Homework #2  Data Structures
Due:  February 4, 2010 (Thursday at 5 PM)

Homework #2 is a pencil-and-paper assignment involving algorithm/program analysis. Answer the questions for each of the following algorithms.

1. Another simple sort is called insertion sort. Recall that in a simple sort:
   - the outer loop keeps track of the dividing line between the sorted and unsorted part with the sorted part growing by one in size each iteration of the outer loop.
   - the inner loop's job is to do the work to extend the sorted part's size by one.

After several iterations of insertion sort’s outer loop, an array might look like:

<table>
<thead>
<tr>
<th>Sorted Part</th>
<th>Unsorted Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8</td>
</tr>
<tr>
<td>10 20 35 40</td>
<td>45 60 25 50 90</td>
</tr>
</tbody>
</table>

In insertion sort the inner-loop takes the "first unsorted element" (25 at index 6 in the above example) and "inserts" it into the sorted part of the list "at the correct spot." After 25 is inserted into the sorted part, the array would look like:

<table>
<thead>
<tr>
<th>Sorted Part</th>
<th>Unsorted Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5</td>
<td>6 7 8</td>
</tr>
<tr>
<td>10 20 25 35</td>
<td>40 45 60 50 90</td>
</tr>
</tbody>
</table>

Code for insertion is given below:

```c
void insertionSort(int array[], int size) {
    int firstUnsortedIndex, testIndex, elementToInsert;
    for (firstUnsortedIndex = 1; firstUnsortedIndex < size; firstUnsortedIndex++) {
        elementToInsert = array[firstUnsortedIndex];
        testIndex = firstUnsortedIndex - 1;
        while (testIndex >=0 && array[testIndex] > elementToInsert ) {
            array[ testIndex + 1 ] = array[ testIndex ];
            testIndex = testIndex - 1;
        } // end while
        array[ testIndex + 1 ] = elementToInsert;
    } // end for
} // end insertionSort
```

To help you analyze the above algorithm to determine its worst-case theta notation, $\Theta()$, complete the following table like we did in class to determine how many times the inner-loop executes.

<table>
<thead>
<tr>
<th>firstUnsortedIndex:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>size-2</th>
<th>size-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range of testIndex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of times</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inner-loop executes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
b) Write a summation formula for the total number of times that the inner-loop executes.

c) What is the overall worst-case \( \Theta(\cdot) \) notation for insertion sort?

d) What is the worst-case \( \Theta(\cdot) \) notation for the number of element moves?

e) What is the worst-case \( \Theta(\cdot) \) notation for the number of element comparisons?

f) What is the best-case \( \Theta(\cdot) \) notation for the number of element moves?

g) What is the best-case \( \Theta(\cdot) \) notation for the number of element comparisons?

2. Consider the following alternative coding of insertion sort which utilizes an insert function.

```c
void insert(int numbers[], int elementToInsert, int firstSortedIndex) {
    int testIndex;
    testIndex = firstSortedIndex;
    while (testIndex >= 0 && numbers[testIndex] > elementToInsert) {
        numbers[testIndex + 1] = numbers[testIndex];
        testIndex = testIndex - 1;
    } // end while
    numbers[testIndex + 1] = elementToInsert;
} // end Insert

void insertionSort(int numbers[], int length) {
    int firstUnsortedIndex;
    for (firstUnsortedIndex = 1; firstUnsortedIndex < length; firstUnsortedIndex++) {
        insert(numbers, numbers[firstUnsortedIndex], firstUnsortedIndex-1);
    } // end for
} // end insertionSort
```

a) Since the insertionSort function only calls Insert, does this improve the worst-case \( \Theta(\cdot) \) notation. Explain your answer.

b) What implications does your answer in part (a) have for analyzing large programs that are split into many functions?
3. Consider a third alternative of insertion sort (no code provided) which utilizes the fact that the “sorted” part of the array is sorted. The previous two versions of insertion sort linearly/sequentially scanned the sorted part of the array for the insertion point. In this problem consider the performance improvements achieved by performing a binary search on the sorted part of the array to find the spot to insert the “first unsorted element.

Answer the following questions about this **third alternative of insertion sort** (you do NOT need to write the code for this algorithm):

a) What is the worst-case $\Theta(\ )$ notation for the number of element moves? (justify your answer)

b) What is the worst-case $\Theta(\ )$ notation for the number of element comparisons? (justify your answer)

c) What is the overall worst-case $\Theta(\ )$ notation for this third insertion sort? (justify your answer)

d) What is the best-case $\Theta(\ )$ notation for the number of element moves? (justify your answer)

e) What is the best-case $\Theta(\ )$ notation for the number of element comparisons? (justify your answer)

f) What is the overall best-case $\Theta(\ )$ notation for this third insertion sort? (justify your answer)