Jordan Carson
Lab 4 Analysis
October 12th, 2022

Introduction

In this repository, I have implemented several helper functions, algorithms, to perform the duties of this Lab 4 for 605.202 Data Structures at JHU. The program creates the data files, as per the requirements, and writes to a directory called *include.* The purpose for me doing this myself is that the provided input files were in various formats, some had all the data in 1 row, while others had the data in 1 column. We then load the data, sort the data, based on the given algorithm being used and creates a dictionary to store the results.

I have done a lot of work comparing iterative vs recursive solutions during this lab and decided that recursive solutions are more natural than an iterative solution for sorting. It makes sense that the results were nearly identical. For the algorithms we created, there are four different variations: simple quicksort, quicksort with a stopping partition of 50, quicksort with a stopping partition of 100, and quicksort using a median-of-three. There was a lot of work required to get quicksort median to work properly. In fact, in the end, I realized that the recursion limit in Python is set to 1000, so all that was required was to set the recursion limit to the max size of an input file. This can be done using sys.setrecursionlimit(20000). This is required for the large reverse data files, such as rev_20k.

Of these variations the 50 and 100 use insertion-sort to take care of the partitions under the stopping case, also known as the limit; while the median of three finds the median value of the first, middle and last element to use as a pivot. I have also written a Natural Merge Sorting algorithm to find previously occurred sorted sub-lists of an array which then incorporates into a merge sort. To compare the results with other sorting algos, I added a heapsort algorithm as well – but the results are not included in the final output below.

There is a total of five algorithms: 4 different variations of files – ascending, duplicates, random, reverse, with 7 different sizes 50, 500, 1000, 2000, 5000, 10000, and 20000. This results in a total of – 5 * (4*7) = 140 runs.

Results

During the program, we store a dictionary of the results for the given algo and file pair. Then use pandas to concatenate the list into a dataframe, which is saved in the excel file file_results.xlsx. The algos are compared in nanoseconds, the difference between the start and end time.time_ns(). Creating a pivot table, and sorting based on the lowest value for each file – we can observe which algos perform the best.

Obviously, the smaller the file size the quicker the sorting algorithm takes. But overall, quicksort median of three has more first place wins while quicksort_first takes the second (re: the second most first place wins). In some cases, especially with the reverse files – quicksort_median outperforms quicksort_100 by an order of magnitude of ~1000.

For most of the files, we can see that merge sort comes in a very close second to quicksort median. I am not surprised to see quicksort median being the dominant algorithm as this algorithm is a hybrid between quicksort and insertion sort. It's also interesting to see that merge sort is very close behind.

Analysis of Results

Median of three

- asc_500.txt, asc_1000.txt, asc_2000.txt, asc_5000.txt, asc_10000.txt, asc_20000.txt
- rev_500.txt , rev_1000.txt, rev_2000.txt, rev_5000.txt, rev_10000.txt, rev_20000.txt,
- ran_500.txt,

quicksort_first

- dup_50.txt, dup_500.txt, dup_1000.txt, dup_2000.txt, dup_5000.txt, dup_10000.txt, dup_20000.txt
- ran_1000.txt, ran_10000.txt, ran_2000.txt, ran_20000.txt, ran_5000.txt

quicksort_100

- rev_50.txt

quicksort 50

- ran_50.txt

Data below is stored in file_results.xlsx / file_results.csv.

Sum of total_time_ns file_name	algo_name	Total
asc_1000.txt	quicksort_median_of_three	1951000
	mergesort	2724000
	quicksort_first	24647000
	quicksort_50	73563000
	quicksort_100	157503000
asc_10000.txt	quicksort_median_of_three	26545000
	mergesort	32306000
	quicksort_first	2332765000
	quicksort_50	7511643000
	quicksort_100	9.90816E+11
asc_2000.txt	quicksort_median_of_three	4121000
	mergesort	6211000
	quicksort_first	95681000
	quicksort_100	296160000
	quicksort_50	301427000
asc_20000.txt	quicksort_median_of_three	54849000
	mergesort	68892000
	quicksort_first	9288034000
	quicksort_50	30108336000
	quicksort_100	3.24221E+11
asc_50.txt	quicksort_50	117000
	mergesort	156000
	quicksort_median_of_three	157000
	quicksort_first	172000
	quicksort_100	271000
asc_500.txt	quicksort_median_of_three	946000
	mergesort	1241000
	quicksort_first	6097000
	quicksort_50	17724000
	quicksort_100	33275000
asc_5000.txt	quicksort_median_of_three	11646000
	mergesort	15056000
	quicksort_first	578119000
	quicksort_50	1867880000
	quicksort_100	3211049000
dup_1000.txt	quicksort_first	1741000
	quicksort_median_of_three	1942000

