Exercise: Sorting

2/9/2024

1/1 Points

Attempt 2 ⌄

○ Review Feedback
2/8/2024

Attempt 2 Score:
1/1

View Feedback

Anonymous Grading: **no**

Unlimited Attempts Allowed
1/19/2024 to 2/9/2024

⌄ Details

The following exercise will compare the performance of sorting algorithms.

- Implement standard Quicksort (where the pivot is the element with index 0), as well as randomized Quicksort (where the pivot is a random element). Implement both Quicksort algorithms using auxiliary arrays to perform the partition.
- Implement Mergesort.
- Compare the performance of all three sorting algorithms. Which performs best and under what inputs? Provide an explanation. Compare the performance of the algorithms using the following three different types of data sets as follows:
  Let n be the input size. Consider $n = 32,\ 64,\ 128,\ 256,\ 512$ (i.e., 5 different input sizes). For each input size, generate an array **arr** of the following three types:
  - $\mathbf{arr} = [n,\ n-1,\ \ldots, 3,\ 2,\ 1]$ (i.e., the numbers from 1 to n in reverse sorted order)
  - **arr** is a random *permutation* of $[1,\ 2,\ \ldots,\ n]$
  - $\mathbf{arr} = [1,\ 3,\ \ldots,\ n-1, 2,\ 4,\ \ldots,\ n]$ (i.e., the odd numbers from 1 to n in increasing order followed by the even numbers from 1 to n in increasing order).

  Present the results (i.e., the execution time) of these algorithms in a table.

Below are example tables made in Google Sheets.

▶ Example Table 1

▶ Example Table 2

⌄ View Rubric

Sorting

Criteria	Ratings		Pts
Correct Quicksort Implementation	**0.25 pts** **Full Marks**	**0 pts** **No Marks**	/ 0.25 pts
Correct Mergesort Implementation	**0.25 pts** **Full Marks**	**0 pts** **No Marks**	/ 0.25 pts
Correct Data Generation	**0.25 pts** **Full Marks**	**0 pts** **No Marks**	/ 0.25 pts
Reasonable Comparison and Explanation	**0.25 pts** **Full Marks**	**0 pts** **No Marks**	/ 0.25 pts
			Total Points: 0

STANDARD QUICKSORT

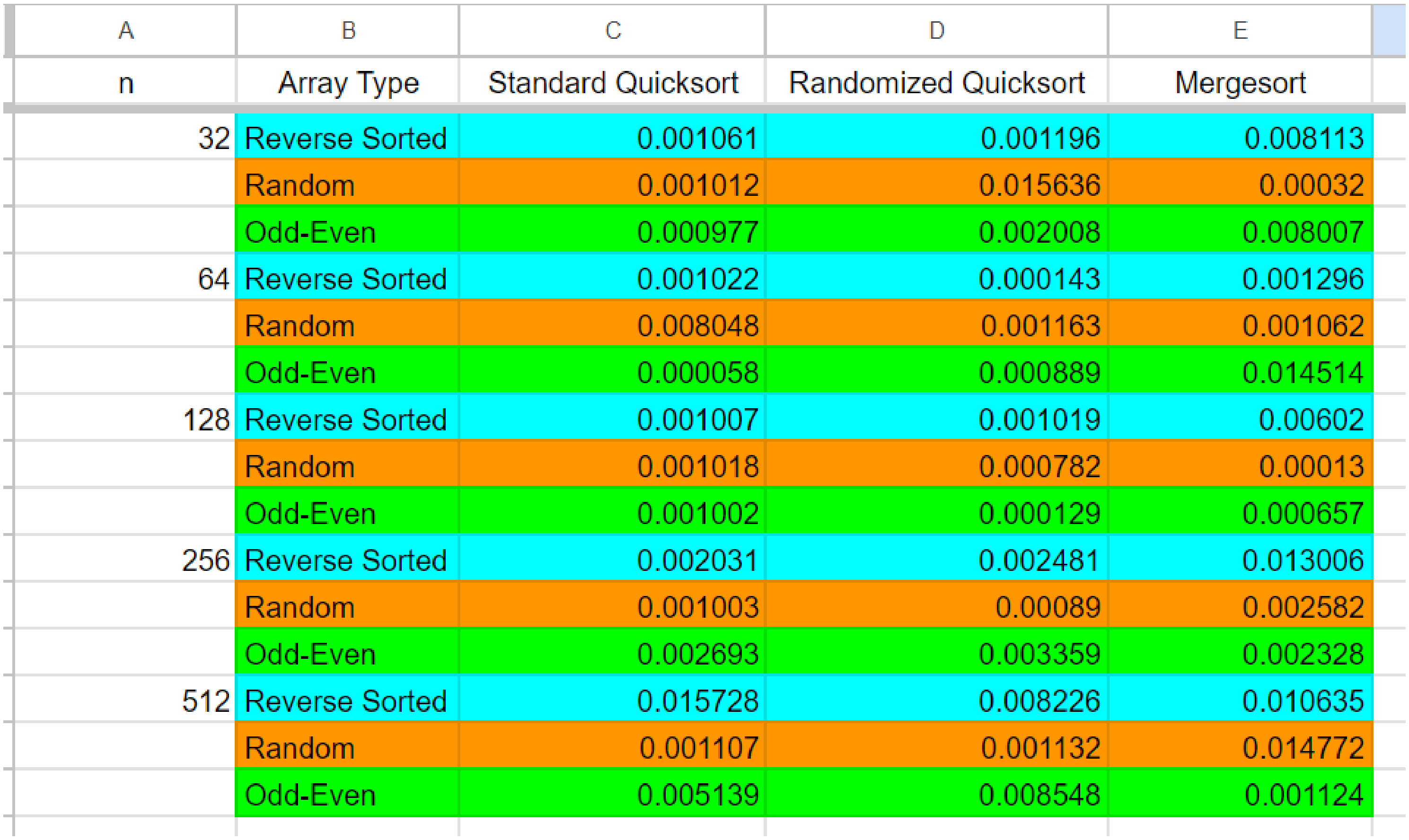
quicksort_std-2.py (https://canvas.uh.edu/users/41721/files/2819257?wrap=1&verifier=LTuD9bAThFZwGQ5aoPoJdl2TG9BAzmhXna0Uj4Md) ⤓ (https://canvas.uh.edu/users/41721/files/2819257/download?verifier=LTuD9bAThFZwGQ5aoPoJdl2TG9BAzmhXna0Uj4Md&download_frd=1)

