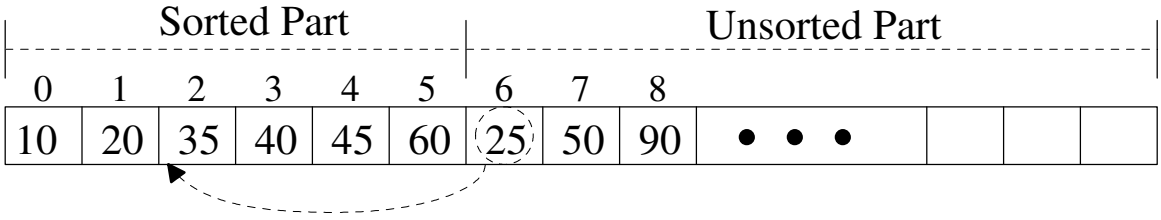
Homework #1 Data Structures
Due: January 27, 2011 (Thursday at 5 PM)

Homework #1 is a pencil-and-paper assignment involving algorithm/program analysis. Answer the questions for each of the following algorithms.

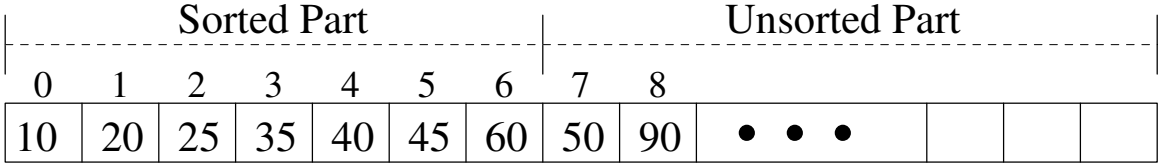
1. Another simple sort is called insertion sort. Recall that in a simple sort:

- 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 list might look like:



In insertion sort the inner-loop takes the "first unsorted item" (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 list would look like:



Code for insertion is given below:

```
def insertionSort(myList):
    """Rearranges the items in myList so they are in ascending order"""

    for firstUnsortedIndex in xrange(1,len(myList)):
        itemToInsert = myList[firstUnsortedIndex]
        # Scan the sorted part from the right side
        # Shift items to the right while you have not scanned past the left
        # end of the list and you have not found the spot to insert
        testIndex = firstUnsortedIndex - 1

        while testIndex >= 0 and myList[testIndex] > itemToInsert:
            myList[testIndex+1] = myList[testIndex]
            testIndex = testIndex - 1

        # Insert the itemToInsert at the correct spot
        myList[testIndex + 1] = itemToInsert
```

a) Write a summation formula for the total number of times that the inner-loop executes.

b) What is the worst-case $O(\ )$ notation for the number of item moves?

c) What is the worst-case $O(\ )$ notation for the number of item comparisons?

d) What is the overall worst-case $O(\ )$ notation for insertion sort?

e) What is the best-case $O(\ )$ notation for the number of item moves?

f) What is the best-case $O(\ )$ notation for the number of item comparisons?

g) What is the overall best-case $O(\ )$ notation for insertion sort?

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

```
def insert(myList, itemToInsert, lastSortedIndex):
    """  Inserts itemToInsert into myList's sorted part at the
         correct spot"""
    # Scan the sorted part from the right side
    # Shift items to the right while you have not scanned past the left
    # end of the list and you have not found the spot to insert
    testIndex = lastSortedIndex
    while testIndex >= 0 and myList[testIndex] > itemToInsert:
        myList[testIndex+1] = myList[testIndex]
        testIndex = testIndex - 1

    # Insert the itemToInsert at the correct spot
    myList[testIndex + 1] = itemToInsert


def insertionSort(myList):
    """Rearranges the items in myList so they are in ascending order"""
    for firstUnsortedIndex in xrange(1,len(myList)):
        insert(myList, myList[firstUnsortedIndex], firstUnsortedIndex-1)
```

a) Since the insertionSort function only calls insert, does this improve the worst-case $O(\ )$ 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. Rewrite the first insertion-sort code (on the first page) so that is sorted in **descending order** (i.e., from largest to smallest). For this part, just turn in a print-out ("hard-copy") of your program.