CS 3753 HW1 (60 + 3 pts)

Electronic submission due 11:59pm, Sat Feb 5, 2022

Submission:.py or .ipynb file in blackboard. Those who submits the .ipynb file will receive 5 * (x-1) bonus points for HWx. x = 1 for HW1.

To solve this homework, simply download the .py or .ipynb code skeleton provided in blackboard, write your own code to replace parts that are marked as

```
#######
Pass
#######
```

Note: Do NOT use numpy or pandas for this homework.

1. (20 points) Python implementation of selection sort.

Complete the function selectionSort. Use any reference that is necessary. Selection sort - Wikipedia

Test your function on a small random integer array of 10 elements for correctness with the code already provided in the script. (Observe the speed difference between your selectionSort function and python built-in function “sorted" – which is an implementation of quick sort. FYI my implementation of selection sort takes about 10 micro seconds on this, 25 times slower than the 400 nano seconds for python quick sort.)

2. (20 points) Python implementation of merge sort. Use any reference that is necessary. Merge sort - Wikipedia (top-down using lists)
    a. Complete the function merge, which takes two lists of numbers, both already sorted, and merge into a single list whose elements are in sorted order. Test your function a two small lists for correctness with the code in the script.

    b. Complete the function mergeSort, which takes a list of numbers in random order and returns a sorted list. Test your function on a small random integer array of 10 elements for correctness using code provided in the script. (Observe the speed difference between your mergeSort function and python builtin function “sorted". FYI My merge sort takes about 30 micro seconds for this.)

3. (20 points) The script included a block of code that runs the three algorithms (selectionSort, mergeSort, and sorted) on six inputs of size ranging from 500 to 16,000. Running time (in milliseconds) of each algorithm is collected for each input size. Use the values collected in the three variables, selection_sort_time, merge_sort_time, and quick_sort_time, to make four plots. The first two plots are plotting size against running time. The next two plots are plotting size against (running time / size), i.e., the average time spent on each element in the list. Try to

reproduce all the details (such as title, axis labels, legends, line color/style and marker shape) of the figure shown on the next page. Note the different axis scaling in the figures.

(Note: if for some reason your experiment runs much slower than a few minutes, try to modify the input sizes to smaller values. Your plot will look somewhat different than the attached figures, but should show similar patterns. There can be many reasons why your algorithm might be slower than expected, but you may want to avoid using the "append" function for list.)

4. **Bonus questions for fun** (**3 points**): (Provide your answers in blackboard submission text box.)
    a. Which figure provides the strongest/clearest evidence that Selection Sort is in Θ(n^2) and why?
    b. Which figure provides the strongest/clearest evidence that Quick Sort and Merge Sort are in the same asymptotic order and why?
    c. Which figure provides the strongest/clearest evidence that Quick Sort is NOT a Θ(n) algorithm and why?

