Homework 4

COS226 Fall2024

DUE: April 4th 2024

Goals

Implement several sorting algorithms. Benchmark them on different data.

Task

For this task you will implement three sorting algorithms: selection sort, quicksort and counting sort. These three represent three different approaches to sorting, *comparison*, *divide-and-conquer* and *non-comparison* respectively. Once implemented you will use them to sort several lists of integers, collecting data about each one in the process.

Details

Each sorting algorithm should be in a file by itself: selection.py, quicksort.py and countsort.py.

You must provide a function that verifies A) the sorting algorithm does produce a sorted list in ascending order and B) the sorted list is the same size as the original. You must use this function to demonstrate that all your sorts work properly.

Steps

1. You will create a list of 1000 random positive integers sampled from the range [1,1000] and use each sorting algorithm to sort the list. Once sorted, you will use the verification algorithm above to demonstrate that your algorithms work. NOTE: This is just to show your sorting algorithms work. This isn't the benchmarking step, so you can use a different list for each one.
2. You will do the following 100 times. Each time you will record the time (in nanoseconds) it takes for your sorting algorithms to sort each list. Once done, you will print the min, max and average times for each list size.
3. You will generate three lists of random positive integers of size 100, 10,000 and 1,000,000. All lists will be randomly sampled from a uniform distribution over the range [1,1000]. This means your lists can or will have duplicate values. This is expected.
4. Sort each list using all three algorithms. In each iteration (from step 2), use the same lists for all three algorithms. This means you will need to make nine total lists (3 of each size).
5. Record all nine sorting times and store them.
6. Once all 100 sorts have been accomplished, find the min, max and average times for each algorithm for each list size. Print these in a nice table, like the one below. Except where you see *time* below, you should put your times in nanoseconds.

Additional Notes:

Remember to time only the call to your sorting algorithm. Don't leave debug prints in your code as they will skew the times. Try to implement your sorting algorithms efficiently.

Implement the algorithms according to the book's pseudocode. However, the book (to my knowledge) doesn't contain selection sort. You can find pseudocode here: https://faculty.cs.niu.edu/~hutchins/csci241/sorting.htm

Do the hard work of implementing from the pseudocode. Resist the temptation to nab full solutions from the net. If you're serious, it's good practice.

Example

```
VERIFICATION
Algorithm   SORTED   SIZE
Selection:  true     true
Quicksort:  true     true
Countsort:  true     true

MIN SORTING TIME
Algorithm   100     10K     1ML
Selection:  time    time    time
Quicksort:  time    time    time
Countsort:  time    time    time

MAX SORTING TIME
Algorithm   100     10K     1ML
Selection:  time    time    time
Quicksort:  time    time    time
Countsort:  time    time    time

AVG SORTING TIME
Algorithm   100     10K     1ML
Selection:  time    time    time
Quicksort:  time    time    time
Countsort:  time    time    time
```