Hacettepe University

Computer Engineering Department

BBM103 Laboratory

Fall 2024

PA5: Algorithmic Sorting and Analysis

Deadline: 07/01/2025 at 23:59:59 - NO EXTENSIONS!

Data Context

In a small yet bustling city known as Algora, a group of dedicated data scientists, engineers, and mathematicians, collectively called the Order of Numeris, were tasked with solving one of the most pressing challenges of their time: optimizing the city's critical logistics network..

Algora had grown into a hub of commerce, with thousands of packages moving daily through its intricate web of warehouses and delivery routes. However, due to rapid growth and chaotic planning, inefficiencies crept into the system, causing delays, increased costs, and unhappy citizens.

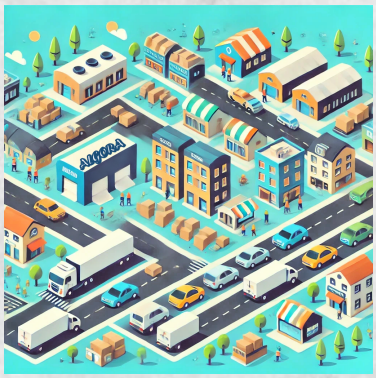


Figure 1: The city of Algora

Project Goal:

Through the Algorithmic Speed Analysis, the Order of Numeris would:

- Test various sorting algorithms on simulated datasets.
- Identify the most efficient sorting strategy for datasets of varying sizes.
- Address real-world issues and adapt algorithms to solve them effectively.

System Specifications

1 Objective

Analyze the efficiency of different algorithms by implementing, testing, and comparing their performances for solving a common problem.

1.1 Dataset Description

The dataset is a synthetic dataset, where each row contains:

- `Warehouse_ID`: A unique identifier for each warehouse.

- **Priority_Level:** An integer representing the priority level.
- **Package_Count:** An integer representing the number of packages.

The dataset generation code is provided and ensures a unique dataset for each student.

1.2 Problem

Write a program to sort the dataset in two levels:

1. **Primary Criteria:** Priority_Level (ascending order).
2. **Secondary Criteria:** Package_Count (ascending order).

Example Input/Output:

Input:
WH-003, 5, 500
WH-001, 3, 400
WH-002, 5, 300

Output:
WH-001, 3, 400
WH-002, 5, 300
WH-003, 5, 500

2 Step-by-Step Explanation of Sorting Methods

You will implement the sorting algorithms described below

Method 1: Bubble Sort

Steps:

1. Start at the beginning of the dataset.
2. Compare each pair of adjacent elements:
   - If the first is greater than the second, swap them.
   - Otherwise, leave them in place.
3. Move to the next pair and repeat until reaching the end of the dataset.
4. After one pass, the largest element will be in its correct position.

5. Repeat the process for the remaining unsorted portion of the dataset until all elements are sorted.

Example:

- Dataset: $[5, 2, 9, 1, 5]$

Pass 1:

- Compare 5 and 2: Swap $\rightarrow [2, 5, 9, 1, 5]$
- Compare 5 and 9: No swap $\rightarrow [2, 5, 9, 1, 5]$
- Compare 9 and 1: Swap $\rightarrow [2, 5, 1, 9, 5]$
- Compare 9 and 5: Swap $\rightarrow [2, 5, 1, 5, 9]$

Pass 2:

- Compare 2 and 5: No swap $\rightarrow [2, 5, 1, 5, 9]$
- Compare 1 and 5: Swap $\rightarrow [2, 1, 5, 5, 9]$
- Compare 5 and 5: No swap $\rightarrow [2, 1, 5, 5, 9]$

Pass 3:

- Compare 2 and 1: Swap $\rightarrow [1, 2, 5, 5, 9]$
- Compare 2 and 5: No Swap $\rightarrow [1, 2, 5, 5, 9]$

Final Sorted Dataset: $[1, 2, 5, 5, 9]$

Method 2: Merge Sort

Steps:

1. Divide the dataset into two roughly equal halves.
2. Recursively apply this method to each half.
3. Once the subsets are reduced to single elements, merge them:
    - Compare the smallest unmerged elements from each subset.
    - Add the smaller element to the result.
    - Repeat until all elements are merged into a single sorted dataset.

