**Self-Assessment:** Please highlight where you think your report grade should be. Example below.

|  |  |  |
| --- | --- | --- |
| **Criteria** | Write simple algorithms using appropriate discrete data structures to solve computational problems (LO3) | Use appropriate methods to analyse the efficiency and correctness of algorithms (LO4) |
| **Weight** | **25%** | **25%** |
| **0 – 29%** | The algorithm does not solve an appropriate problem or has serious errors. There is little or no discussion of how the algorithm works.  No discrete data structure has been used, or the choice of data structure is inappropriate. | The analysis is limited and seriously flawed. |
| **30 – 39%** | The algorithm solves part of an appropriate problem. There may be substantial aspects of the problem which are not attempted or explained, or errors in the solution.  The explanation is unclear or missing important details about how the algorithm works. | An attempt has been made to analyse an algorithm, but appropriate methods of analysis were not used, and the results of the analysis may be incorrect or meaningless. |
| **40 – 49%** | A rudimentary algorithm solving a basic problem. There may be some errors which could be corrected with further work.  There is a limited discussion of how the algorithm works. The choice of data structure is inappropriate, or unjustified. | An attempt has been made to measure the running time of the algorithm for some inputs, but the methodology is unclear or the measurement may be inaccurate. There is a limited discussion of some other issues relating to efficiency.  Analysis of the algorithm’s correctness is vague, or not attempted. |
| **50 – 59%** | The algorithm solves an appropriate problem, though it may have minor errors or fail to account for special cases. There is an explanation of how the algorithm works.  The choice of data structure may be inappropriate or poorly justified. | The running time of the algorithm has been measured accurately for an appropriate range of inputs, and the methodology has been explained. There is some discussion of other issues relating to efficiency.  There is a basic or informal analysis of the algorithm’s correctness. |
| **60 – 69%** | The algorithm correctly solves an appropriate problem. There is a clear explanation of how the algorithm works.  At least one appropriate data structure has been used, and the choice has been adequately justified. | The efficiency of the algorithm has been accurately measured using an appropriate methodology, which has been explained. The measurements may include more than one metric.  There is an analysis of the algorithm’s correctness, which may specify pre- and post-conditions for part of the algorithm. |
| **70 – 79%** | The algorithm correctly solves a challenging problem. There is a clear explanation of how the algorithm works, and the explanation makes clear references to the relevant parts of the source code.  Appropriate data structures have been used, and justification is given for each with reference to the specific problem. | The efficiency of the algorithm has been accurately measured using an appropriate methodology, with multiple metrics and a clear explanation. The asymptotic complexity of the algorithm is given. The efficiency may be compared with appropriate alternative algorithm(s).  There is a formal analysis of the correctness of at least part of the algorithm. |
| **80 - 90%** | A well-designed algorithm which correctly solves a challenging problem. There is a clear, detailed explanation of how the algorithm works, with clear references to the relevant parts of the source code.  Appropriate data structures have been used, and justification is given for each with reference to the specific problem. | The efficiency of the algorithm has been accurately measured using an appropriate methodology, with multiple metrics and a clear, detailed explanation. The asymptotic complexity of the algorithm is given. The efficiency has been compared with appropriate alternative algorithm(s).  There is a detailed formal analysis of the correctness of the algorithm. |
| **90 – 100%** | An excellent algorithm written, explained and evaluated to the highest standards. | An excellent analysis of the efficiency, complexity and correctness of an algorithm, conducted and explained to the highest standards. |

**A close-up of a logo

Description automatically generated**

**Technical report: College Exam Results (SAT)**

**Authors: Yuthika Rani Timalsina**

**Date: 31st May 2024**

**Word Count: 1819**

**Page Count: 24**

**Confidential: NO / YES – INTERNAL ONLY/YES – X**

Table of Contents

[Introduction 5](#_Toc169695616)

[Theory 6](#_Toc169695617)

[Data Structure 6](#_Toc169695618)

[List: 6](#_Toc169695619)

