drawn correctly (8)	Recursion tree is not drawn correctly (0)
	Best case tree is described correctly (2)	Best case tree is not described correctly (0)
Problem 2	**10 points**	**0 points**
	Provided correct formula for number of nodes in tree (3)	Did not provide correct formula for number of nodes in tree (0)
	Provided correct Big-Theta for execution time (2)	Did not provide correct Big-Theta for execution time (0)
	Provided correct formula for tree height (3)	Did not provide correct formula for tree height (0)
	Provided correct Big-Theta for memory (2)	Did not provide correct Big-Theta for memory (0)
Problem 3	**10 points**	**0 points**
	Provided class correctly implements a priority queue (4)	Provided class does not correctly implement a priority queue (0)
	Provided class is generic (1)	Provided class is not generic (0)
	Provided class uses an array list (1)	Provided class does not use an array list (0)
	List in class is maintained in sorted order (2)	List in class is not maintained in sorted order (0)
	Implementation is most efficient (2)	Implementation is not most efficient (0)
Problem 4	**10 points**	**0 points**
	Provided correct worst case analysis (8)	Did not provide correct worst case analysis (0)
	Provided correct efficiency comparison to Java priority queue (2)	Did not provide correct efficiency comparison to Java priority queue (0)