A close-up of a logo

Description automatically generated

Technical Report: University Ranking

Author: Anupama Rai

Student ID: 24128432

Subject Code: CMP4272

Date: June 19, 2023

Wordcount: 1967

Executive Summary

This technical documentation presents an analysis of university rankings based on key performance indicators such as entry standards, student satisfaction, research quality, research intensity, and graduate prospects. The study uses a comprehensive dataset to offer insights into the relative performance of various universities. To enhance the usability and accessibility of this dataset, several robust algorithms have been developed and implemented.

The system uses a modified selection sort algorithm to rank universities by student satisfaction, allowing for the easy identification of the top 40 institutions. Additionally, it implements a linear search algorithm to quickly locate specific universities within this top tier. For alphabetical sorting, the system employs Timsort with a custom key function that ignores common prefixes, ensuring a user-friendly order. Finally, a linear search with a counter is used to count universities with satisfaction scores above 85%. These solutions streamline data access and provide critical insights, making the system a valuable tool for journalists and researchers.

Table Of Contents

[Introduction…………………………………………………………………………………………. ……..1](#_Toc169718954)

[Theory 2](#_Toc169718955)

[I. Data Structures 3](#_Toc169718956)

[1. List 3](#_Toc169718957)

[2. Tuple 3](#_Toc169718958)

[3. List of Tuples 4](#_Toc169718959)

[4. Pandas DataFrame 4](#_Toc169718960)

[II. Algorithms 5](#_Toc169718961)

[1. Selection Sort 5](#_Toc169718962)

[2. Linear Search 5](#_Toc169718963)

[3. Custom Sort 6](#_Toc169718964)

[4. Counting with a Condition 6](#_Toc169718965)

[III. Empirical Basis Implementation 7](#_Toc169718966)

[1. Data Loading and Preparation 7](#_Toc169718967)

[2. Sorting Functions 8](#_Toc169718968)

[3. Search and Count Functions 9](#_Toc169718969)

[4. Main program 9](#_Toc169718970)

[IV. Comparisons 11](#_Toc169718971)

[1. Pandas dataframe and list of tuples 11](#_Toc169718972)

[2. List of tuples and dictionary of lists 11](#_Toc169718973)

[3. Selection sort and Quick sort 11](#_Toc169718974)

[4. Linear search and Binary search 12](#_Toc169718975)

[5. Custom Sort with key function and Prefix tree 12](#_Toc169718976)

[6. Counting with a condition and Binary indexed tree 12](#_Toc169718977)

[Operational Assessment 13](#_Toc169718978)

[Asymptotic analysis 14](#_Toc169718979)

[Assertion Table 15](#_Toc169718980)

[Conclusion 17](#_Toc169718981)

[References 18](#_Toc169718982)

[Appendix-I (Code) 19](#_Toc169718983)

[Appendix-II (Output) 22](#_Toc169718984)

List of Figures and Tables

[Figure 1: Chart of the current system 3](#_Toc169719938)

[Figure 2:List data structure implementation 4](#_Toc169719939)

[Figure 3: Tuple implementation 4](#_Toc169719940)

[Figure 4: List of Tuples implementation 5](#_Toc169719941)

[Figure 5: Pandas DataFrame implementation 5](#_Toc169719942)

[Figure 6: Selection sort algorithm code 6](#_Toc169719943)

[Figure 7: Linear search code 6](#_Toc169719944)

[Figure 8: Timsort implementation 7](#_Toc169719945)

[Figure 9: Counting with a condition code 7](#_Toc169719946)

[Figure 10: Data loading and preprocessing code 8](#_Toc169719947)

[Figure 11: Main program code 11](#_Toc169719948)

[Table 1: Asymptotical analysis of the data structures and algorithm……………………………………..15](#_Toc169719929)

[Table 2: Assertion table of the code 17](#_Toc169719930)

# Introduction

This document presents a comprehensive analysis of university rankings based on a diverse set of factors, including entry standards, student satisfaction, research quality, research intensity, and graduate prospects. The dataset used in this study comprises detailed information on these key performance indicators for various universities.