Example:

- Dataset: $[5, 2, 9, 1, 5]$

Step 1: Divide

- $[5, 2, 9]$ and $[1, 5]$

Step 2: Recursively Sort

- $[5, 2, 9] \rightarrow [5, 2]$ and $[9] \rightarrow [5]$ and $[2] \rightarrow [2, 5] \rightarrow [2, 5, 9]$
- $[1, 5] \rightarrow [1]$ and $[5] \rightarrow [1, 5]$

Step 3: Merge

- Merge $[2, 5, 9]$ and $[1, 5]$:
  - Compare 2 and 1: Add $1 \rightarrow [1]$
  - Compare 2 and 5: Add $2 \rightarrow [1, 2]$
  - Compare 5 and 5: Add $5 \rightarrow [1, 2, 5]$
  - Add $5 \rightarrow [1, 2, 5, 5]$
  - Add $9 \rightarrow [1, 2, 5, 5, 9]$

Final Sorted Dataset: $[1, 2, 5, 5, 9]$

Method 3: Quick Sort

Steps:

1. Select a "pivot" element from the dataset (commonly the middle or last element).
2. Partition the dataset into two groups:
   - Elements smaller than the pivot.
   - Elements larger than the pivot.
3. Recursively apply the process to each group.
4. Combine the sorted groups and the pivot into a single sorted dataset.

Example:

- Dataset: $[5, 2, 9, 1, 5]$

Step 1: Choose Pivot

- Pivot: 5

Step 2: Partition

- Smaller than 5: $[2, 1]$
- Larger than 5: $[9]$

- Pivots: $[5, 5]$

Step 3: Recursively Sort

- $[2, 1] \rightarrow$ Pivot: $2 \rightarrow$ Smaller: $[1] \rightarrow$ Larger: $\emptyset \rightarrow [1, 2]$
- $[9] \rightarrow$ Already sorted.

Step 4: Combine

- Combine $[1, 2]$, $[5, 5]$, and $[9] \rightarrow [1, 2, 5, 5, 9]$

Final Sorted Dataset: $[1, 2, 5, 5, 9]$

2.1 Summary of Algorithms

1. **First Method:**
   - Simple but slow due to repeated comparisons.
   - Time Complexity: $O(n^2)$
2. **Second Method:**
   - Efficient for large datasets with predictable performance.
   - Time Complexity: $O(n \log n)$
3. **Third Method:**
   - Efficient for average cases but sensitive to pivot choice.
   - Time Complexity: $O(n \log n)$ average, $O(n^2)$ worst-case.

3 Rules and Guidelines

1. **Code Requirements:**
   - Write clean, modular, and well-documented code.
   - Implement each sorting algorithm in its own dedicated function.
   - Include comments explaining the purpose of each function and critical sections of your code.
   - Follow the provided guidelines for where to modify or add code. **Do not change code outside the specified areas.**
   - **Note 1: In the quick sort implementation, select the middle element of the array as the pivot. If the array length is even, choose the element just above the midpoint.**

- **Note 2: In the merge sort implementation, divide the array into two equal parts. If the array length is odd, right array should be larger in size.**
- **WARNING:** Do **not** use any built-in functions for **sorting** or any **library**!