RANDOM QUICKSORT

quicksort_rand-1.py (https://canvas.uh.edu/users/41721/files/2819262?wrap=1&verifier=71NRG6Vxm3fyWexMdrQud1WrgEnEH41LVtxjzWqw) ⤓ (https://canvas.uh.edu/users/41721/files/2819262/download?verifier=71NRG6Vxm3fyWexMdrQud1WrgEnEH41LVtxjzWqw&download_frd=1)

MERGE SORT

merge_sort-3.py (https://canvas.uh.edu/users/41721/files/2819271?wrap=1&verifier=wIGcmxnYeXYbuV887ERV5VJmDi3DbsDnz0w63gsZ) ⤓ (https://canvas.uh.edu/users/41721/files/2819271/download?verifier=wIGcmxnYeXYbuV887ERV5VJmDi3DbsDnz0w63gsZ&download_frd=1)



A	B	C	D	E
n	Array Type	Standard Quicksort	Randomized Quicksort	Mergesort
32	Reverse Sorted	0.001061	0.001196	0.008113
	Random	0.001012	0.015636	0.00032
	Odd-Even	0.000977	0.002008	0.008007
64	Reverse Sorted	0.001022	0.000143	0.001296
	Random	0.008048	0.001163	0.001062
	Odd-Even	0.000058	0.000889	0.014514
128	Reverse Sorted	0.001007	0.001019	0.00602
	Random	0.001018	0.000782	0.00013
	Odd-Even	0.001002	0.000129	0.000657
256	Reverse Sorted	0.002031	0.002481	0.013006
	Random	0.001003	0.00089	0.002582
	Odd-Even	0.002693	0.003359	0.002328
512	Reverse Sorted	0.015728	0.008226	0.010635
	Random	0.001107	0.001132	0.014772
	Odd-Even	0.005139	0.008548	0.001124

For arrays of input size 32:

- Standard Quicksort performs the best for reverse sorted array.
- Merge Sort performs the best for random array.
- Standard quicksort performs the best for odd-even array.

For arrays of input size 64:

- Randomized quicksort performs the best for reverse sorted array.
- Merge sort performs the best for random array.
- Standard quicksort performs the best for odd-even array.

For arrays of input size 128:

- Standard quicksort performs the best for reverse sorted array.
- Merge sort performs the best for random array.
- Randomized quicksort performs the best for odd-even sorted array.

For arrays of input size 256:

- Standard quicksort performs the best for reverse sorted array.
- Randomized quicksort performs the best for random array.
- Merge sort performs the best for odd-even array.

For arrays of input size 512:

- Randomized quicksort performs the best for reverse sorted array.
- Standard quicksort performs the best for random array.
- Merge sort performs the best for odd-even array.

For smaller arrays with input sizes (n = 32, n = 64), Standard quicksort works best for almost all array types.

For larger arrays with input sizes (n = 128, n = 256, n = 512), Merge sort works best for odd-even array type.

Overall, **Standard quicksort** performs the best for all array types, as it can utilize the partially sorted array, especially for the smaller input sizes, but slower for larger arrays due to worst-case scenarios. **Merge sort** performs the best with odd-even inputs, especially with larger inputs because of its stable divide-and-conquer approach that ensures good performance. **Standard quicksort** performs the best for reversed sorted inputs for almost all input sizes. **Randomized quicksort** performs the worst for random array as it could lead to more comparisons and swaps than expected depending on the pivot selection.

<
(https://canvas.uh.edu/courses/10626/modules/items/849873)

>
(https://canvas.uh.edu/courses/10626/modules/i