To enhance the usability and accessibility of this dataset, several key problems have been addressed and solved through the development of robust algorithms. It uses a modified selection sort algorithm to rank universities by student satisfaction, making it easy to highlight the top 40 institutions; implements a linear search to quickly locate specific universities within this top tier; employs Timsort with a custom key function to efficiently sort universities alphabetically while ignoring common prefixes; and utilizes a linear search with a counter to count the number of universities with satisfaction scores above 85%.

This technical documentation outlines the processes and algorithms implemented to enhance the functionality of the university ranking system. By addressing key problems such as sorting by student satisfaction, locating specific universities, and counting high satisfaction institutions, the system offers a powerful tool for journalists and researchers. These solutions not only streamline data access but also provide critical insights, aiding in the dissemination of valuable information about higher education institutions.

# Theory

The aim is to efficiently manage and analyze university data to provide accurate rankings and facilitate informed decision-making. To achieve this, careful consideration is given to data formats and algorithm selection, ensuring that the system can handle large datasets while maintaining performance and efficacy.

The primary idea involves the creation of an efficient data management framework that employs appropriate algorithms and data structures. For sorting universities by student satisfaction, a modified selection sort algorithm is utilized, allowing for the easy identification of the top 40 institutions. In addition, a linear search algorithm is implemented to quickly locate specific universities within the sorted list. This method is chosen for its simplicity and effectiveness, especially given the relatively small size of the dataset segment being searched.

For sorting universities alphabetically, a custom key function is used in conjunction with Python's Timsort algorithm. This approach ensures that universities are sorted in a user-friendly order, ignoring common prefixes like "University of".

Finally, to count the number of universities with student satisfaction scores above 85%, a linear search with a counter is employed. This method ensures that the system can efficiently aggregate data and provide valuable insights into overall student satisfaction.

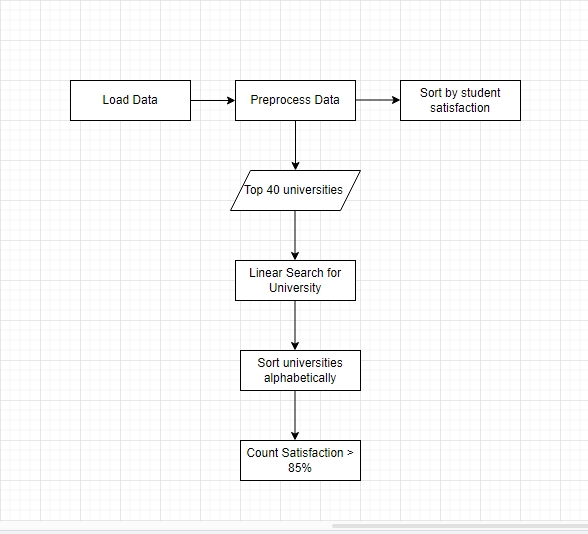


Figure 1: Chart of the current system

## Data Structures

### List

A list is a built-in data structure in Python that holds an ordered collection of items. Lists are mutable, meaning their elements can be changed after the list is created. It is a sequence data type that is able to hold multiple elements of same or different types of data type [(Samad, S.R. *et al., 2023*).](#link1)

In this system, the list ‘lst’ is used to store the university data with each element being a tuple containing various attributes of university. For operations like; sorting universities by student satisfaction, finding specific universities from the top 40 university’s list, etc. lists are used to store and handle the data.

A computer code with text

Description automatically generated with medium confidence

Figure 2:List data structure implementation

### Tuple

Tuples are fundamental data structures in Python that provide ways to store and manage collections of multiple items in a single variable [(*Python Tuples*)](#link2). Tuples are immutable, so changes with elements is not possible while using tuples, that’s why tuples are best to use when the dataset are fixed and unchangeable.

In this case, each university's data is stored as a tuple within the list ‘lst’. This tuple contains all the attributes of the university, such as name, student satisfaction, entry standards, etc. which are a fixed standard that bases university’s ranking, searching and other operations.



Figure 3: Tuple implementation

### List of Tuples

A list of tuples is a data structure where each element of the list is a tuple. This combination allows for storing a collection of records, where each record is a tuple containing multiple related items.