[Dictionary: 6](#_Toc169695620)

[Pandas Data Frame: 7](#_Toc169695621)

[Algorithm 7](#_Toc169695622)

[Merge sort: 7](#_Toc169695623)

[MaxHeap: 7](#_Toc169695624)

[Implementation 8](#_Toc169695625)

[Time Complexity 8](#_Toc169695626)

[Space Complexity 10](#_Toc169695627)

[Assertion Table: 12](#_Toc169695628)

[Conclusion 13](#_Toc169695629)

[References 14](#_Toc169695630)

[Appendix A 15](#_Toc169695631)

[Appendix B 18](#_Toc169695632)

**Exclusive Summary**

This technical report delves into the dataset detailing SAT performance among college-bound seniors in New York City during 2010. It encompasses data from 460 schools, featuring unique identifiers (DBN), school names, and average SAT scores in Critical Reading, Mathematics, and Writing. Additionally, the dataset includes counts of test takers per school, offering insights into participation levels.

Employing data structures like lists and dictionaries, along with algorithms such as Merge Sort and Max Heap, this analysis efficiently manages and examines extensive datasets. Notably, the Max Heap algorithm facilitates identifying schools with the highest average SAT scores, aiding educators in adopting successful practices. Meanwhile, Merge Sort categorizes schools by performance in specific SAT sections, highlighting strengths in Critical Reading, Mathematics, or Writing.

The report computes average scores per SAT section, provides foundational performance metrics, and features a user-friendly tool to locate schools based on their initial alphabetic character. It acknowledges the limitations of SAT scores in capturing all aspects of academic achievement, advocating for further research. Ultimately, by leveraging data structures and algorithms, this report underscores their pivotal role in analyzing comprehensive datasets to inform educational practices and policy decisions, promoting equitable education in New York City.

# Introduction

College board has created a dataset which has detailed information about the critical reading, math and writing results of high school graduates in NY from 2010. The average scores for every school are available in this dataset, and other unique identifiers include DBN identification number or school name and the number of test takers per school. Therefore, there is availability of these records for deeper analysis of test rates and school performance.

It’s important to look at this data set because you want an understanding of the education system in New York City. Differences with respect to the SAT points are possible; hence a result could be disparities in terms of school size and budgeting among others. This analysis details areas that could be corrected in educational resource and strategy provision to ensure that all students are fairly treated.

The increased code by us confronts various obstructions usually encountered in the analysis of educational data. One such obstacle solved by our approach is providing a tough cut-off value that ensures no data point or its feature could be carried into further analysis simply because they are missing, and this could prevent any useful knowledge extraction. Secondly, we develop an unsystematic procedure means of calculating average points, arranging them in a top-down order depending on score counts per establishment, and finally determining the best schools that help encourage competition through better performance target setting techniques. In conclusion the code groups schools by score into ranges and orders them according to the alphabet, which makes it easy for a person to go through the information. Taken together these changes make it easier to understand and scrutinize SATs performances in NYC schools.

# Theory

The analysis depends on two significant data structures and algorithms: Development Sort and Maximum Heap. Development Sort is a stable sorting algorithm that recursively divides the input list into halves until each sub list only has one element, which is intrinsically sorted. Afterward, the sorted sub lists are merged in a sorted order to get O(n log n) time complexity. A binary tree-based data structure known as a Max Heap that maintains a hierarchical structure whereby all child nodes have values less than the parent node so the root of the entire tree always contains the maximum element. Since O(log n) operations are used for insertions or extractions of the max element, it is a good way to pick out high values.

# Data Structure

## List:

Lists are collections of elements that are ordered in a specific way. Thus, you can obtain each one successively as it has its position or index [(Tagliaferri, 2021)](#Tag). For example, it is possible to add or remove some entities later on this data structure type which is also known as mutable. Lists store data about each school, including individual SAT scores as well as other pertinent details. Another way that lists are used is in holding those schools that have been entered into this specific Max Heap structure where each item stands for an institution’s average exam result alongside its name.