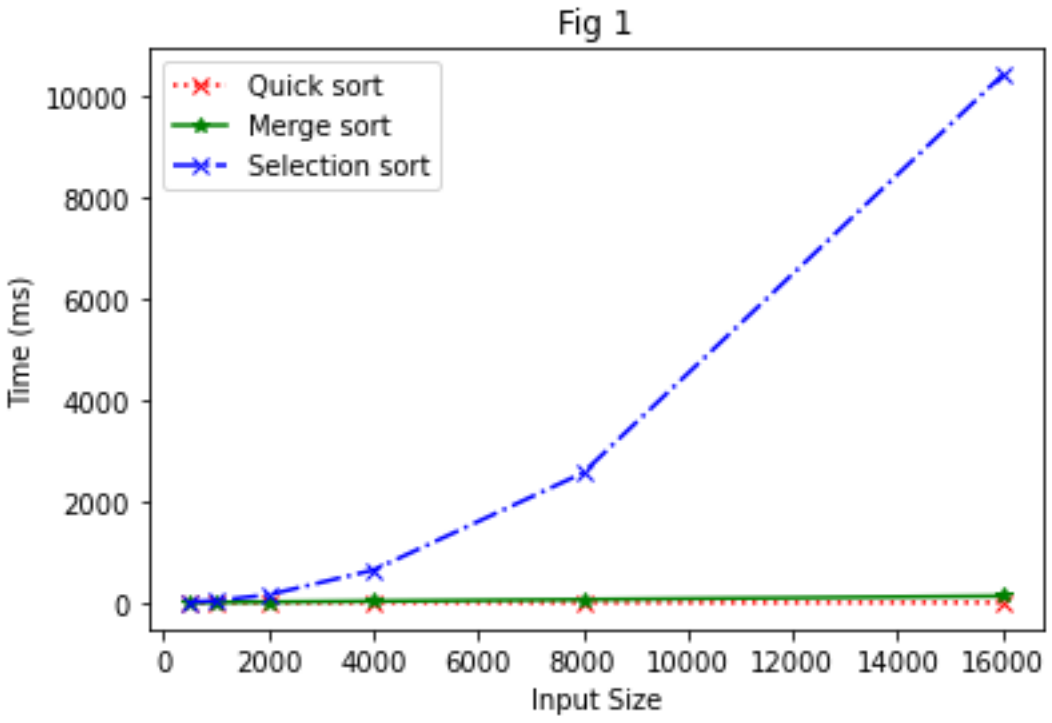


Fig 2

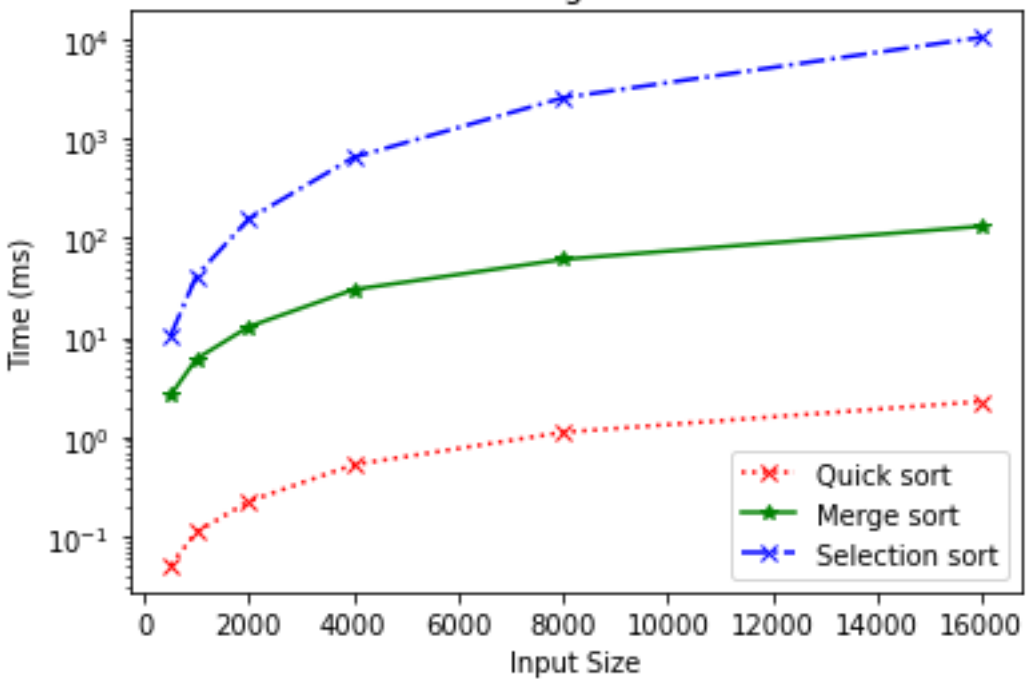

Time (ms)
10^4
10^3
10^2
10^1
10^0
10^{-1}
0
2000
4000
6000
8000
10000
12000
14000
16000
Input Size
Quick sort
Merge sort
Selection sort

