CS/IT 200

Lab 12: Sorting

Due Date: Sunday, December 8, 11:59pm (Night before Finals Week)

Submission

- Submit all code and other materials in a single zip file to the Lab 12 assignment folder on Kodiak.

Goal: Analyze and report the running times of sorting algorithms.

Part I

The sorting module distributed for this lab contains code for Selection Sort, Insertion Sort, Bubble Sort, Merge Sort, and Quick Sort. Using that class, write a series of small programs to measure the efficiency of the various sorting algorithms for different sizes of N using lists (the built-in Python object).

(a) Gather measurements of the following for each of the five sorting algorithms in the `sorting` module, using the `timeit` module. For each subpart i-iii, collect data points for at least the following values of N = 100; 1000; 10,000; 100,000. Place this code in `lab12.py`.

   i. Create a list of random integers and sort it. The range of random values should be between 0 and 10*N.
   ii. Create a list of integers that is already sorted and sort it.
   iii. Create a list of integers that is sorted backwards and sort it.

   If any condition produces a `RuntimeError` or `RecursionError`, report ERR instead of a time.

   Hint 1: Make sure that you are starting with an appropriate list prior to each sorting test. Don't accidentally use a sorted list from the previous trial where you should be using a random list.

   Hint 2: The `default_timer()` function in the `timeit` module can be used to time a block of code.

```
start_time = timeit.default_timer()
# YOUR CODE TO GET TIMED GOES HERE
end_time = timeit.default_timer()
time_elapsed = end_time - start_time
```

(b) Present your data in a table that makes it easy to compare the difference between sorting algorithms for each of the subparts.

(c) Compare and contrast the relative performance of the sorting algorithms for the different operations. Which one is faster/slower? Under which conditions? Do your observations match the complexities we have discussed in class? Which conditions, if any led to `RuntimeErrors`? What was the cause of these errors? Why did those conditions lead to these errors?

WARNING: Give yourself plenty of time to take these measurements. Execution of N=100,000 may take a long time to finish. Have other work handy, or write your programs so they can run while you take a nap. In the past, some students ran this code overnight.

Part II

The `_choose_pivot()` function selects a pivot point for the Quick Sort algorithm. (If we have not yet discussed this in class, you may wish to review that algorithm). Currently, the pivot is whatever is already in `mylist[b]`, where `a` and `b` are the edges of the section of the list that we are looking at.

A different way to choose a pivot is to take the **median** value of the three values `mylist[a]`, `mylist[b]`, and `mylist[(a+b)//2]`. Modify `_choose_pivot()` to select a pivot in this way. The `_choose_pivot()` function should return the index where the median value is located.

Reminder: Determine the median based on the ***values*** in the array, not the indices themselves, but **return the index**. In the diagram below, the median value is 17, because 8 < 17 < 21, even though 21 is in the `(a+b)//2` slot. Since the median is 17, we would want `_choose_pivot()` to return `a`.

	a		(a+b)//2		b	
...	17	...	21	...	8	...

Collect and report the same data that you did for Part I (a) for this Modified Quick Sort and add it to your table from Part I (b).

Answer the following questions:

- How did Modified Quick Sort compare to the original quick sort? Did it ever succeed when the original algorithm failed, or vice versa?

Part III

We discussed (or will discuss) the Radix Sort algorithm, which is substantially different from the other sorting algorithms.

Add an implementation of radix sort to the sorting module that was distributed with this lab. Your implementation can assume a list of integers. Name the function `radix_sort()`. Its only parameter should be a list.

Collect and report the same data that you did for Part I (a) for Radix Sort on lists of integers. Report the data in the same table as Part I and II.

What to Submit:

- Your code (`lab12.py` and `sorting.py`, plus any additional modules that you used)
- A write-up that includes your table of data and your answers to the questions.