The ‘SAT\_\_College\_Board\_\_2010\_School\_Level\_Results\_20240506.csv’ file is displayed as a wide table that contains details per school. During scripting in Python, we create tables using lists. Data is loaded into a Pandas Data Frame, consisting of lists that represent columns and rows. Afterwards, this code changes Data Frame to a list containing many lists called schools data within which each sub-list corresponds to a row in the table and contains some information about only one institution.

## Dictionary:

Data is stored in dictionaries using key-value pairs; each key corresponds to only one value. [(2024)](#Dict). The keys can be used to identify its respective values, and they share no sequence as is the case with lists. This implies that you cannot be sure of the arrangement in which you enter these pairs. They are best for associating one piece of information to another like student identification number with their names or schools together with some average SAT marks.

The group\_schools\_by\_score\_range () function is designed to organize schools according to their score ranges by use of dictionaries. It basically takes the form of a "map" in which each score range (e.g., "300-400") is a key while the schools enclosed within it are values.

## Pandas Data Frame:

The Pandas Data Frame is a robust data structure used frequently in programs for handling and manipulating tabular data. A dataset is initially loaded into a Pandas Data Frame from a CSV file where columns represent SAT scores and other information related to each school while rows represent different attributes or records. Among the operations that can be performed on the Data Frame are those related to data cleaning (e.g., getting rid of spaces, putting zeros in place of missing values), calculating the mean SAT scores in verbal, mathematics and writing components and transforming resultant information into another form - lists for more detailed research. Educational data analysis benefits from the way in which it is organized along with other possibilities for computation that come with this.

# Algorithm

## Merge sort:

Merge Sort has a process based on divide and conquer which splits the list into halves until each sublist consists of only one item, then merge them back in a sorted order [(Carlsson et al., 1990)](#Car). In this code, merge\_sort(arr, column\_index) arranges in order of their SAT scores the elements of the data on schools (arr) with the argument colum\_index representing the SAT section. Therefore, you can easily get the best sections by calling the function: sort\_top\_schools\_by\_section (data, section\_column\_index, top\_count). Thanks to its stability as well as an average performance of O (n log n) for sorting.

## MaxHeap:

The heap property is maintained by operations in the MaxHeap class that heapify. Heapify Up is used to move the new element up by comparing and swapping with its parent until the heap property is restored during insertion. To maintain the heap, Heapify Down is used during deletion where the root is swapped with the larger child. The \_heapify\_up(index) and \_heapify\_down(index) are used to implement these operations, have a time complexity of O(logn).

# Implementation

Python script implementing SAT score analysis system uses different data structures and algorithms for handling provided dataset and drawing information from it. It uses data structures and algorithms to efficiently see, change and handle things and give users many analytical tools. Some of its main features include finding the best schools by average sat scores using a max heap, classifying schools according to performance in specific sat sections via merge sort, listing them alphabetically according to their names, calculating the average score per sat section and grouping them at equal intervals. The program has a main function which allows friendly interaction.

## Time Complexity

**A screenshot of a computer program

Description automatically generated**

**Fig:** **Code snippet of Maxheap**

**A screen shot of a computer program

Description automatically generated**

**Fig: Constructing MaxHeap for Top Schools**

The find\_top\_schools\_by\_average\_score function uses a max heap to retrieve quickly the schools having the highest average SAT scores. For this purpose, the time used is O(nlogn+klogn) where n is the number of schools and k is the number of high scoring institutions selected. When sorting – merge\_sort and sort\_top\_schools\_by\_section, we apply merge sort with a time complexity of 𝑂(𝑛log𝑛) to ensure fast sorting even when dealing with large data sets.

