Analysis of Algorithms

BLG 335E

Project 1

Res. Assist. Meral Kuyucu
korkmazmer@itu.edu.tr

Instructors:
Assoc. Prof. Dr. İlkay Öksüz
Asst. Prof. Dr. Mehmet Baysan
Asst. Prof. Dr. Tuğçe Bilen

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 03.11.2024

1. Project Brief

You have graduated and landed a data scientist at a cutting-edge tech company. Your team has been tasked with conducting important analyses on large datasets. The challenge? You'll be using different algorithms to get the job done—whether it's searching through data, sorting it, or gathering key statistics to draw conclusions. Each task requires a different approach, and not all algorithms are created equal when it comes to efficiency.

Recently, **ITU Corporate Relations and Alumni Communication Office** approached your company with a request. They are interested in analyzing the activity surrounding the **#ITUHepYanimda** hashtag used during ITU's graduation day. Specifically, they want to know which tweets received the most engagement in terms of favorite count and retweet count. This analysis will help them understand how the ITU community engaged with this event and how they can improve future alumni relations efforts.

You are provided with a dataset of tweets related to ITU's graduation, containing **tweet IDs**, **favorite counts**, and **retweet counts**. Your job is to implement various algorithms with different time and space complexities, running them on this dataset to extract insights and improve performance. You'll see firsthand how the number of steps (or operations) required by each algorithm can drastically affect performance. Additionally, the distribution and permutation of the dataset will significantly impact how the algorithms perform. The more efficient you are, the faster you'll get results—and the more coffee breaks you'll earn!

Reporting is just as important as developing solutions and discovering insights. Proper reporting ensures that your findings are communicated clearly to your fellow engineers, allowing them to continue or expand on your work. It also enables projects you've contributed to be taken further by others in the company. For this reason, you will be required to document and present your findings as part of this project.

Efficiency: The Core of Scalable Data Solutions

Given a large dataset, one of the most fundamental tasks is to sort and order the items. By finding extreme values and statistics, such as the maximum, minimum, or average values for the **favorite counts** and **retweet counts**, you can describe the dataset without needing to examine each element individually.

Almost every data processing task involves some form of searching or sorting, whether it's finding a specific piece of data, organizing information for faster access, or making sense of large datasets. Searching through unsorted data can be slow and inefficient. However, when the data is sorted, retrieval is much faster, and more advanced search methods, such as binary search, become possible. Many algorithms depend on sorted data to function optimally, as you'll discover during this project.

By mastering these fundamental algorithms, you'll be better equipped to handle real-world data efficiently, making you a more effective and versatile engineer, ready to meet the demands of large-scale data analysis.

Objectives

The objective of this project is to explore and analyze how different algorithms handle tasks such as searching, sorting, and computing statistics on the provided dataset. Each algorithm has a unique time complexity, and your job is to implement them, run them on different permutations of the dataset, and evaluate their performance. Specifically, you will:

- Implement algorithms with sublinear, linear, log-linear, and quadratic time complexities.
- Test these algorithms on datasets with various arrangements.
- Compare the performance of each algorithm and explain why some perform better than others depending on the dataset structure.

By the end of this project, you'll have gained a deeper understanding of how different algorithms address tasks with varying levels of efficiency—and how these techniques play a crucial role in real-world data analysis for organizations like ITU Corporate Relations, as they seek to understand engagement during the graduation ceremony and improve future events.

Data

For this assignment you are given a synthetically generated data. The data columns include **Tweet ID**, **Number of Favorites** and **Number of Retweets**. Kindly be informed that multiple sized data sets are shared for comparison. The report template provided has the necessary guidelines for you to test your code out.

2. Implementation Details

2.1. Sorting Strategies for Large Datasets

When sorting a dataset, one of the most intuitive methods is pairwise comparison, followed by reordering based on those comparisons. Many sorting algorithms use this approach, though they differ in how pairs are selected and compared. Your firm has asked you to implement two such pairwise comparison methods as described in Section 2.1.1, as well as a more efficient approach discussed in Section 2.1.2.