2. **Error Handling:**
   - Account for edge cases, such as:
     - An empty **input:**
     - Non-integer values in the **input:**
     - Duplicates in the **input:**
     - Single-element **input:**

3. **Output:**
   - A file named **hw05_output.txt** must include:
     - Results of Bubble Sort, including:
       - ⋆ Sorted data (ascending by **Priority Level**, then by **Package Count**).
       - ⋆ Iteration counts for both Priority Level and Package Count sorts.
     - Results comparison:
       - ⋆ Whether Bubble Sort and Merge Sort results are identical.
       - ⋆ Whether Bubble Sort and Quick Sort results are identical.
     - Counters for each sorting algorithm:
       - ⋆ **Bubble Sort:**
         - · Number of iterations for sorting by **Priority Level.**
         - · Number of iterations for sorting by **Package Count.**
       - ⋆ **Merge Sort:**
         - · During the merging process, count how many times an element in the right array is smaller than an element in the left array for sorting based on **Priority Level.**
         - · During the merging process, count how many times an element in the right array is smaller than an element in the left array for sorting based on **Package Count.**
         - · For more details, refer to the example provided in the **Piazza discussion.**
       - ⋆ **Quick Sort:**
         - · Number of recursive steps for sorting by **Priority Level.**
         - · Number of recursive steps for sorting by **Package Count.**
     - **Notes:**

* The sample input and output provided as part of the assignment materials are for a dataset of size 10.
* Update the '*SIZE*' variable in the starter code to 100 to generate a dataset of size 100.
* The output format is pre-configured in the starter code, so **no** additional formatting is required.

4 Submission Format

Submit a ZIP file containing the following:

- **b<studentID>.zip**
  - **hw05_input.csv**
  - **hw05_output.txt**
  - **hw05.py**

Note: Only the submissions with the shared format will be accepted!

- A Python script named **hw05.py** containing:
  - Implementations of all three sorting algorithms (**bubble_sort**, **merge_sort**, and **quick_sort**).
  - Implementation of **two_level_sorting** function for dual criteria sorting.
  - Code to generate datasets to perform sorting.
  - Outputs written to a file named **hw05_output.txt**.
- The **hw05_output.txt** file must contain the sorted results for the provided dataset. This file will be validated for correctness.
- Ensure the program adheres to the autograder's input and output format specifications.

Note: Failure to follow the format or filename conventions will result in a failed submission.

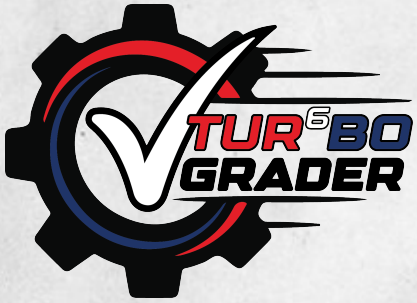
5 Evaluation

Submissions will be automatically graded. Ensure your submission meets the following criteria:

1. **Correctness of Sorting (40 points):**
   - Correct bubble sorting results (20 points as 5 points for priority level and 15 for package count).
   - Correct merge sorting results (10 points).
   - Correct quick sorting results (10 points).
   - Note that if bubble sorting output fails, the other 2 outputs will automatically fail too!
2. **Design Check (60 points):**

- Correct bubble sorting implementation (20 points).
- Correct merge sorting implementation (20 points).
- Correct quick sorting results (20 points).
- **Note 1:** If the outputs fail, the design will automatically fail too!
- **Note 2:** Ensure efficient implementation. Excessively long execution times may result in a penalty!

6 Automatic Grading [VERY IMPORTANT!]



Submissions will be graded using an automatic grading script. **There will not be any manual grading.** Therefore, you are expected to **match the output file 100%** to receive full credit. **You MUST download and use the starter code files (as shared on piazza) before starting your work! Do NOT alter the structure of the code beyond the specified areas as previously mentioned!**

You must test your code using the Tur^{6}Bo Grader before submission to verify which tests you are passing.

Note: Tur^{6}Bo Grader is for testing purposes only. You MUST still submit your full code via https://submit.cs.hacettepe.edu.tr/ to be graded.

7 Plagiarism Control Notice

Students must implement their solutions individually. All submissions will be submitted to a plagiarism check. Any submissions that show a high level of similarity will be reported as plagiarism attempts to the ethics committee.

8 Submission Instructions

Submissions will be accepted via **https://submit.cs.hacettepe.edu.tr/.**

The deadline for submissions is Wednesday, 07/01/2025 at 23:59:59, and no extensions will be applied! Late submissions will not be accepted!