Queens College of CUNY
Department of Computer Science
Design and Analysis of Algorithms (CSCI 323/700)
Summer 2024

Assignment #2:
"Sorting Algorithms"
Due: June 7, 2024

Introduction:

The previous assignment focused on "search algorithms" that find a numeric key in an ordered array, and the tabulation and visualization of the empirical performance of such algorithms.

In this assignment, we turn our attention to sorting algorithms. As mentioned in our introduction to the topic, sorting algorithms are important both inherently as solutions to a real-life application, and for their role in the design and analysis of other algorithms. More specifically, you are to write a program that executes numerous sorting algorithms, in addition to the built-in one from your programming language.

1. Native Sort (not the name of a sorting algorithm, rather your language's built-in sort)
2. Bubble Sort (https://www.geeksforgeeks.org/bubble-sort/)
3. Selection Sort (https://www.geeksforgeeks.org/selection-sort/)
4. Insertion Sort (https://www.geeksforgeeks.org/insertion-sort/)
5. Binary Insertion Sort (https://www.geeksforgeeks.org/binary-insertion-sort/)
6. Cocktail Sort (https://www.geeksforgeeks.org/cocktail-sort)
7. Shell Sort (https://www.geeksforgeeks.org/shellsort/)
8. Merge Sort (https://www.geeksforgeeks.org/merge-sort/)
9. Quick Sort (https://www.geeksforgeeks.org/quick-sort/)
10. Quick Insertion Sort (Quick Sort, with Insertion Sort when subarray is 10 or less)
11. Heap Sort (https://www.geeksforgeeks.org/heap-sort/)
12. Counting Sort (https://www.geeksforgeeks.org/counting-sort/)
13. Bucket Sort (https://www.geeksforgeeks.org/bucket-sort-2/)
14. Radix Sort (https://www.geeksforgeeks.org/radix-sort/)
15. Tim Sort (https://www.geeksforgeeks.org/timsort/)
16. Comb Sort (https://www.geeksforgeeks.org/comb-sort/)
17. Bingo Sort (https://www.geeksforgeeks.org/bingo-sort-algorithm/)
18. Pigeonhole Sort (https://www.geeksforgeeks.org/pigeonhole-sort/)
19. Cycle Sort (https://www.geeksforgeeks.org/cycle-sort/)
20. Stooge Sort (https://www.geeksforgeeks.org/stooge-sort/)
21. Gnome Sort (https://www.geeksforgeeks.org/gnome-sort-a-stupid-one)

You are permitted to utilize publicly available code for the individual sorts and integrate them into one program, following the working model we developed in class for search algorithms in Assignment #1. You do not necessarily have to use the "Geeks for Geeks" (G4G) versions linked above. Other good sites are:

- https://brilliant.org/wiki/sorting-algorithms/
- https://rosettacode.org/wiki/Category:Sorting_Algorithms
- https://www.programiz.com/dsa
- https://realpython.com/sorting-algorithms-python/

The Wikipedia articles on these algorithms also are helpful in terms of understanding, illustration, and references to code.

Submissions:

In the Google form, please submit the following files:

- Assignment2.py (source code)
- Assignment2.txt (console output including table)
- Assignment2.png (bar graph)

Tasks:

[0] Install Python and the PyCharm IDE as outlined in Assignment #0, and follow the other preliminary assignment setup tasks described in Assignment #1

[1] Write a function *random_list(range_max, size)* that returns a list of *size* random numbers in range(1, *range_max*). For this assignment, (a) do not sort the random list upfront, and (b) do allow duplicates

[2] After generating the random list, you will need to make a copy of it for each algorithm. Otherwise, once you sort it for one algorithm, it is already sorted! For how to copy an array, see https://docs.python.org/3/library/copy.html

[3] Write or call a function that will confirm that the list has been sorted. If it detects an error, make sure to print it very conspicuously (or throw an exception) so that the error can be researched.

[4] For each of the thirteen aforementioned sorting algorithms, write a function that implements it. You can copy the code from an online source as long as you acknowledge it with a comment just before the function. Keep the signatures consistent (same arguments in the same positions) so they can all be called in the same fashion as shown in class..

[5] In your main function, iterate over all the sorts for the same list size. Capture the time before and after so you can track the elapsed time for the sorting algorithm. [Repeat these runs multiple times for the same size and find their average time.]

[6] Repeat these runs for lists of different sizes: n = 10, 100, 1000

[7] Summarize the results in a table using Pandas Dataframe or equivalent. Place the sorting algorithms along the left side and the different sizes across the top.

[8] Visualize the statistics in a graph using matplotlib or another graphic package. Place all sorts in the same graph for easy comparison. Use a different color for each sort and create a color legend either in or below the graph.

For Counting Sort, use this code instead of the one linked above:

```
def counting_sort(arr):
    n = len(arr)
    mx = n
    output = [0 for _ in range(n)]
    count = [0 for _ in range(mx)]
    for a in arr:
        count[a] += 1
    for i in range(n):
        arr[i] = output[i]
```