Fig 3

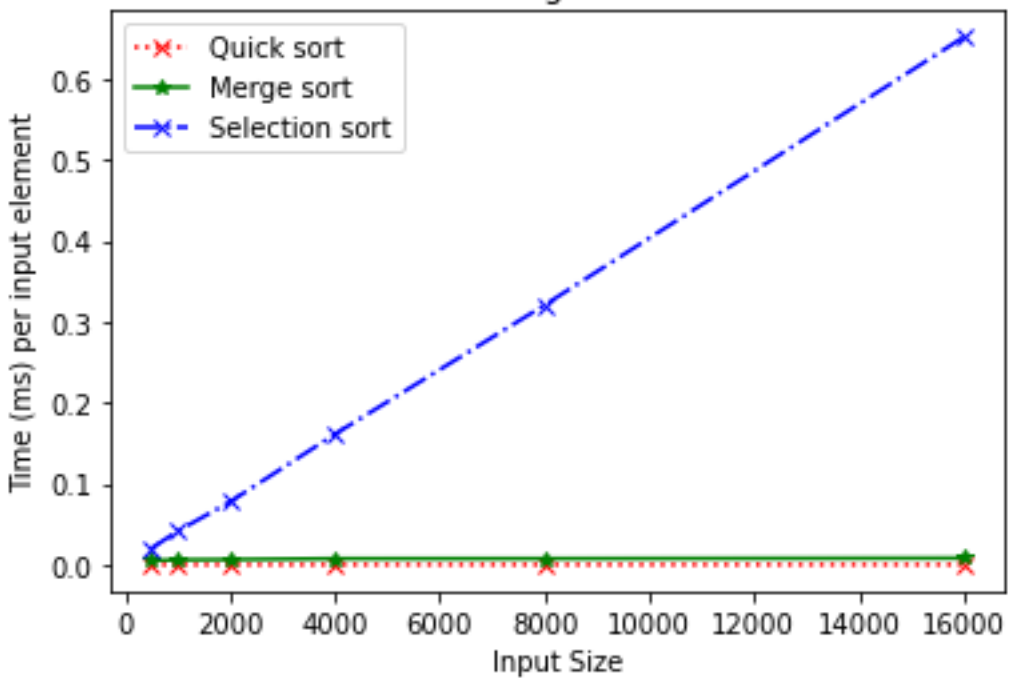
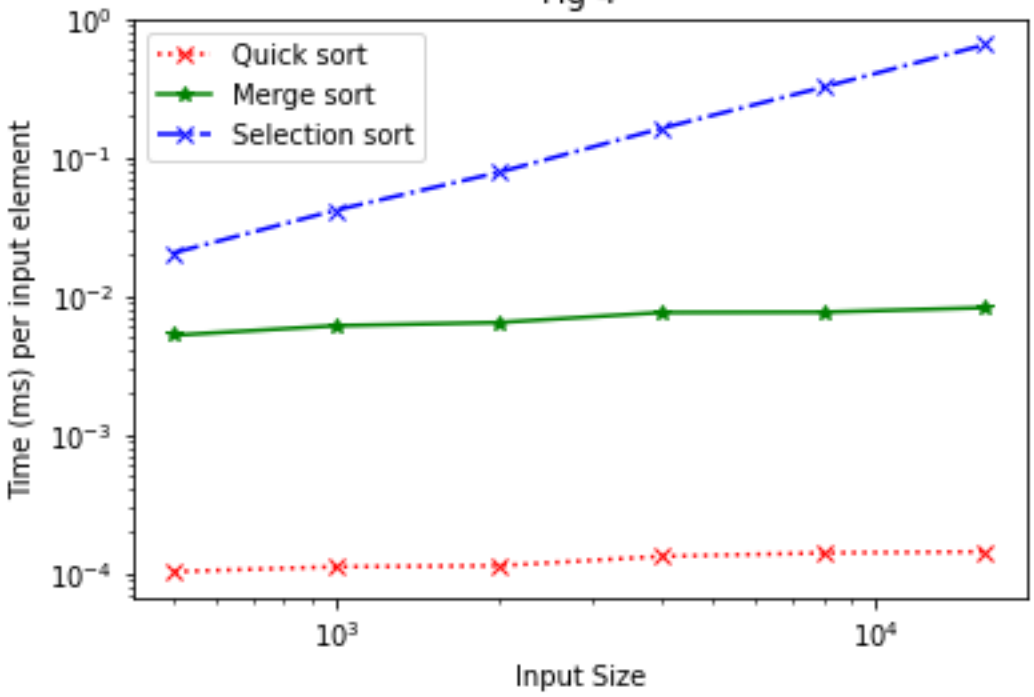

Time (ms) per input element
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000
12000
14000
16000
Input Size
Quick sort
Merge sort
Selection sort

Fig 4


Quick sort
Merge sort
Selection sort
Time (ms) per input element
Input Size
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^3
10^4