CIT 244 Python II

Programming Project #3

INTRODUCTION

The goal of this programming project is to give the student practice of implementing two sort methods on data stored in lists in Python and comparing their performance. The student will conduct an experiment of comparing running times and data movements of the different methods. The student will then report the findings.

PROJECT TASKS

1. Read the problem definition below and write a Python program that implements your solution. Readability of the program will be graded based on variable naming, indentation, commenting (not too little and not too much), and styling in general including choice of functions.
2. Run experiments and report the performance of your implementation.

PROGRAMMING SKILLS TO BE USED

1. Declaring a list and assigning random numbers to **part** of the list.
2. Copying elements from one list to another
3. Calling standard library method of the random class to generate random numbers
4. Declaring and defining functions that take parameters of type list and return integers.
5. Using selection structures
6. Using nested loops that have selection structures inside.
7. Output formatting

PROBLEM

Write a Python program that has the following:

A) Standalone functions representing different sort methods as follows:

- **BubbleSort**
  Implement the unimproved version of BubbleSort algorithm whose pseudocode is:

```
for index = 0 to listSize – 1
    for position from 0 to listSize – index – 2
        if numbersList[position] is greater than numbersList[position + 1]
          swap the contents of numbersList[position] and numbersList[position + 1]
        end if
     end for
end for
```

  *** DO NOT USE THE LIST CLASS sort() METHOD ***

  The function receives a parameter that is the list of integers. In the function, initialize two variables to 0, one will keep a count of loops and the other will keep a count of swaps. The function sorts the list that was received as a parameter using a nested loop as in pseudocode.

Put a statement to **add one to the loop count** *just before* “if numbersList[position] is greater than numbersList[position + 1]” inside the inner loop block.

Put a statement to **add one to the swap count** *just before* the swap statement inside the block of the **if** statement.

Return the variables that store the loop and swap counts.

- **SelectionSort**:
  Implement the SelectionSort algorithm whose pseudocode looks as follows:

  ```
  for startScan is set to each subscript in list from 0 through the next-to-last subscript
      set index variable to startScan.
      set minIndex variable to startScan.
      set minValue variable to numbersList[startScan].

      for index is set to each subscript in list from (startScan + 1) through the last subscript
          if numbersList[index] is less than minValue
              set minValue to numbersList[index].
              set minIndex to index.
          end if.
      end for.
      swap the values subscripted by startScan and minIndex.
  end for.
  ```
  *** DO NOT USE THE LIST CLASS sort() METHOD ***

  The function receives a parameter that is the list of integers. In the function, initialize two variables to 0, one will keep a count of loops and the other will keep a count of swaps. The function sorts the list that was received as a parameter using a nested loop.

  Put a statement to **add one to the loop** count *just before* “if numbersList[index] is less than minValue” inside the inner loop block.

  Put a statement to **add one to the swap count** *just after* swap statement as the last statement inside the outer loop block.

  Return the variables that store the loop and swap counts.

B) The **main()** function:

In the main() function, declare an empty list. You should declare other variables that will store counters. You need 4 counter variables, two for the loop count for each sort type respectively and 2 for swap count for each sort type respectively.

In the main function, ask a user how many numbers should be generated. Read the number into a variable which you may call **UserNumOfNumbers**. Check to ensure the number entered is between 10 and 20000. The default should be 10000 numbers (user should be able to say, “choose a number for me” in which case you assign 10000 to **UserNumOfNumbers**).

Use the **random()** method found in the **random** standard library to generate **UserNumOfNumbers** positive integers (use a **for** loop) and append them into the list.

Copy the list of generated numbers into the second list then call BubbleSort passing the second list. Assign the returned integers to the bubblesort loop count and swap count variables.

Copy the list of generated numbers into the second list again and then call SelectionSort passing the second list as a parameter. Assign the returned integers to the selectionsort loop count and swap count variables.

Finally, display the counts that were returned by the different sort functions using **format** for column widths as follows:

```
SORT ALGORITHM RUN EFFICIENCY
ALGORITHM          LOOP COUNT    DATA MOVEMENT
Bubble              xxxxxxxxxx       xxxxxxxxxx
Selection           xxxxxxxxxx       xxxxxxxxxx
```

REPORT

Run your program several times and keep a record of the output of each run. For each run, you should change the number of integers to be generated. For example, you may input increments of 2,000 from 2,000 to 10,000. Enter the results in a spreadsheet, one table per sort method. Use the spreadsheet software chart feature to plot 4 graphs for: loop counts for BubbleSort, SelectionSort, and swap counts for BubbleSort, and SelectionSort. If you use Excel for your charts, you may add a trend line for each plot.

Identify which sort type requires more loops and which requires less loops especially as the number of data values increases. You should also identify the sort type moves data a lot and which moves data less using the swap counts.

Search the World Wide Web BubbleSort and SelectionSort efficiency and explain whether your results agree or disagree with the theoretical performances of these sort methods.

SUBMISSION

a) Upload to Blackboard a copy of your report in Word or PDF (copy your charts into your Word document).
b) Upload to Blackboard a copy of your Python program.

MAXIMUM POSSIBLE SCORE

Below is the grading rubric for this project. The rubric is in the context of a program written to satisfy the core object of this project and that runs without syntax errors. If a project works but does not have the code to satisfy the core objective or runs with errors, it will automatically be awarded a score of 0.

This program will be graded out of 60 points distributed as follows:

ITEM	MAX. POINTS
Report	20
Style: comments, meaningful names, indentation	10
Program written to specification	15
Program works correctly	15