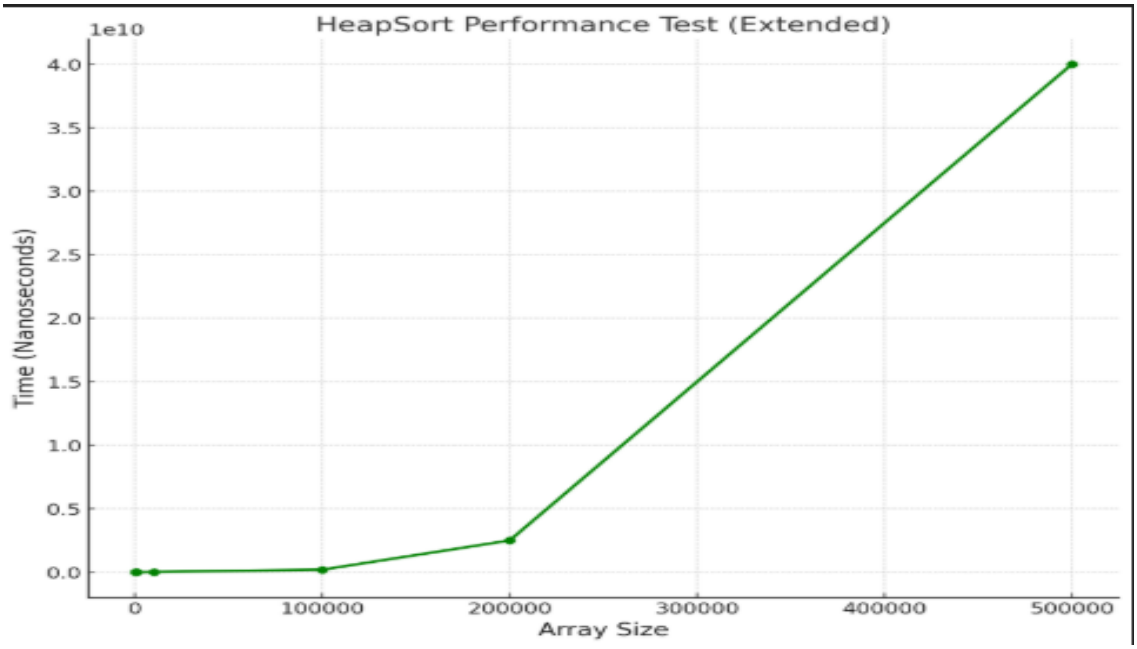
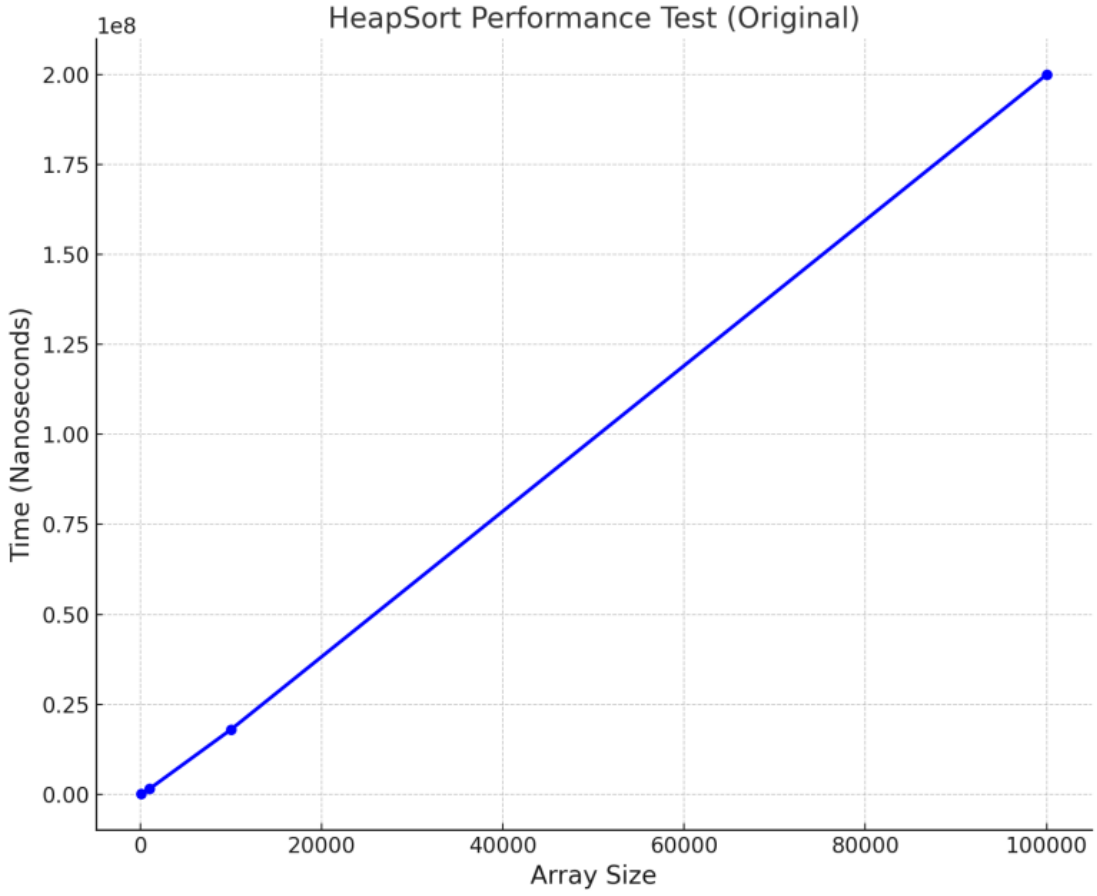
Kaber Daryn SE-2430

During this assignment, I implemented the HeapSort algorithm, which is an efficient in-place sorting algorithm based on the binary heap data structure. The first step was to write the core logic for building a max-heap and then repeatedly extracting the maximum element from the heap, placing it at the end of the array, and restoring the heap property using the heapify function. I also implemented a test suite using JUnit 5 to validate the correctness of the algorithm, covering edge cases such as empty arrays, single-element arrays, reverse-sorted arrays, and arrays with duplicate elements. After implementing the algorithm and testing it for correctness, I focused on analyzing the performance of the HeapSort algorithm by measuring the time taken for arrays of varying sizes (100, 1000, 10000, and 100000 elements), which confirmed the theoretical time complexity of O(n log n). Additionally, I wrote a report to describe the algorithm, its time and space complexities, the test cases, and the empirical results. In terms of optimization, the HeapSort algorithm was already quite efficient, and I did not find significant areas for improvement in terms of time complexity. However, I identified opportunities to optimize constant factors, such as by tweaking the implementation or utilizing a different heap structure. The work was completed by ensuring the project was properly uploaded to GitHub with all necessary files, including the code, tests, performance data, and the final report


HeapSort Performance Test (Extended)
1e10
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Time (Nanoseconds)
0
100000
200000
300000
400000
500000
Array Size


HeapSort Performance Test (Original)
1e8
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Time (Nanoseconds)
0
20000
40000
60000
80000
100000
Array Size