3-5-08 Lab 7 CS III Name:_____

Objectives: To experiment with sorting to get a better feel for big-oh notation. To practice debugging skills.

Activity 1: Copy the folder P:\810-063-CSIII\LABS\LAB_7\ and run the timeSorts.py program. Studying the code, you'll observe that it creates a list of lists, called "lists", that holds 6 identical copies of a randomly generated list of integers -- one list for each sorting algorithm from the text: bubbleSort, shortBubbleSort, selectionSort, insertionSort, shellSort, and mergeSort. Each sorting algorithm is timed twice: first with the random array and then with the sorted array.

Initial timeSorts.py has the listSize set at 1000 items. Complete the following timings by changing the listSize and rerunning timeSorts.py. You need only record the timings on the random lists.

Timings on Random Lists (seconds)						
listSize	bubbleSort	shortBubbleSort	selectionSort	insertionSort	shellSort	mergeSort
1,000						
2,000						
4,000						
8,000						

Answer the following questions about the sorting algorithms:

a. Why does the bubbleSort algorithm take less time on a sorted list?

b. Why does the shortBubbleSort algorithm take less time on a sorted list?

c. Why is the selectionSort algorithm not effected by whether the list is sorted?

d. Why does the insertionSort algorithm take less time on a sorted list?

e. Why does the mergeSort algorithm take less time on a sorted list?

f. Using your knowledge of big-oh, predict how large of a list will require bubbleSort to execute for 60 seconds.

g. Check your prediction. What was the actual time for your prediction to part f?

h. Using your knowledge of big-oh, predict how large of a list will require mergeSort to execute for 60 seconds.

i. Check your prediction. What was the actual time for your prediction to part h? **(only run the mergeSort)**

After you have answered the above questions, raise your hand and explain your answers.

Activity 2: Run the timeQuickSort.py program that only times the quickSort algorithm imported from quickSort.py. Currently, this program sorts a list of 6 items. This is the same algorithm we taked about yesterday in class, but unfortunately it has an error that causes it to run indefinitely.

```
def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
    if first<last:
        splitpoint = partition(alist,first,last)
        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)


def partition(alist,first,last):
    pivotvalue = alist[first]

    leftmark = first+1
    rightmark = last

    done = False
    while not done:

        while leftmark <= rightmark and alist[leftmark] < pivotvalue:
            leftmark = leftmark + 1

        while alist[rightmark] > pivotvalue and rightmark >= leftmark:
            rightmark = rightmark -1

        if rightmark < leftmark:
            done = True
        else:
            alist[leftmark],alist[rightmark]= alist[rightmark],alist[leftmark]

    alist[first],alist[rightmark]= alist[rightmark],alist[first]

    return rightmark
```

Your goal is to discovery where the bug is and fix it! When debugging, it is always useful to localize where the error is occuring. The many ways to determine this are (1) add extra print statements to gain more information, or (2) use a debugger to step throught the program. With IDLE we could do either, but I usually prefer the first approach.

a. After you have debugged quickSort, how much time does quickSort take using a listSize of 8000?

b. Why does the quickSort algorithm take more time on a sorted list?

c. How does the timing for quickSort and mergeSort compare on randomly sorted lists?

After you have answered the above questions, raise your hand and explain your answers.

NOTE: If you complete all of the activities within the lab period, you do not need to hand anything it. However, if you need to finish activities outside of the lab period, then hand in written questions to activities not completed during the lab period.