These sorting methods will be used to maintain leaderboards for both favorite count and retweet count, helping ITU's office identify the most engaged tweets at a glance. With the growing volume of data, organizing it efficiently becomes critical to quickly assess which tweets gained the most traction.

Additionally, ITU's office needs to perform range queries, such as retrieving tweets with retweet counts between 1,000 and 10,000. Sorting the data by retweet count will allow these range queries to be handled efficiently, as you can quickly access the relevant subset of data. Since the dataset is currently unsorted, performing such operations is more time-consuming without prior sorting.

2.1.1. Laborious Effort - "Rome wasn't built in a day."

The first sorting algorithm compares adjacent elements and swaps them if they are out of order. This process repeats through the dataset, with the largest (or smallest) element "bubbling" to the top after each pass. The second sorting algorithm builds a sorted section by assuming that the first element is already sorted. For each subsequent element, it is inserted into its correct position within the sorted portion, shifting elements as needed.

Task: Implementing Bubble Sort and Insertion Sort

Implement both bubble sort and insertion sort, ensuring that your implementations are capable of sorting in both ascending and descending order based on all three attributes.

2.1.2. Balanced Efficiency - "Divide and conquer."

As an engineer, you often solve problems by breaking them down into smaller, manageable sub-problems. In this case, you will take a divide-and-conquer approach to sorting the dataset. By splitting the dataset into smaller parts, sorting each individually, and then combining them to form the final sorted sequence, this approach distributes the workload efficiently and ensures scalability, even with large datasets.

Task: Sorting by Retweet Count

Implement merge sort to sort the dataset all three attributes. Your implementation must be capable of sorting in both ascending and descending order.

2.2. Targeted Searches and Key Metrics

2.2.1. Swift Precision - “Zero in on the target.”

You are tasked with finding a specific item in a dataset that’s already organized. A common approach would be to traverse the dataset sequentially, reducing the size of your search space by one with each comparison. However, since the dataset is sorted, a more efficient method is to repeatedly cut the dataset in half, exponentially reducing the search space. This approach enables you to locate your target swiftly, even in large datasets.

You are asked to retrieve a specific tweet that has exactly 250 favorite counts. This tweet will receive a special award in honor of ITU’s 250th anniversary.

Task: Binary Search

Implement a binary search algorithm which is capable of locating a tweet with n favorites/retweets.

2.2.2. Steady Progress - “Slow and steady wins the race.”

Your next task as a data analyst is to determine how many tweets have reached significant engagement levels. Specifically, you are required to count how many tweets exceed a certain threshold of favorites or retweets. The ITU Alumni Office plans to highlight these highly engaged tweets during an upcoming event, and they need an overall count to showcase the level of community interaction. The number of steps required for this task will be proportional to the size of the dataset.

Task: Count Above Threshold

Implement a function that counts how many tweets have a favorite or retweet count greater than a given threshold. Return the number of tweets that meet this condition.

3. Deliverables

3.1. Code [70%]

A zip file has been provided with the assignment description. In order to set up your development environment, please do the following:

1. Install Docker Desktop
2. Install Visual Studio Code
3. Download the zip file provided with this assignment. Unzip it and place it in your file system.
4. Launch visual studio.
5. Click on File » Open Folder.
6. Select the folder you've downloaded.
7. Visual Studio Code will ask to open the folder in container (Figure 3.1). Click "Reopen in Container". It might also ask whether or not to trust the author. Select "Trust".

You should then be able to see the folder hierarchy on the left side of the screen. This folder contains the skeleton of the assignment. The content can be summarized as:

- bin: holds the compiled binary files from your project. Your executable goes here.
- data: holds your input datasets.
- include: stores header files. These files declare the structures, functions, and constants that are implemented elsewhere.
- lib: stores external libraries that your project depends on (we will use the munit testing framework).
- src: stores the code implementation.