**A screen shot of a computer program

Description automatically generated**

**Fig: Iterating Over Schools to Sum Scores**

To calculate the average scores in calculate\_average\_scores we have to go through the data leading to a time complexity of 𝑂(𝑛⋅𝑚) where m is the number of sections over which averages were to be taken. On the other hand group\_schools\_by\_score\_range also has an time complexity of 𝑂(𝑛⋅𝑏) where b is the number given for score bins and it incurs this complexity through iterating the data to put schools in predetermined clusters based on how they scored.

**A screenshot of a computer program

Description automatically generated**

**Fig: Assigning Schools to Score Bins**

Finally, sorting and displaying school data that starts with the user-defined first letter in list\_schools\_by\_first\_letter has 𝑂(𝑛) time complexity because it goes through the dataset filtering schools by initial letter according to the user input. They present a package of tools which encourage efficient scrutiny along with ease of presentation of SAT score data from high schools alike.

## Space Complexity

**A screenshot of a computer program

Description automatically generated**

**Fig: Code snippet of MaxHeap Operation**

**A screenshot of a computer

Description automatically generated**

**Fig: Code snippet of Top Schools**

**A screen shot of a computer

Description automatically generated**

**Fig: Group Schools by Score Range**

The calculate\_average\_scores function has a space complexity of O(1) because it only uses a fixed amount of space for totals and counts, regardless of the number of schools or sections.

The space complexity of the group\_schools\_by\_score\_range function is O(n+b), where n represents the number of schools and b represents the number of bins. The function keeps references to the original school data for the groups.

The function list\_schools\_by\_first\_letter has a space complexity of O(n) because it creates a new list for the filtered results, potentially containing all schools in the worst-case scenario. Efficient management of space makes it possible for the code to manage large datasets without using too much memory.

## Assertion Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Assertion** | **Snip of Code** | **Condition** |
|  | Top count is a positive integer |  | **Working** |
|  | Top count is a positive integer |  | **Working** |
|  | Valid section column index (3, 4, or 5)  (sort\_top\_schools\_by\_section) |  | **Working** |
|  | Section column indices list is not empty(calculate\_average\_scores) | **A screen shot of a computer code  Description automatically generated** | **Working** |
|  | At least one valid score for calculation (calculate\_average\_scores) | **A screenshot of a computer screen  Description automatically generated** | **Working** |
|  | Valid section column index (3, 4, or 5) |  | **Working** |
|  | Bins list is not empty (group\_schools\_by\_score\_range) |  | **Working** |
|  | First letter is a single character (list\_schools\_by\_first\_letter) |  | **Working** |

# Conclusion

A closer look at the 2010 SAT performance data for New York City schools shows that there are considerable disparities in student achievement among various schools. While some schools have always recorded impressive results, others hardly achieve the set mean. This implies there is a requirement for comprehending more about what contributes to such gaps.

The review, which relies on using data structures like arrays and dictionaries as well as algorithms such as Merge Sort or Max Heap, demonstrates how data-driven methods can be used to discover the best schools, comprehend test outcomes as well as promote faster exploration of information. Nevertheless, the paper recognizes that there are many other dimensions to student achievement aside from the SAT score.

Findings of the study also revealed that there was a significant relationship between SAT scores, in relation to school resources, socioeconomic status and government’s educational policies. These findings should guide future research, therefore, towards a more comprehensive examination of how these aspects relate to one another in influencing students’ academic performance. The current investigation further demonstrates that better strategies which teachers can use while enhancing students’ performance exist.