For this case, the selection sort algorithm operates on the list , sorting the tuples based on the student satisfaction score (a specific element in each tuple). Likewise, the linear search algorithm operates on the sorted list of tuples, scanning each element to find a specific university. Also, to count universities with student satisfaction higher than that of 85%, the algorithm iterates through each tuple, checks the satisfaction score, and increments a counter if the score exceeds 85.

A close up of a text

Description automatically generated

Figure 4: List of Tuples implementation

### Pandas DataFrame

Pandas DataFrame is a tabular formed data structure with labeled rows and columns that is two-dimensionally mutable in size [(GeeksforGeeks, 2024).](#link3) It is similar to a spreadsheet or SQL table, or a dictionary of Series object.

In this implementation, a pandas DataFrame is used to load, clean, and preprocess university ranking data from a CSV file. The 'Index' column is dropped, and percentage signs are removed from relevant columns, converting them to floats. The DataFrame is then converted into a list of tuples for further processing. This allows for structured and efficient data manipulation before sorting, searching, and counting operations are performed.

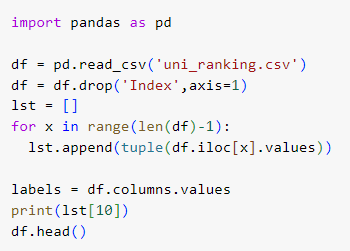


Figure 5: Pandas DataFrame implementation

## Algorithms

### Selection Sort

Selection Sort is a simple comparison-based sorting algorithm. It divides the input list into two parts: a sorted sub-list of items which is built up from left to right at the front (left) of the list and a sub-list of the remaining unsorted items. The algorithm repeatedly selects the smallest (or largest, depending on the sorting order) element from the unsorted sub-list, swaps it with the leftmost unsorted element, and moves the sub-list boundaries one element to the right.

The function ‘selection\_sort\_by\_satisfaction(lst)’ sorts the list of tuples ‘lst’ based on the student satisfaction score (index 3 of each tuple). It finds the university with the highest satisfaction in the remaining unsorted part of the list and swaps it into the current position.

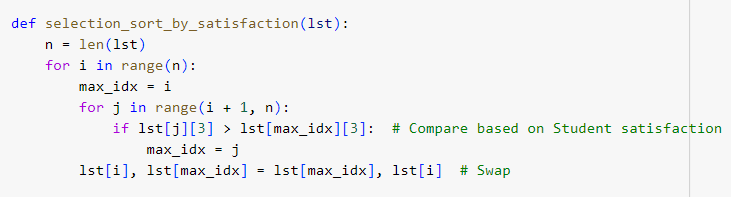


Figure 6: Selection sort algorithm code

### Linear Search

Linear Search is a straightforward search algorithm that checks every element in the list sequentially until the desired element is found or the list ends. It operates in O(n) time complexity.

The function ‘linear\_search\_university(lst, university\_name)’ searches for a specific university by name in the top 40 universities sorted by student satisfaction. It iterates through the list and compares each university's name to the target name.

A white background with black text

Description automatically generated

Figure 7: Linear search code

### Custom Sort

Python's built-in sorted function sorts elements using a custom key. This custom key can be defined using a lambda function or a separate function, allowing for complex sorting criteria.

The function ‘sort\_universities\_by\_name(lst)’ sorts the list of universities by name. It uses a custom key to ignore the prefix "University of" in the names, allowing for a more intuitive alphabetical sorting.

A computer code with many colored text

Description automatically generated with medium confidence

Figure 8: Timsort implementation

### Counting with a Condition

Counting elements with a condition involves iterating through the list and incrementing a counter each time an element satisfies a specific condition.

The function ‘count\_high\_satisfaction\_universities(lst, threshold=85.0)’ counts the number of universities with a student satisfaction score greater than a specified threshold (default is 85.0). It iterates through the list and checks if the satisfaction score (index 3 of each tuple) exceeds the threshold.

A computer code with text

Description automatically generated with medium confidence

Figure 9: Counting with a condition code

## Empirical Basis Implementation

This section provides an empirical basis for the Python-based university ranking system implemented. The system is designed to sort, search, and analyze university data based on student satisfaction ratings, utilizing a menu-driven interface for user interaction.

