SORTING ALGORITHMS
CME 2204

2023510057
SERAY KESKİNKILINÇ
05.05.2025

Java Code Description

This java code implements and compares the performances of some sorting algorithms.Insertion sort,Merge sort,Heap sort and Quick sort. The main class Sorting.java is located in the src folder and contains the core methods to perform sorting, generate input arrays, read/write files, and record execution time.
Each algorithm is executed on three types of inputs:

Random integers (read from text files)

Increasing order integers (generated in code)

Decreasing order integers (generated in code)

Execution times are measured using System.nanoTime() and results are saved into text files named according to algorithm and input type.

Hardware Specifications

My;

Processor: Apple M2

RAM: 16 GB

OS: macOS

Java Version: Java SE 1.8

IDE: Eclipse 2023-12

Performance Comparison Matrix

	RANDOM INTEGERS (Read numbers from input files)			INCREASING INTEGERS (Create arrays in your code)			DECREASING INTEGERS (Create arrays in your code)		
N	1,000	10,000	100,000	1,000	10,000	100,000	1,000	10,000	100,000
Insertion Sort	1 ms	19 ms	587 ms	0 ms	0 ms	0 ms	3 ms	11 ms	1142 ms
Merge Sort	0 ms	0 ms	8 ms	0 ms	0 ms	3 ms	0 ms	0 ms	3 ms
Heap Sort	0 ms	1 ms	8 ms	0 ms	0 ms	5 ms	0 ms	0 ms	5 ms
Quick Sort	0 ms	1 ms	5 ms	1 ms	49 ms	4893 ms	2 ms	34 ms	3456 ms

About The Results

Insertion Sort worked well on small arrays and especially on already sorted (increasing) data, taking 0 ms even for 100K inputs. However, it performed very poorly on decreasing inputs, taking over a second (1142 ms) for 100K items.

Merge Sort showed stable and consistent performance in all cases. It completed sorting in a few milliseconds regardless of input type or size,complexity and divide-and-conquer strategy.

Heap Sort also performed consistently well but was slightly slower than Merge Sort on average.

Quick Sort was fast on random inputs but performed terribly on already sorted and reverse-sorted data. It took nearly 4 seconds (3899 ms and 3549 ms) for 100K increasing and decreasing inputs, respectively, due to bad pivot choices that led to unbalanced recursion and even a StackOverflowError in earlier tests. This confirmed how important pivot selection is for Quick Sort's efficiency.

Overall, Merge Sort was the most reliable algorithm in my experiments, while Quick Sort had the fastest times in some cases but also the worst in others.

Algorithm Properties Table

	Average Case Complexity	Best Case Complexity	Worst Case Complexity	Is Stable?	Is in Place?	Comparison Based or Divide and Conquer
Insertion Sort	O(n^2)	O(n)	O(n^2)	Yes	Yes	Comparison b.
Merge Sort	O(n logn)	O(n logn)	O(n logn)	Yes	No	Divide and Con.
Heap Sort	O(n logn)	O(n logn)	O(n logn)	No	Yes	Comparison b.
Quick Sort	O(n logn)	O(n logn)	O(n^2)	No	Yes	Divide and Con.