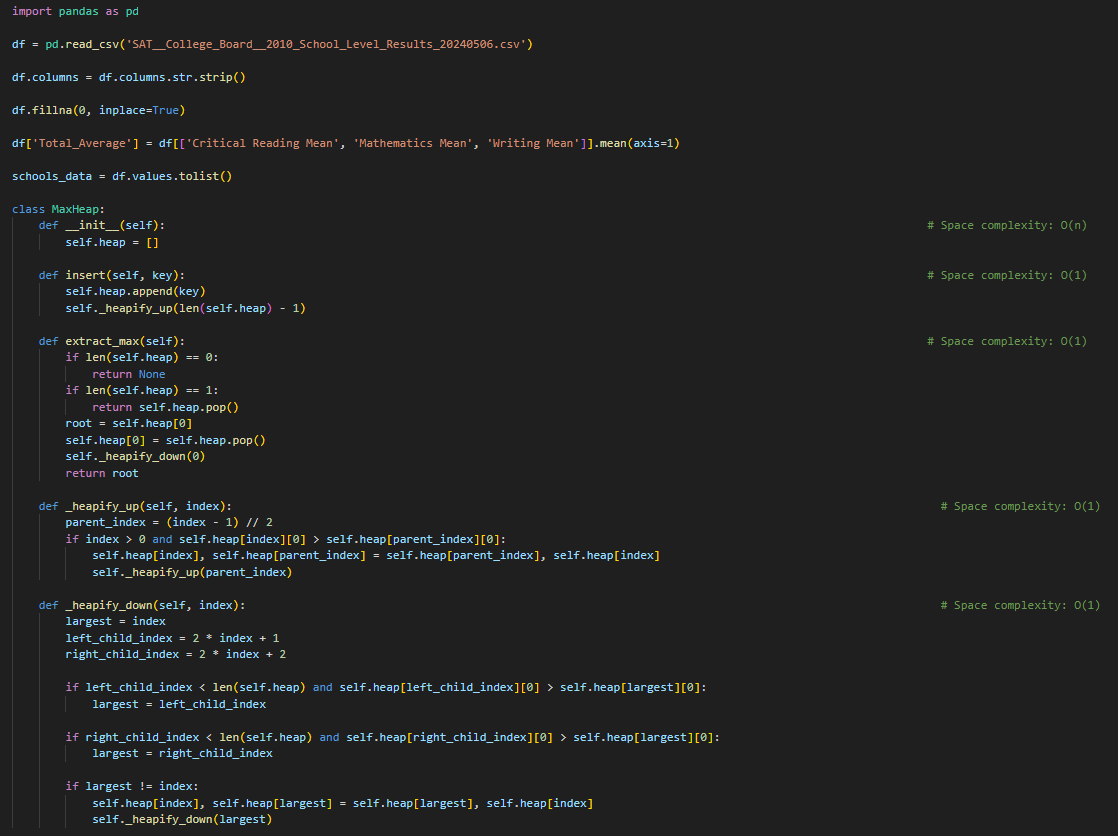
# References

Carlsson, S., Levcopoulos, C. and Petersson, O. (1990) ‘Sublinear merging and natural merge sort’, *Algorithms*, pp. 251–260. doi:10.1007/3-540-52921-7\_74.

*Dictionaries in python* (2024) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/python-dictionary/ (Accessed: 18 June 2024).

Tagliaferri, L. (2021) *Understanding lists in python 3*, *DigitalOcean*. Available at: https://www.digitalocean.com/community/tutorials/understanding-lists-in-python-3 (Accessed: 18 June 2024).