### **Data Loading and Preparation**

University ranking data loads from CSV file into pandas DataFrame for its data manipulation capabilities. "Index" column drops due to lack of relevance to analysis. Percentage strings representing satisfaction ratings convert to float values to enable numerical operations like sorting and comparison. DataFrame transforms into list of tuples for simpler manipulation in later steps. This transformation facilitates easier sorting and searching of the information.

A screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

Figure 10: Data loading and preprocessing code

### Sorting Functions

The system implements two sorting functionalities: selection sort by satisfaction and sort universities by name.

* **Selection sort by satisfaction:** This function utilizes the selection sort algorithm to arrange universities based on student satisfaction ratings. These ratings reside at the third position within each tuple. The function iterates through the list, identifying the highest satisfaction rating and swapping it to the beginning, resulting in a descending order list.

A computer screen shot of a code

Description automatically generated

* **Sort universities by name:** This function sorts universities alphabetically based on their names. It incorporates a key function, named "university\_name\_key," which removes the prefix "University of " from the names to achieve consistent sorting. This function leverages Python's built-in "sorted" method for efficient sorting.

A computer code with text

Description automatically generated with medium confidence

### Search and Count Functions

The system expands its capabilities with two new functions: linear search by university and counting universities with high satisfaction.

* **Linear search by university :** This function utilizes a linear search approach to locate the position (index) of a particular university within the list. If the university exists, it returns the index. Otherwise, it returns -1 to indicate the university wasn't found.

A screen shot of a computer

Description automatically generated

* **Counting universities with high satisfaction:** This function calculates the number of universities where student satisfaction surpasses a specific threshold. By default, this threshold is set at 85%. These functions empower users to search for specific universities and analyze the spread of satisfaction ratings.

A computer screen shot of text

Description automatically generated

### Main program

The system offers a menu-driven interface through the main() function, facilitating user interaction with the ranking system. This interface presents users with a recurring menu offering various options: sorting universities by student satisfaction, searching for a specific university within the top 40 based on satisfaction, sorting universities alphabetically, counting universities exceeding an 85% satisfaction threshold, and exiting the program. The system executes the corresponding function based on the user's selection. Sorting by satisfaction displays the top 40 universities. Search functionality returns the index of the sought-after university within the top 40 list. This design ensures a user-friendly command-line experience for analyzing university rankings.

A screenshot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

A computer screen shot of a program

Description automatically generated

Figure 11: Main program code

## Comparisons

There has been use of different types of data structures and algorithms in this system, for a certain problem there is always a different approach or alternatives that can be used, so here’s discussed the differences and the reason for the use of the particular data structure or an algorithm.

### Pandas dataframe and list of tuples

The list of tuples is a simple and intuitive data structure, where each tuple represents a university and contains various attributes such as name, entry standards, student satisfaction, etc. This approach provides quick access to elements via indexing and is straightforward to use. However, its simplicity comes with limitations, especially when dealing with larger datasets. The list of tuples is less flexible and requires linear search times, making it inefficient for extensive data manipulation or retrieval operations.

On the other hand, a Pandas DataFrame offers a more sophisticated structure, providing labeled axes and supporting complex operations. It is highly optimized for large datasets, allowing for more efficient data manipulation, filtering, and analysis. The primary disadvantage of using a DataFrame is the higher memory usage and the complexity of implementation, especially for users unfamiliar with the Pandas library. Despite these drawbacks, the Pandas DataFrame is generally preferred for larger datasets due to its powerful data handling capabilities.

### List of tuples and dictionary of lists

Using a list of tuples is straightforward and ensures that data remains ordered as inserted, making it easy to iterate over. However, this structure becomes inefficient as the size of the dataset increases, due to linear search and the difficulty of updating individual elements.

A dictionary of lists offers faster lookups by key and greater flexibility in data manipulation. Each key in the dictionary can correspond to a different attribute, enabling quicker access and updates. The primary downside is the increased complexity of the data structure and slightly higher memory usage. This method is particularly useful when frequent updates and lookups are needed, providing a balance between performance and complexity.

### Selection sort and Quick sort

Selection Sort is a simple algorithm that works by repeatedly finding the minimum element from the unsorted part and putting it at the beginning. This algorithm is easy to understand and implement, making it suitable for small datasets.

