MSML606 HW4: Quicksort, Heapsort

Due: 7/20/2025 by 11:59 pm

In this assignment, you will:

1. Implement two sorting algorithms to sort an array of floating-point numbers in increasing order

- Randomized Quicksort
- Heapsort

Note: The students should implement the heap themselves without using any packages.

2. Test your implementations on various types of input arrays (provided test cases)
    1. Random order
    2. Repeated Values
    3. Ordered/reverse ordered
3. Analyze the time complexity of each algorithm for different input cases.

- Vary the input size N (e.g., N=10^4, 10^5, 10^6, ...). Adjust the range until it takes more than several seconds or a couple of minutes, or until any issues arise. You may use Python's random package to shuffle the input of n numbers.
- Measure the exact running time using timers (e.g., time.perf_counter())
- Measure memory usage using tools like 'tracemalloc'
- Handle exceptions encountered during the computation (e.g., 'RecursionError') and document when they occur. It's totally fine if no issues are observed.

4. Report your observations and results.

Deliverables (Refer Rubrics for marks distribution)

1. **Source code** (boilerplate template along with the test cases have been provided to you)
2. **Report**

- Implementation details: briefly describe each algorithm
- Time complexity analysis: best, worst, average case
- Compare the performance of Randomized Quicksort and Heapsort based on their theoretical complexities and actual running times. For problem 3, include a plot or table comparing running times and memory usage.
- Discuss any patterns or trends you observed in the running times

You may load the above deliverables as separate files or in a zipped folder.

Criteria	Ratings	Pts
This criterion is linked to a Learning OutcomeRandomized Quicksort Implementation	**10 ptsFull Marks** **0 ptsNo Marks**	10 pts
This criterion is linked to a Learning OutcomeHeapsort Implementation	**10 ptsFull Marks** **0 ptsNo Marks**	10 pts
This criterion is linked to a Learning OutcomePerformance Analysis (Time & Space Complexity) o Use timers (e.g., time.perf_counter()) to measure exact running times over varying input sizes (e.g., N = 10^4, 10^5, 10^6, ...). o Use tools like tracemalloc to measure memory usage. o Correctly handles exceptions (e.g., RecursionError) and documents when/if they occur.	**5 ptsFull Marks** **0 ptsNo Marks**	5 pts

Criteria	Ratings	Pts
This criterion is linked to a Learning OutcomeReport 1. Implementation Details (4 Marks): o Briefly describe each algorithm 2. Time Complexity Analysis (4 Marks): o Provide best, worst, and average-case complexity for each algorithm. o Explain how the measured performance corresponds to these theoretical complexities. 3. Performance Comparison (4 Marks): o Compare the performance (running time and memory usage) of Randomized Quicksort vs. Heapsort. o Include a table or plot summarizing the experimental results. 4. Discussion of Observations (3 Marks): o Discuss any trends, patterns, or issues observed during testing. o Clearly document insights on how input types (random, repeated, ordered, reverse-ordered) affect performance.	**15 ptsFull Marks** **0 ptsNo Marks**	15 pts

Total Points: 40