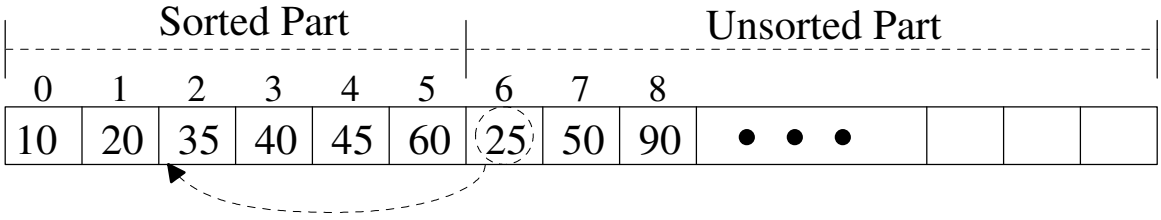
Homework #1 Data Structures
Due: January 27, 2011 (Friday at 2 PM)

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

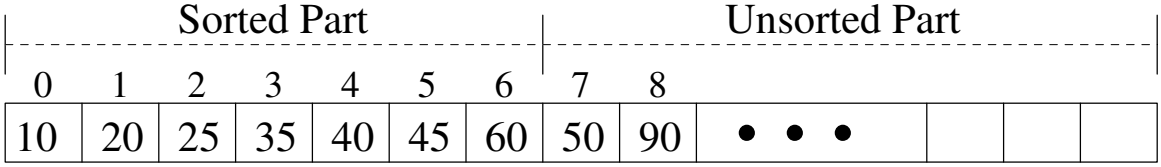
1. Another simple sort is called insertion sort. Recall that in a simple sort:

- 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 we developed in class 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) What is the worst-case $O(\ )$ notation for the number of item moves?

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

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

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

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

f) 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. Let "n" be the len(myList). What is the best (slowest growing) big-oh notation for the following code? (You may assume that n is a power of 2.)

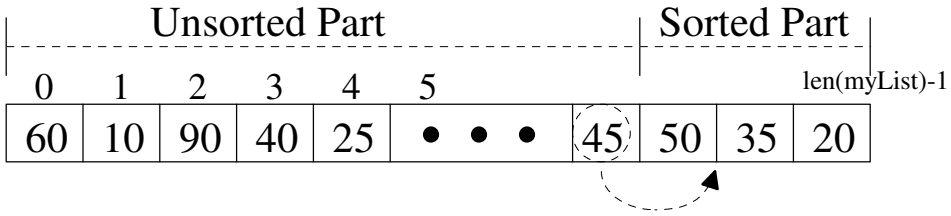
```
i = 1
while i <= len(myList):
    for j in xrange(i):
        myList[j] = 2 + myList[j]

    i = i * 2
```

Programming Part:

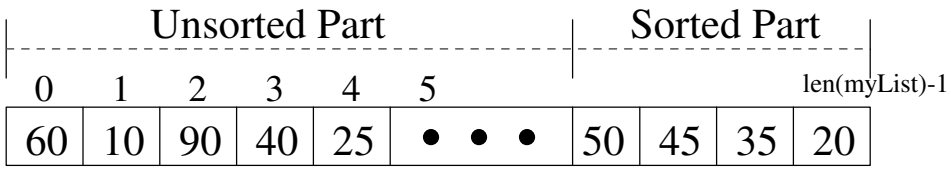
4. Rewrite the first insertionSort code (without `insert` funtion) so that it:

- sorted in **descending order** (i.e., from largest to smallest)
- builds the sorted part on the right side of the list (see below for example)

After several iterations of your insertion sort's outer loop, a list might look like:



Your insertion-sort inner-loop should take the "last unsorted item" (45 in the above example) and "inserts" it into the sorted part of the list "at the correct spot." After 45 is inserted into the sorted part, the list would look like:



Just turn in a print-out ("hard-copy") of your program and the resulting output of sorting a list initially ordered as: 60, 10, 90, 40, 25, 20, 35, 45, 50