Quick Sort, a divide-and-conquer algorithm, is more efficient on average, with a time complexity of O(n log n). Quick Sort is generally faster and more efficient than Selection Sort, especially for larger datasets. However, it is more complex to implement and can degrade to O(n^2) in the worst case, though this can be mitigated with proper pivot selection i.e. for small dataset situation like this.

### Linear search and Binary search

Linear Search is a straightforward algorithm that checks each element in the list until it finds the target value or reaches the end of the list. Its simplicity makes it easy to implement, and it performs adequately for small datasets.

Binary Search, comparatively is much faster with a time complexity of O(log n), but it requires the data to be sorted. This algorithm works by repeatedly dividing the search interval in half, significantly reducing the number of comparisons needed. The main disadvantage is the need for pre-sorted data.

### Custom Sort with key function and Prefix tree

Using a custom sort with a key function in Python is flexible and straightforward. It allows for customized sorting criteria, such as ignoring certain prefixes in university names. This method leverages Python's built-in sorting algorithms, which are highly optimized, making it both easy to implement and efficient for most cases.

Alternatively, a Trie (Prefix Tree) is an efficient data structure for prefix-based operations and lookups. It is particularly useful for handling large sets of strings where common prefixes need to be managed. However, Tries are more complex to implement and require more memory. For datasets where the primary requirement is sorting with custom criteria, the custom sort with a key function is simpler and sufficiently efficient.

### Counting with a condition and Binary indexed tree

Counting with a condition involves iterating through the list and counting elements that meet a specific criterion. This approach is straightforward and easy to implement, with a time complexity of O(n). It is suitable for small to medium-sized datasets, making the operations much more efficient.

A Binary Indexed Tree (Fenwick Tree) is a more advanced data structure that supports efficient cumulative frequency queries with a time complexity of O(log n). It is useful for dynamic datasets where updates and prefix sums are frequent. However, it is more complex to implement and requires the data to be structured appropriately. For the given problem, where the requirement is to count elements based on a condition, the simplicity and ease of implementation of a straightforward iteration make it a practical choice.

# Operational Assessment

The performance analysis of the university ranking system focuses on evaluating the efficiency and scalability of the various components, including data loading, sorting, searching, and counting functions. The performance of these components is critical to ensure the system can handle large datasets efficiently.

* 1. Data Loading and Preparation

Loading the CSV file using pd.read\_csv() is typically O(n), where n is the number of rows. Dropping a column with drop() is O(1) as it involves a single operation on the DataFrame's metadata. Converting percentage strings to floats using a loop through columns is O(m\*n), where m is the number of columns and n is the number of rows. Converting the DataFrame to a list of tuples is O(n). Overall, the data loading and preparation steps are linear with respect to the number of rows and columns, making them efficient for reasonably large datasets.

* 1. Sorting Functions

The selection sort algorithm has a time complexity of O(n^2) due to the nested loops. This quadratic complexity means that performance degrades significantly as the dataset size increases. The sort\_universities\_by\_name function uses Python’s built-in sorted() function, which implements Timsort. Timsort has a time complexity of O(n log n), making it significantly more efficient than selection sort.

* 1. Search and Count functions

The linear search algorithm has a time complexity of O(n), where n is the number of universities in the list. It iterates through the list to find the matching university. Counting universities with high satisfaction involves a single pass through the list, resulting in a time complexity of O(n). This operation is efficient and scales linearly with the size of the dataset, making it suitable for small as well as large datasets.

* 1. Main program

The main function itself does not perform intensive computations. Its performance depends on the underlying functions it calls (sorting, searching, and counting).

The most of the used algorithm’s efficiency in case of large dataset is very poor but in terms of the university ranking dataset this system is based on is relatively a small dataset which makes these algorithm now more effective.