	mergesort	2625000
	quicksort_50	5634000
	quicksort_100	67470000
dup_10000.txt	quicksort_first	20420000
	quicksort_median_of_three	25255000
	mergesort	32337000
	quicksort_50	234144000
	quicksort_100	10848947000
dup_2000.txt	quicksort_first	3780000
	quicksort_median_of_three	4162000
	mergesort	5592000
	quicksort_50	17087000
	quicksort_100	149701000
dup_20000.txt	quicksort_first	41940000
	quicksort_median_of_three	51672000
	mergesort	68740000
	quicksort_50	1643391000
	quicksort_100	45359527000
dup_50.txt	quicksort_first	145000
	quicksort_50	167000
	mergesort	180000
	quicksort_100	188000
	quicksort_median_of_three	188000
dup_500.txt	quicksort_first	774000
	quicksort_median_of_three	923000
	quicksort_50	1129000
	mergesort	1216000
	quicksort_100	16453000
dup_5000.txt	quicksort_first	8810000
	quicksort_median_of_three	10888000
	mergesort	15227000
	quicksort_50	161629000
	quicksort_100	3315454000
ran_1000.txt	quicksort_first	1624000
	quicksort_median_of_three	1919000
	mergesort	2595000
	quicksort_50	6086000
	quicksort_100	238677000
ran_10000.txt	quicksort_first	19365000
	quicksort_median_of_three	25520000
	mergesort	32137000
	quicksort_50	487113000
	quicksort_100	12915708000
ran_2000.txt	quicksort_first	3296000

	quicksort_median_of_three	4244000
	mergesort	5673000
	quicksort_50	23199000
	quicksort_100	295688000
ran_20000.txt	quicksort_first	40898000
	quicksort_median_of_three	54099000
	mergesort	68857000
	quicksort_50	1182296000
	quicksort_100	9.06978E+11
ran_50.txt	quicksort_50	124000
	mergesort	159000
	quicksort_median_of_three	160000
	quicksort_first	288000
	quicksort_100	360000
ran_500.txt	quicksort_median_of_three	928000
	quicksort_first	1159000
	mergesort	1238000
	quicksort_50	1472000
	quicksort_100	17441000
ran_5000.txt	quicksort_first	9417000
	quicksort_median_of_three	11845000
	mergesort	15154000
	quicksort_50	92108000
	quicksort_100	3321606000
rev_1000.txt	quicksort_median_of_three	1948000
	mergesort	2630000
	quicksort_50	4889000
	quicksort_first	30480000
	quicksort_100	72877000
rev_10000.txt	quicksort_median_of_three	24870000
	mergesort	32359000
	quicksort_50	319481000
	quicksort_first	3029697000
	quicksort_100	7529062000
rev_2000.txt	quicksort_median_of_three	4343000
	mergesort	5538000
	quicksort_50	17053000
	quicksort_first	122947000
	quicksort_100	531606000
rev_20000.txt	quicksort_median_of_three	53632000
	mergesort	68167000
	quicksort_50	1241831000
	quicksort_first	12579897000
	quicksort_100	54171235000

rev_50.txt	quicksort_100	123000
	mergesort	163000
	quicksort_50	178000
	quicksort_first	188000
	quicksort_median_of_three	4911000
rev_500.txt	quicksort_median_of_three	974000
	mergesort	1213000
	quicksort_50	1390000
	quicksort_first	7352000
	quicksort_100	17092000
rev_5000.txt	quicksort_median_of_three	11879000
	mergesort	15128000
	quicksort_50	83875000
	quicksort_first	772100000
	quicksort_100	3771044000
Grand Total		**2.44368E+12**