Math 590 Project 1: Sorting Fall 2018

Due: Friday, November 2 at Midnight

In this project, you will implement a number of sorting algorithms that we have discussed in class. You will be required to implement your solutions in their respective locations in the provided `project1.py` file.

1 Pair Programming

You are allowed to work in pairs for this project. If you elect to work with a partner:

- You should submit only one version of your final code and report. Have one partner upload the code and report on their Sakai site. Make sure that both partner's names are clearly indicated at the top of both the code and the report.

- When work is being done on this project, both partners are required to be physically present at the machine in question, with one partner typing ('driving') and the other partner watching ('navigating').

- You should split your time equally between driving and navigating to ensure that both partners spend time coding.

- You are allowed to discuss this project with other students in the class, but the code you submit must be written by you and your partner alone.

2 Style Points

Part of your grade for this project will be 'style points'. The idea here is that the code you turn in must be well commented and readable. A reasonable user aught to be able to read through your provided code and be able to understand what it is you have done, and how your functions work. For these sorting algorithms, this means that a grader should be able to read over your code and tell that your algorithms are implemented correctly.

The guidelines for these 'style points':

- Your program file should have a header stating the name of the program, the author(s), and the date.

- All functions need a comment stating: the name of the function, what the function does, what the inputs are, and what the outputs are.

- Every major block of code should have a comment explaining what the block of code is doing. This need not be every line, and it is left to your discretion to determine how frequently you place these comments. However, you should have enough comments to clearly explain the behavior of your code.

- Please limit yourself to 80 characters in a line of code. In python, you can use the symbol \ to indicate a line break/continuation.

3 Python Installation

We will be using Python version 3.7.1 for this class, which you can find here:

`https://www.python.org/downloads/`

Once you have installed python, you will need to install `numpy`, `scipy`, and `matplotlib`. For Mac and Linux machines, you should be able to simply open a terminal and run the command:

`python -m pip install numpy scipy matplotlib`

For Windows machines, you will need to first find the location of your Python 3.7 installation. If you did not change the path during installation, the path should be similar to:

`C:\Users\Eric\AppData\Local\Programs\Python\Python37`

If you want to verify this location, you may have to show hidden files and folders (since the `AppData` folder is often default to hidden).

Once you have located your python installation, you will need to open a command prompt. Do this by typing `cmd` into the start menu's search (or on older systems, by selecting start→run and typing `cmd` there). Once the command prompt is open, navigate to your python 3.7 directory using the `cd` command, and then run the installation commands, i.e.:

```
>cd C:\Users\Eric\AppData\Local\Programs\Python\Python37
>python.exe -m pip install numpy scipy matplotlib
```

If you have any problems with installing python, please see the professor or a TA for assistance. You can come to office hours, or email me directly at `eric.autry@duke.edu` to arrange a time to meet.

4 Python Tutorials

For those of you new to python, there are several possible tutorials you can use.

- Drew Hilton from the ECE department has a pdf tutorial for python at:

  `https://adhilton.pratt.duke.edu/files/u37/py-intro.pdf`

- `PyCharm` is a python IDE available for download, and has included tutorials that should only take a couple hours to complete (though you will likely not need all of the provided information for this project).

5 Provided Code

The file `project1.py` has a number of pre-defined functions listed below:

- `InsertionSort(A)`: this will contain your implementation for Insertion Sort.

- `SelectionSort(A)`: this will contain your implementation for Selection Sort.

- `BubbleSort(A)`: this will contain your implementation for Bubble Sort.

- `MergeSort(A)`: this will contain your implementation for Merge Sort.

- `QuickSort(A,i,j)`: this will contain your implementation for Quick Sort. Note that the values of the indices `i` and `j` are **required inputs**, though they do not have to be used in your implementation. All sorting with this function will be done with the function call `QuickSort(A,0,len(A))`. If you wish, you can define a new function to implement Quick Sort, and simply call that function from this provided `QuickSort(A,i,j)`, i.e.,

  ```
  """
  myQuickSort
  """
  def myQuickSort(A):
      code
      return A


  """
  QuickSort

  Sort a list A with the call QuickSort(A, 0, len(A)).
  """
  def QuickSort(A, i, j):
      return myQuickSort(A)
  ```

- `isSorted(unA, sA)`: this function returns True if the array `sA` is the sorted version of the array `unA`, and False otherwise.

