Computer Science I CSCI 141
Quiche Sort and Other Holiday Treats
Homework (due end of Monday 12/3/13)

11/25/2013

1 Problem

Did you know the word *quiche* derives from the Lorraine Franconian dialect for *Küeche* which means *cake*? In this assignment you can truly have your cake and eat it too.

This homework will have you experiment with some minor improvements to the quick sort algorithm. You may know that it is impossible to create a sorting algorithm that runs any faster than, $O(n \times log(n))$, in worst case, if all you have to work with is a comparison function. But it is possible to speed things up on a smaller scale if we try.

The first experiment is to try a different way of choosing the pivot. Choosing the very first value in a list is a terrible choice, if the list is already sorted (or reverse sorted). So to improve our chances of evenly dividing the list, we will choose a pivot from among the first, middle, and last elements in the list. Whichever of those three is the median value, i.e., one other value is smaller and the remaining value is larger, is the one that will be chosen. Note that our use of the words *first*, *middle*, and *last* have nothing to do with where those values would appear in the sorted list. The terms refer to their *positions* in the original list. This technique is called **Median-of-three**.

The second experiment is to try switching to a heap-based sort if quick sort is not doing too well. A program could detect this inferior behavior if the depth of the recursive calls is getting beyond the expected $log(n)$, which is the limit when the list is always partitioned evenly. So when quick sort has called itself $log(n)$ times *without returning*, the two lists resulting from partitioning will be passed to a heap sort function for further processing. The two calls to the heap sort function will return one list each. Those two lists will be concatenated as per normal quick sort operation, and the quick sort code resumes execution.

The name of this hybrid algorithm was derived from its two parts, "quick" and "heap". Eliminating some letters at each name's end, we end up with **Quiche sort**.

1.1 Setup

Fetch all of the Python files from: the directory containing this homework description document

You will have all of the following.

- Full implementations of an array-based heap and quick sort from lecture:
  - **arrayHeap.py** - Renamed from `array_heap.py` from Wk 13 lecture.
  - **qsPivotFirst.py** - Renamed from `quickSort.txt` from Wk 14 lecture.
- **testSorts.py**, a test program for your finished system
- Skeleton code files for the three things you must implement (in this suggested order):
  1. **qsPivotMedian3.py** – where you write a modified quicksort to choose as the pivot the median of the first, middle, and last elements in the list. For an even length list, there are technically two *middle* values, so just choose one.
  2. **heapSort.py** – where you use the functions in **arrayHeap.py** to create a heap-sort implementation (Don't forget that the array returned has to be a copy, and that the original list is unchanged.) You should review the lecture code for Heaps and pay attention to the `heapsort` algorithm on page 9.
  3. **quicheSort.py** – where you modify the quick-sort code you developed in **qsPivotMedian3.py** to also switch to heap-sort when the recursion depth exceeds the logarithm base 2 of the size of the original list (The logarithm function can be found in the `math` module).
- **output.txt**, these are our sample outputs when running the various sorting algorithms on our UNIX machines. The times here are all relative to the machine we are running on. You may get better or worse results depending on what machine you run on. The important thing to note is the test cases when the recursion limit is exceeded. You should not manually adjust the runtime stack size, so you should see similar behavior with those tests.

1.2 Implementation

Implement your sorting code in the order that the skeleton files were listed above. Pay attention to the materials already contained in those files; they are important tips as to

how to implement the sorts.

Note that there is some test code at the end of each file. This means that, if you run one of these files by itself, the part of the overall test program that is appropriate for the file that is run will execute.

The test program does the following. For each type of sort, the sort function is run on differing sizes of data and different orderings of that data. The sort function's execution is timed, so you will see the execution time, in seconds, of each test run. Each test case ends with code to make sure that the list is indeed sorted. Make sure you see"`True`" on each of those lines.

Some list size / data type combinations have been pre-excluded because they are known to cause a stack overflow (from excessive recursion). Beyond this, some tests may cause a "`RUNTIME ERROR!`" due to stack overflow. It is up to you to decide if this is a plausible outcome of your test, based on your implementation. If you ever see a "`MEMORY ERROR!`" there is probably a bug in your code. Contact a student instructor, TA, or faculty, if you're not sure.

1.3 Report

Just so we can see the variety of platforms on which these tests are run, please also include a *plain text* file named `report.txt` that includes the following information.

1. Make and model number of computer
2. CPU and clock speed
3. Amount of memory in your computer
4. Operating system name and version
5. Version of Python used

The first piece of information should be available on a label on your computer. The second, third, and fourth pieces should be available from a menu item with a name like "System Properties" or "About This Computer". On many UNIX/Linux-like systems, you can type "`uname -a`" in a shell to get much of it. The fifth piece of information appears whenever you run a Python program or invoke a "Python shell".

1.3.1 Getting System Information - Windows 7

First, go to `Start -> Control Panel -> System and Security -> System`.

1. To get the make and model number of your computer, go to `Device Manager -> [ComputerName] -> Computer`.
2. To get the processor information, backup to `System` and look under `Processor`.
3. To get the memory information, also look under `Processor` for the field `Installed memory (RAM)`.
4. To get the operating system information, go to `Windows Edition + System -> System Type`.
5. To get the version of Python3, run idle3 and look at the top line of text in the window.

1.3.2 Getting System Information - OS X (Mac)

First, select the Apple/Mac icon in the upper left and select `About This Mac`.

1. To get the make and model number of your computer, click on the `More Info...` button. Next click on the `System Report...` button. Look under `Hardware` and record these two pieces of information under `Model Name` and `Model Identifier`.
2. To get the processor information, back up to `About This Mac` and look under `Processor`.
3. To get the memory information, look under `Memory`.
4. To get the operating system information, look directly above the `Software Update...` button (don't click it).. There should be a huge `OS X` followed below by the `Version` number.
5. To get the version of Python3, run idle3 and look at the top line of text in the window.

1.4 Grading

- 20%: Correct implementation of quick sort Median-of-three
- 30%: Correct implementation of heap sort from arrayHeap
- 40%: Correct implementation of quiche sort
- 10%: Submission of `report.txt`

1.5 Submission

Submit your solution as a zip, `lab13.zip`, of four files called `report.txt`, `heapSort.py`, `qsPivotMedian3.py`, and `quicheSort.py` to the MyCourses dropbox for this assignment.