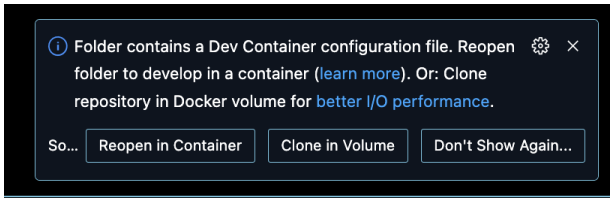

Figure 3.1: Environment Setup

```
root@20d70ce98e55:/workspaces/BLG335E-2024# make tests
bin/tests
Running test suite with seed 0x053a254d...
/TweetTests/bubble_sort_ascending     [ OK    ] [ 0.00100454 / 0.00100187 CPU ]
/TweetTests/bubble_sort_descending    [ OK    ] [ 0.00064754 / 0.00064713 CPU ]
/TweetTests/insertion_sort_ascending  [ OK    ] [ 0.00072392 / 0.00072358 CPU ]
/TweetTests/insertion_sort_descending[ OK    ] [ 0.00082004 / 0.00081746 CPU ]
/TweetTests/merge_sort_ascending      [ OK    ] [ 0.00065967 / 0.00065871 CPU ]
/TweetTests/merge_sort_descending     [ OK    ] [ 0.00057050 / 0.00056979 CPU ]
/TweetTests/binary_search             [ OK    ] [ 0.00052267 / 0.00052108 CPU ]
/TweetTests/count_above_threshold     [ OK    ] [ 0.00062608 / 0.00062658 CPU ]
8 of 8 (100%) tests successful, 0 (0%) test skipped.
```

Figure 3.2: Munit test results.

- tests: reserved for unit tests.
- Makefile: is used to automate the process of compiling and linking your C++ code into an executable.

The structure definition and the prototypes of the functions that are required are given in **“include/tweet.h”**. Modify the files in **“src/utilities.cpp”**, **“src/metrics.cpp”**, **“src/sort.cpp”** and **“src/main.cpp”**.

You are free to create helper functions for other miscellaneous tasks in the assignment, however, bear in mind that the functions required in “tweet.h” will be subject to unit tests, therefore, the inputs and outputs should follow the protocol described. Example tests have been provided under the **tests** directory. While developing, you can either build your code with the following terminal command:

```
g++ -g -o bin/tweet src/*.cpp -Iinclude
```

Then, you can run with:

```
./bin/tweet
```

You can also build and run with the makefile. To build, run the command:

```
make build
```

To run, use the command:

```
make run
```

To test, use the command:

```
make tests
```

If your code passes the tests provided, you should see an output similar to the one given in Figure 3.2. You are strongly advised to make sure your code passes those tests. Be informed that additional tests will be conducted when grading. **Zip and submit ONLY your “src” folder.**

3.2. Report [30%]

Upload the provided report template to Overleaf. **Fill in your report (not by hand) and submit ONLY the pdf file.** If you are not sure how to measure time elapsed, please refer to this link.

3.3. Guidelines and Grading

Your homework will be graded under the following criteria:

- Code: File Read: 5%, File Write: 5%, Bubble Sort: 15%, Insertion Sort: 15%, Merge Sort: 20%, Binary Search 5%, Count Above Threshold: 5%
- Report: 30 %

Please note that a **50% overall penalty** will be imposed on code that fails to compile and run within the provided **docker** environment. Additionally, the grade for the report will be **capped** at the number of points achieved in the code implementation part.

You will be graded on the following:

- Your efficient C++ **Implementation** of the assignment requirements. [70 pts]
  - Correctness
  - Clarity
  - Good Coding Practice
- **Report** detailing your implementation. [30 pts]
  - Content
  - Organization
  - Relevance
  - Quality

IMPORTANT NOTE:

- Please be informed that your code and report will be checked for plagiarism using plagiarism detection tools and disciplinary action will be taken upon detection of plagiarism. Please write your code and report on your own.
- Copying code fragments from any source including books, websites, or classmates is considered plagiarism.
- You are free to conduct research on the topics of the assignment to gain a better understanding of concepts at an algorithmic level. Make sure to cite any such sources that you refer to.

- Refrain from posting your code/report on any public platform (i.e. Github) before the deadline of the assignment.
- You are free to use STL for data structures ONLY (i.e. you can use vector, but not built-in sorting algorithms).