- `testingSuite(alg)`: this function runs a number of tests on the algorithm in question. This is not an exhaustive list of tests by any means, but covers the edge cases for your sorting algorithms. The valid inputs to this testing function are the strings:

  - 'SelectionSort'
  - 'InsertionSort'
  - 'BubbleSort'
  - 'MergeSort'
  - 'QuickSort'

- `measureTime(sortedFlag = False, numTrials = 30)`: this function runs your algorithms on a number of randomized inputs of varying sizes while tracking the runtimes. It will plot the runtime versus n for each algorithm and save these as `.png` files to the current directory. It also creates a log-log plot of the runtime for several of the algorithms, and will print some info about those plots. This function will be discussed in further detail later.

6   Problem Statement

You will implement five algorithms for sorting an array. We define this sorting task as:

- The input array will have a positive integer length.

  - Will not be asked to sort an empty array.
  - Could be asked to sort a singleton array.

- The elements need not be distinct (i.e. repeats allowed).

- The elements can be any (positive or negative) real number.

The goal is to sort the elements from the input array into ascending order so that the smallest element is at index 0, i.e.,

$$A[0] \leq A[1] \leq \cdots \leq A[k-1] \leq A[k] \leq A[k+1] \leq \cdots \leq A[end]$$

You will implement the following five algorithms as discussed in class:

- Selection Sort

- Insertion Sort

- Bubble Sort

- Merge Sort

- Quick Sort

Most of your grade for this project will be the correctness of these algorithms.

6.1   Runtime Comparisons

In the provided code is the function `measureTime(sortedFlag = False, numTrials = 30)`. This function can be used to time your implementations and obtain plots of the runtime versus input size. There are several ways this function can be called:

- The call `measureTime()` will use randomly generated test arrays, and for each input size n considered, will average the runtime over 30 separate trials.

- The call `measureTime(False,x)` will use randomly generated test arrays, and for each input size n considered, will average the runtime over x separate trials, where x is an integer.

- The call `measureTime(True)` will use already sorted test arrays, and for each input size n considered, will average the runtime over 30 separate trials.

- The call `measureTime(True,x)` will use already sorted test arrays, and for each input size n considered, will average the runtime over x separate trials, where x is an integer.

Once the runtimes have been determined, a number of plots are created so you can see directly the runtime versus the input size.

6.2   Log-Log Runtime

The function `measureTime` will also generate a log-log plot of runtime versus input size for Selection Sort, Insertion Sort, and Bubble Sort. It then attempts to fit a line to these log-log plots and will output the fitted slope. It first attempts this fit using all of the runtime data (including even very small values of n) and will report those slopes. It then attempts the fitting using only larger values of n. One of the questions you must answer is which attempted fit gives more accurate results, and why you think that is the case.

Looking at log-log plots of the runtime versus the input size is a very common method used to interpret the runtime of a polynomial time algorithm. To understand why we do this, consider the following hypothetical:

Let's assume that we have an algorithm that runs in $O(n^k)$ time, where k is some positive integer. Once we have implemented this algorithm, how can we determine if it really has obtained the promised runtime complexity?

The claim that the runtime is $O(n^k)$ means that

$$T(n) \sim n^k.$$

Now take the log of both sides of this equation. This gives us

$$\log T = \log(n^k).$$

But, by a property of logarithms, we can write this as

$$\log T = k \log n.$$

So if we plot $\log T$ versus $\log n$, we should see a straight line! Moreover the slope of that line should be the value of k.

This means that if you want to verify that an algorithm runs in $O(n^2)$ time, you should plot the runtime versus the input size on a log-log scale. If you fit the resulting log-log plot to a straight line, the slope should be 2.

6.3 Questions

Along with your code, you should submit a short report (1-3 pages) that addresses the following questions. Your report can include any of the generated plots that *you deem relevant*. You are not required to include any of the images. Note that you should not directly answer these questions one at a time, but rather your report should discuss their answers with details/evidence obtained from the results of running your code.

- Do your algorithms behave as expected for both unsorted and sorted input arrays?

- Which sorting algorithm was the best (in your opinion)? Which was the worst? Why do you think that is?

- Why do we report theoretical runtimes for asymptotically large values of n?

- What happens to the runtime for smaller values of n? Why do you think this is?

- Why do we average the runtime across multiple trials? What happens if you use only one trial?

- What happens if you time your code while performing a computationally expensive task in the background (i.e., opening an internet browser during execution)?

- Why do we analyze theoretical runtimes for algorithms instead of implementing them and reporting actual/experimental runtimes? Are their times when theoretical runtimes provide more useful comparisons? Are their times when experimental runtimes provide more useful comparisons?

7 Submission

You **must** submit your `project1.py` code and your report online on Sakai. If you worked with a partner, only submit one version of your completed project and indicate clearly the names and NetIDs of both partners.