

**2024-2025 SPRING
CME 2204 ALGORITHM ANALYSIS**

**Assignment-1
Comparison of Insertion Sort, Merge Sort, Heap Sort and Quick Sort**

**Due
07.05.2025 23:59**

In this assignment, you will implement and test four sorting algorithms: Insertion sort, Merge sort, Heap sort, and Quick sort. You will implement your own algorithms and compare performances on random, increasing-order order, and decreasing-order integers with three different sizes, including 1000, 10000, and 100000.

1. Algorithms

1.1. Insertion Sort

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration. Insertion sort works similarly as we sort cards in our hand in a card game. We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.

1.2. Merge Sort

Merge sort sorts a list by dividing it into smaller sublists, sorting those sublists, and then merging them back together. It works by recursively splitting the list into halves until each sublist has only one element, which is inherently sorted. Then, it merges the sorted sublists back together to produce the final sorted list.

1.3. Heap Sort

Heap sort uses a binary heap data structure. The binary heap is a complete binary tree that satisfies the heap property: in a max heap, each parent node is greater than or equal to its children.

Heap sort divides the input into a sorted and an unsorted region and iteratively shrinks the unsorted region by extracting the largest element and moving it to the sorted region.

1.4. Quick Sort

Quick sort is a highly efficient divide-and-conquer sorting algorithm. It works by selecting a 'pivot' element from the list and partitioning the other elements into two sub-arrays: those less than the pivot and those greater than the pivot. The pivot element is then in its final sorted position. The algorithm then recursively sorts the sub-arrays.

2. Input Files

You are given three input files containing 1000, 10000, and 100000 random integer numbers.

- 1K_random_input.txt
- 10K_random_input.txt
- 100K_random_input.txt

These files contain a number in each line. Place them in your project directory for direct access. They will be replaced with new ones during automatic code control.

You must create arrays for the increasing-order and decreasing-order integers in your code.

3. Output Files

After the algorithms are executed, the results must be written to the output files in the same location as the input files (a total of 36 output files). Name the output files with the given name convention below:

(algorithm_name)_(n)_(data_type)_output.txt

Example = heapsort_1K_random_output.txt

algorithm_name = insertionsort/mergesort/heapsort/quicksort

n = 1K/10K/100K




data_type = random/increasing/decreasing

4. Submission

You must upload your Java source code in a zip folder and your report via **online.deu.edu.tr**.

4.1. Java source code and class files

You are required to create a main class named *Sorting.java*. You can create more classes if necessary. Place all your .java files into the *src* folder and .class files into the *bin* folder. Then, name and compress the main folder as given below and upload the zip file (exp. 2043901815_code.zip) via online.deu.edu.tr.

	(student_number)_code	// main folder
	src	// source file folder
	- <i>Sorting.java</i>	
	- ...	
	bin	//class file folder
	- <i>Sorting.class</i>	
	- ...	

4.2. Report

You must prepare a project report that

1. containing a cover page specifying your name and student number,
2. describing your Java code,
3. indicating the hardware specifications of the machine on which you are running the algorithms.
4. presenting the performance comparison matrix of the sorting algorithms (Table 1),
5. including a discussion about the results,
6. presenting the properties of the algorithms (Table 2).

You are expected to compare each sorting algorithm according to its running time (in milliseconds) for different inputs and fill out Table 1 accordingly. After you fill out the table, discuss the best and worst algorithms for various conditions.

* Note that the elapsed time of creating the arrays and writing output files should not be considered when calculating the total running time of sorting algorithms. You only must check how long the actual sorting algorithm is running.

Table 1. 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									
Merge Sort									
Heap Sort									
Quick Sort									

In addition to the performance comparison matrix, you must complete Table 2, which indicates the average, best, and worst-case time complexities and the properties of algorithms, including stability, to run in place and programming approach (comparison-based or divide-and-conquer).

Table 2. 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						
Merge Sort						
Heap Sort						
Quick Sort						

Name your report as (student_number)_report.pdf (exp. 2043901815_ report.pdf) and upload via online.deu.edu.tr with your source files.

5. Plagiarism Control

The assignment will be submitted individually. The submissions will be checked for code similarity. Copy assignments will be graded as zero!

6. Grading Policy

Job	Grade %
Insertion sort	15
Merge sort	15
Heap sort	15
Quick sort	15
Reading input files and generating output files	10
Report - Performance comparison - Algorithm properties, etc.	30