## Asymptotic analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data structure & Algorithm | Space complexity | Worst case time complexity | Average case time complexity | Best case time complexity |
| Pandas DataFrame | O(n) | O(m\*n) | --- | O(1) |
| Selection sort | O(1) | O(n^2) | O(n^2) | O(n^2) |
| Timsort | O(n) | O(n log n) | O(n log n) | O(n log n) |
| Linear search | O(1) | O(n) | O(n) | O(1) |
| Count algorithm | O(1) | O(n) | O(n) | O(n) |

Table 1: Asymptotical analysis of the data structures and algorithms

# Assertion Table

|  |  |  |  |
| --- | --- | --- | --- |
| Assertion | Code | Expected Result | Functionality |
| Top 40 universities by student satisfaction | A screenshot of a computer code  Description automatically generated | When choice is ‘1’,  Top 40 universities and their respective weightage in other aspects are listed. | working |
| Search a particular university |  | When choice is ‘2’, after giving the university name as input, the rank of the university among the Top 40 is shown or if not present in the list prompts output as not being in Top 40. | working |
| Sort universities by name (alphabetically) |  | When choice is ‘3’, list of the alphabetically ordered universities name alongside their information is shown. | working |
| Count universities with satisfaction > 85% |  | When choice is ‘4’, number of the universities with student satisfaction above 85% is shown. | working |

Table 2: Assertion table of the code

# Conclusion

This documentation details the development of an efficient university ranking system that manages and analyzes university data based on key performance indicators such as entry standards, student satisfaction, research quality, research intensity, and graduate prospects. The system employs a modified selection sort algorithm to rank universities by student satisfaction, making it easy to identify the top 40 institutions. It also uses a linear search algorithm to quickly locate specific universities within this top tier. For alphabetical sorting, Python’s Timsort with a custom key function ensures user-friendly ordering by ignoring common prefixes like "University of." Additionally, a linear search with a counter is implemented to count universities with satisfaction scores above 85%, providing valuable insights into overall student satisfaction.

These robust algorithms and data structures ensure the system is both efficient and user-friendly, capable of handling large datasets while maintaining performance and accuracy. This makes the university ranking system a valuable tool for journalists, researchers, and other stakeholders interested in higher education. By addressing key challenges in sorting, searching, and data aggregation, the system enhances data accessibility and usability, offering critical insights into the comparative performance of higher education institutions.

# References

Samad, S.R. *et al.* (2023) ‘Organizing data using lists’, *Advances in Systems Analysis, Software Engineering, and High Performance Computing*, pp. 35–51. doi:10.4018/978-1-6684-7100-5.ch002.

*Python Tuples*. <https://www.w3schools.com/python/python_tuples.asp>.

GeeksforGeeks (2024) *Python Pandas DataFrame*. https://www.geeksforgeeks.org/python-pandas-dataframe/.

# Appendix-I (Code)

A screenshot of a computer code

Description automatically generatedA computer screen shot of a program code

Description automatically generated

A close-up of a computer screen

Description automatically generated

A computer code with text

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer code with many text

Description automatically generated with medium confidence

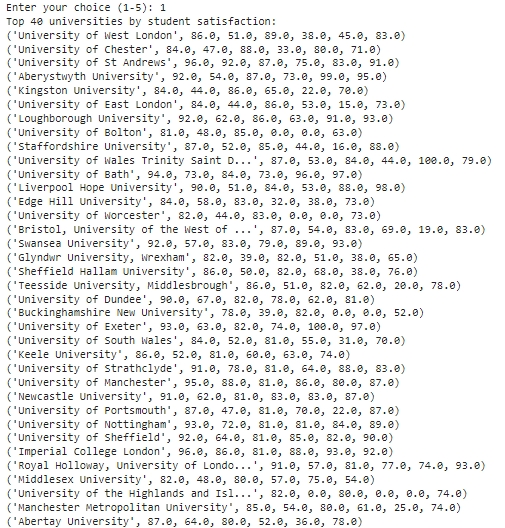
A computer screen shot of a program

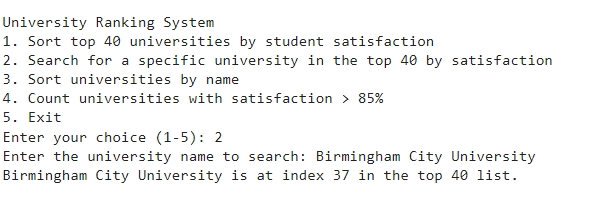
Description automatically generated

# Appendix-II (Output)

A screenshot of a computer

Description automatically generated





A screenshot of a computer program

Description automatically generated

A white text with black text

Description automatically generated

A screenshot of a computer program

Description automatically generated