# Appendix A



A black screen with green lights

Description automatically generated

A black screen with white text

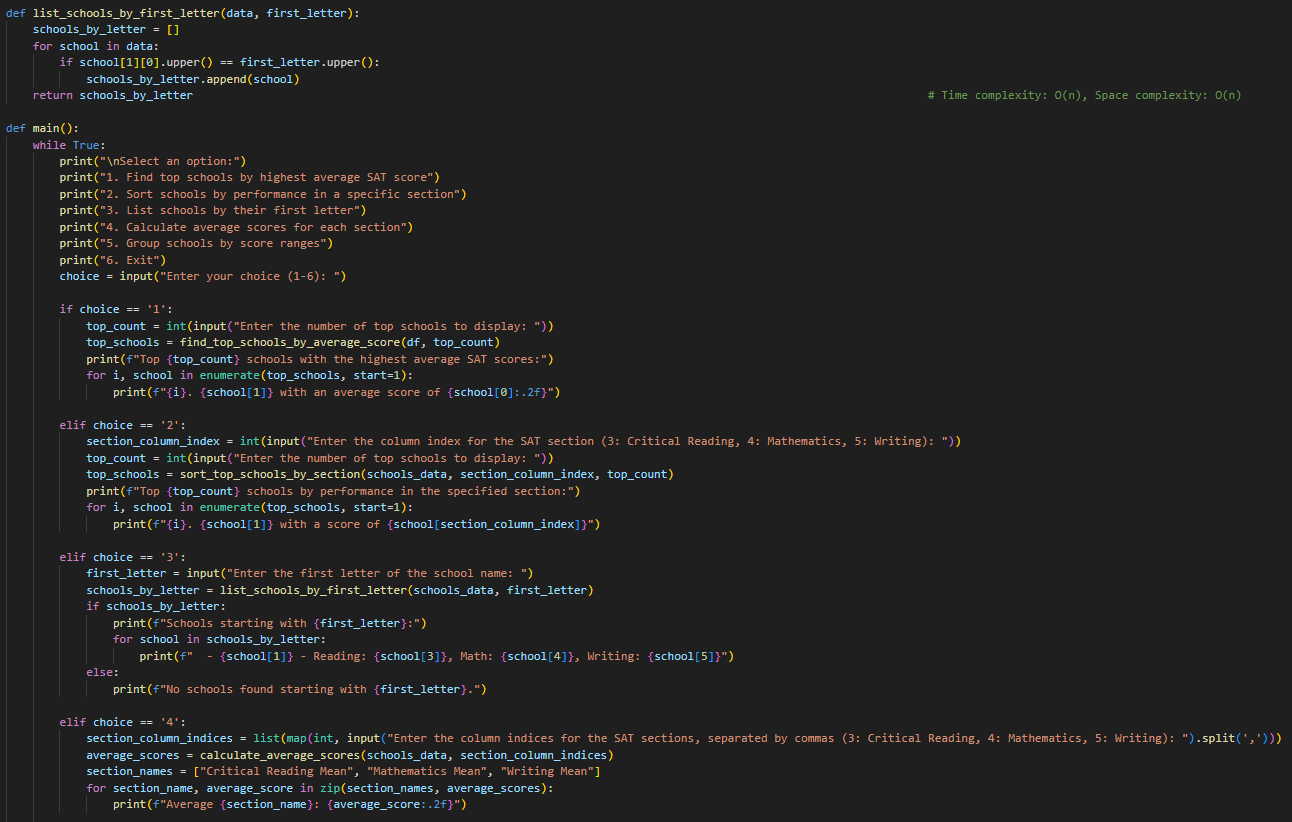
Description automatically generated

A screenshot of a computer

Description automatically generated

A black screen with a black background

Description automatically generated



A computer screen with colorful text

Description automatically generated

# Appendix B

***Output:***

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

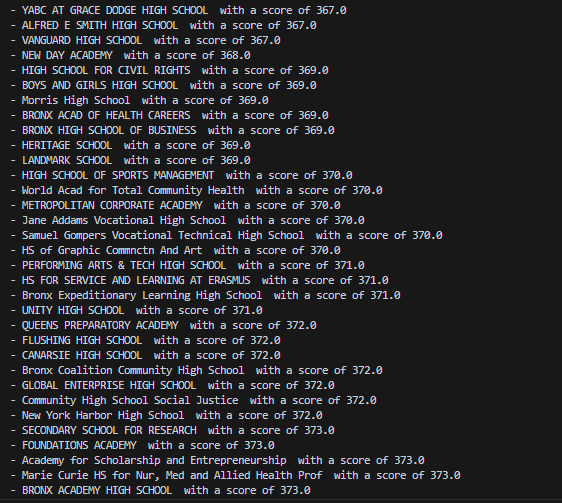
Description automatically generated

A computer screen shot of a program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA black screen with white text

Description automatically generatedA screenshot of a computer program

Description automatically generated A screenshot of a computer screen

Description automatically generatedA screen shot of a computer

Description automatically generated