Department of Computer Science, UNM
CS361 Programming Project: Sorting

Spring 2025

Due: May 11 (Sunday) End of Day

Project Description

Sorting is at the core of a lot of algorithms. The goal of this project is to implement four common sorting algorithms and compare their performances.

1 Implementation

You are allowed to work in groups of up to **4** students.

1.1 Problem statement:

The algorithms:

- Sorting using a quad heap: You will be implementing a modified heap called a quad-heap, which is a degree 4 tree that is filled on all levels except the very last, that is filled from left to right.
- 3-way merge sort: You will be implementing a 3-way merge sort, which divides the given input into 3 subsets of roughly equal size and recursively sort each subset.
- Randomized quick sort: You will implement the randomized quick sort algorithm. In each iteration, the pivot must be randomly chosen using a random number generator.
- Tim sort: Tim sort is a hybrid sorting algorithm derived from merge sort and insertion sort, designed to perform well on real-world data. It is the default sorting algorithm in Python, Java, and Android.

1.2 Requirements and Programming Instructions

- For the basic implementation, you can assume that your inputs are numbers.

- You may choose to use C, C++, Java, or Python. However, all algorithms must be implemented using the same programming language.
- You may **NOT** use any data structures from existing libraries except:
    1. Basic data structures such as arrays.
    2. A random number generator.
    3. System functions, such as print to screen, write to file, etc.
- Create a function that executes all algorithms for a give input and provides output.

2 Performance Benchmarking

1. For each of the algorithm, you will benchmark its running time with respect to input data of size: $2^{20}, 2^{21}, 2^{22}, 2^{23}, 2^{24}, 2^{25}, 2^{26}, 2^{27}, 2^{28}, 2^{29}, 2^{30}$. You should exclude the I/O time when benchmarking. In other words, if your program is reading in the input from a file, your timer should start after all the data is loaded into the main memory.
2. When benchmarking, be sure to also compare the running time of these four algorithms when inputs are integers vs. double precision floating point numbers.
3. Compare the running times between the four algorithms.
4. **Extra credit** Change the implementation of one of your algorithms that goes beyond simple numerical sorting. This algorithm should be capable of sorting numbers, texts, characters, and handling complex data structures (for example, objects or records) based on multiple keys. It should sort based on at least two criteria (e.g., sort a list of student records first by GPA, then by last name). It should ensure that your algorithm is stable (i.e., the relative order of equal elements is preserved) if that is applicable to the chosen criteria.

3 Project Report

You will need to submit a final report of your findings. The report must be typewritten in a scientific paper format with a minimum 12pt font size. The report should include the following key sections:

1. **Title, authors, and institutions of the authors**.
2. **Introduction**.

3. **Preliminaries and Methods**: You will introduce the key ideas and performance characteristics of the implemented algorithms, and describe the algorithms exactly as implemented. As an example, in the preliminaries, you should discuss how the parent-child relation is calculated in a quad-heap.

4. **Performance analysis and discussions**: Here are some of the things you should discuss in your report:

   (a) Are the asymptotic running times of these algorithms close to their actual performance? What is the constant factor you have found in your running time analysis?

   (b) Do all 4 programs have the same coding complexity? What about debugging effort?

   (c) Which sort is the fastest or slowest? Why?

   (d) For every algorithm, what is the highest input size that the algorithms can handle without running into memory issues in your machine?

5. **Conclusion**.

6. **Bibliography** : Reference your work.

7. **Contributions of Each Team Member**: Describe in detail the contribution of each member of your team. For example, member A implemented the quad-heap and performed benchmark, member B implemented quick sort and generated plots, member C implemented 3-way merge sort and wrote parts of report, and member D implemented Tim Sort and took the lead in compiling the final report. This is just a sample division of labor. You will decide in your group what will be the division of tasks among team members.

8. **Acknowledgement**.

9. If you are attempting the extra credit you have to discuss in detail the implementation changes in code and performance changes in your algorithms. It should have a separate section in the report. It is an open-ended question and marking will be at the discretion of the instructor. Some discussions you can include in the report (not limited to):

   - Analyze how the added complexity (text,multiple keys, object comparisons) affects the overall performance compared to the basic algorithm.
   - Consider how the sort handles edge cases, such as missing or null values in one or more sorting keys.
   - Provide examples of scenarios or applications where a multi-key sort would be beneficial over simpler sorting methods.
   - What optimization strategies can be applied to improve the performance of your sort when handling large datasets?

4 Submission

You will be uploading the following on Canvas by midnight on May 11 (Sunday):

- Your final report in a single PDF file. Only one report is needed for a team.
- A single zipped file containing your source code, test case(s) to run your code (can be .txt file), and a .txt file with instructions on how to run the code.
  It is your responsibility to make sure that your code runs. Grader will run test cases that you will be providing to test your code.

Note: There will be no extension for the final project. No partial credit for late submissions. Plan accordingly and start now!