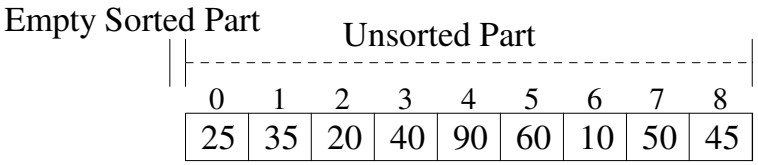
Homework #1 Data Structures
Due: Sept. 6, 2011 (Tuesday at 5 PM)

Homework #1 contains both a pencil-and-paper algorithm/program analysis, and a programming part.

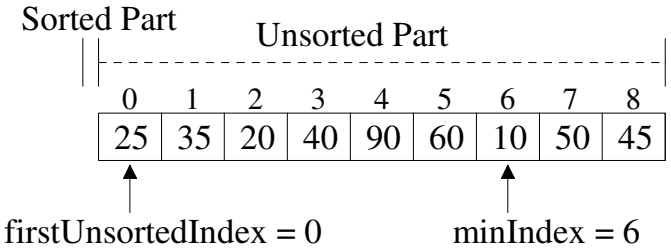
Pencil-and-paper questions: Answer the questions for each of the following algorithms.

1. Recall the selection sort algorithm discussed in class. Selection sort's inner loop scans the unsorted part of the list to find the minimum item. The minimum item in the unsorted part is then exchanged with the first unsorted item to extend the sorted part by one item.

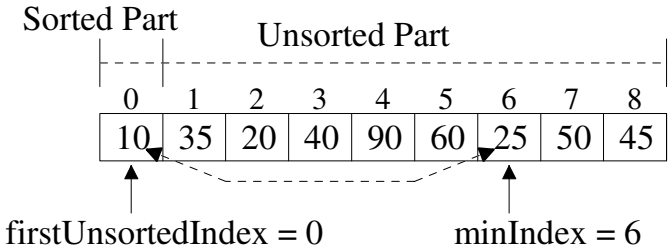
At the start of the first iteration of the outer loop the initial list is completely unsorted:



The inner loop scans the unsorted part and determines that the index of the minimum item, minIndex = 6.



After the inner loop (but still inside the outer loop), the item at minIndex is exchanged with the item at firstUnsortedIndex. Thus, extending the Sorted Part of the list by one item.



Code developed in class for selection sort is given below:

```
def selectionSort(aList):
    for firstUnsortedIndex in xrange(len(aList)-1):
        # look for minimum item in unsorted part of list
        # Assume minimum is the first item in the unsorted part
        minIndex = firstUnsortedIndex

        # scan down the unsorted part of the list to correct the assumption
        for testIndex in xrange(firstUnsortedIndex+1, len(aList)):
            if aList[testIndex] < aList[minIndex]:
                minIndex = testIndex

        # exchange the items at minIndex and firstUnsortedIndex
        temp = aList[firstUnsortedIndex]
        aList[firstUnsortedIndex] = aList[minIndex]
        aList[minIndex] = temp
```

a) Write the worst-case $O(\ )$ notation for the total number of times that the inner-loop executes (i.e., number of comparison).

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

2. Consider the following alternative coding of selection sort which utilizes a findMin function.

```
def findMin(aList, startIndex, endIndex):
    """Returns the location of the minimum value in aList between startIndex
       and endIndex"""
    minIndex = startIndex

    # scan down the unsorted part of the list to correct the assumption
    for testIndex in xrange(startIndex+1, endIndex+1):
        if aList[testIndex] < aList[minIndex]:
            minIndex = testIndex
    return minIndex

def selectionSortB(aList):
    for firstUnsortedIndex in xrange(len(aList)-1):
        # Call findMin to find the minimum item in unsorted part of list
        minIndex = findMin(aList, firstUnsortedIndex, len(aList)-1)

        # exchange the items at minIndex and firstUnsortedIndex
        temp = aList[firstUnsortedIndex]
        aList[firstUnsortedIndex] = aList[minIndex]
        aList[minIndex] = temp
```

a) Since the selectionSortB function only calls findMin, 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.)

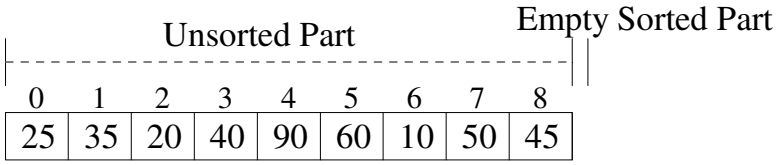
```
i = 1
while i <= len(myList):
    for j in xrange(i):
        myList[j] = 2 + myList[j]

    i = i * 2
```

Programming Part:

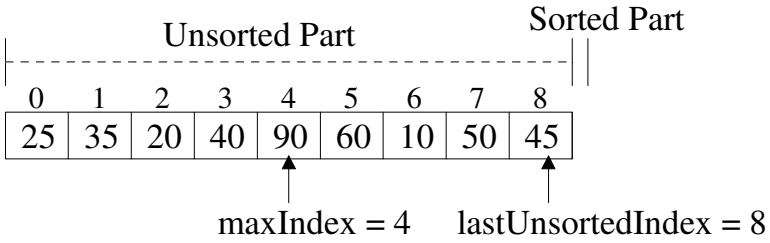
4. Rewrite the first selectionSort code (without findMin) so that it:

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

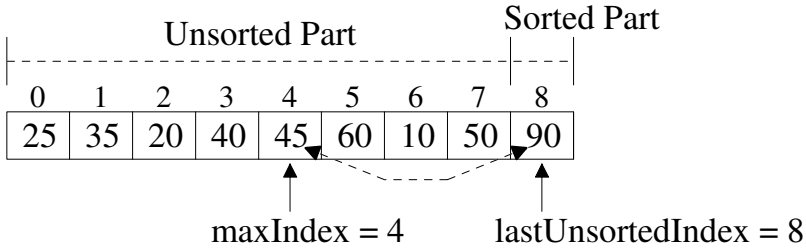
At the start of the first iteration of the outer loop the initial list is completely unsorted:



The inner loop scans the unsorted part and determines that the index of the maximum item, maxIndex = 4.



After the inner loop (but still inside the outer loop), the item at maxIndex is exchanged with the item at lastUnsortedIndex. Thus, extending the Sorted Part of the list by one item.



The algorithm is repeated until the whole list is sorted.

Just turn in a print-out ("hard-copy") of your program and the resulting output of sorting the above list.