

|  |  |
| --- | --- |
| **Name:** | Reece Benson |
| **Student No.:** | 16021424 |
| **Module:** | Design and Analysis of Data Structures and Algorithms |
| **Assignment No.:** | Assignment 2 |
| **Due Date:** | Thursday 22nd March, 2018 |

Table of Contents

[Table of Contents 2](#_Toc499806586)

[Brief 2](#_Toc499806587)

[Identify and Justify Data Structures 3](#_Toc499806588)

[Design a Solution for Ranking Players (pseudo code) 7](#_Toc499806589)

[Implement the Solution for Ranking Players (python code) 9](#_Toc499806590)

[Explain the algorithm implemented and justify its choice 10](#_Toc499806591)

[Design a Solution for Ranking on Prize Money (pseudo code) 11](#_Toc499806592)

[Implement the Solution for Ranking on Prize Money (python code) 11](#_Toc499806593)

[Discuss and justify your choice of entering match results 12](#_Toc499806594)

[Additional Pseudo Code for Main Features 14](#_Toc499806595)

[Monitoring User Input 14](#_Toc499806596)

[Loading Files 15](#_Toc499806597)

[Manual Input 16](#_Toc499806598)

[Generating Rounds 17](#_Toc499806599)

[File Save Handling 18](#_Toc499806600)

[Loading Previous Data 18](#_Toc499806601)

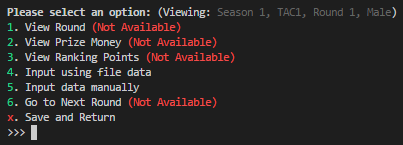
Brief

1. Design a solution to allow the processing of scores for both input types
2. Implement the solution designed in Task 1 in Python.
3. Design a solution for showing player information
4. Implement the design shown in Task 3
5. Evaluate the efficiency of your software
6. A second season will be introduced
7. Implement the design for Task 6

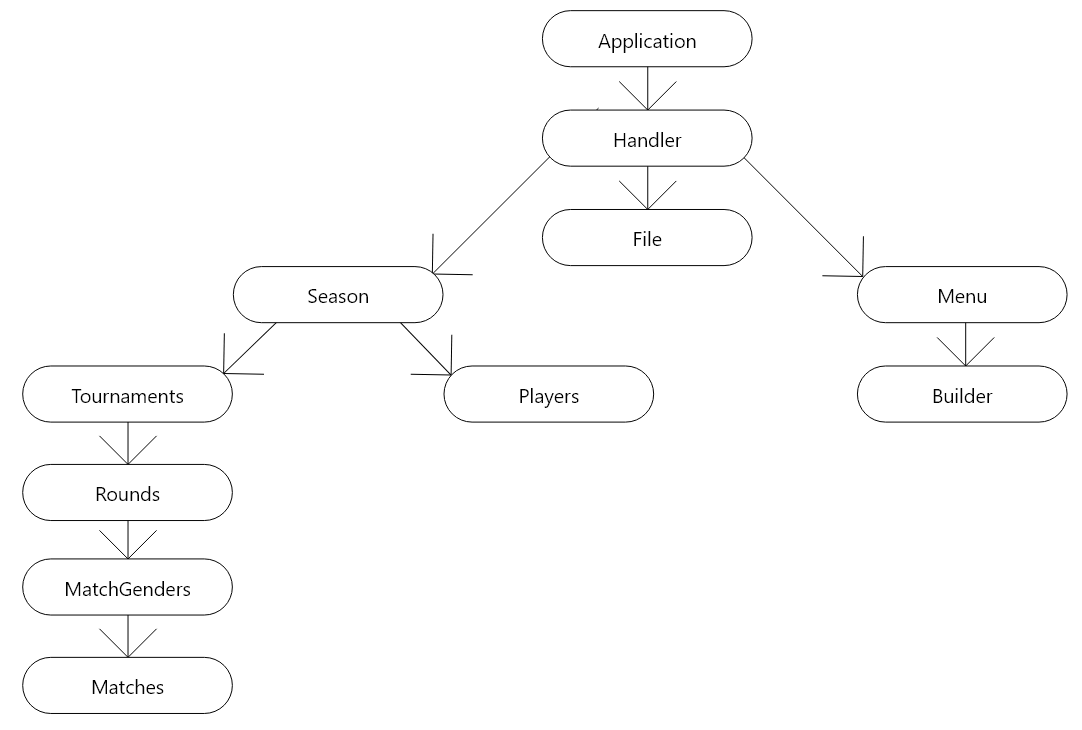
Design a solution to allow the processing of scores for both input types

For this task, we are to design a solution in order to allowing the follow features:

* Allow the processing of scores for both input types (manual input/file input)
* Upon the end of each round, provide the user with the winners and options to go to the next round or save and exit the application.
* Upon choosing the next round, provide the user with options to manually input data or read from the file. In my application, reading from file is disabled when the previous round data is modified, and no longer fits in line with the file data.
* Once all rounds have been completed, the winner will be declared and options to view the tournaments rankings will be shown. In my application, you can access the current tournament rankings at any round, to see the progress of each player.



In order to implement my solution, I visualised my application into a tree-like manner and followed the flow for my development process, for example:



## Pseudo Code for Task 1

“Allow the processing of scores for both input types”:



“Provide the user with the winners and options”:



“Once all rounds have been completed, the winner will be declared and options to view the tournaments rankings will be shown”:



Design a solution for showing player information (pseudo code)





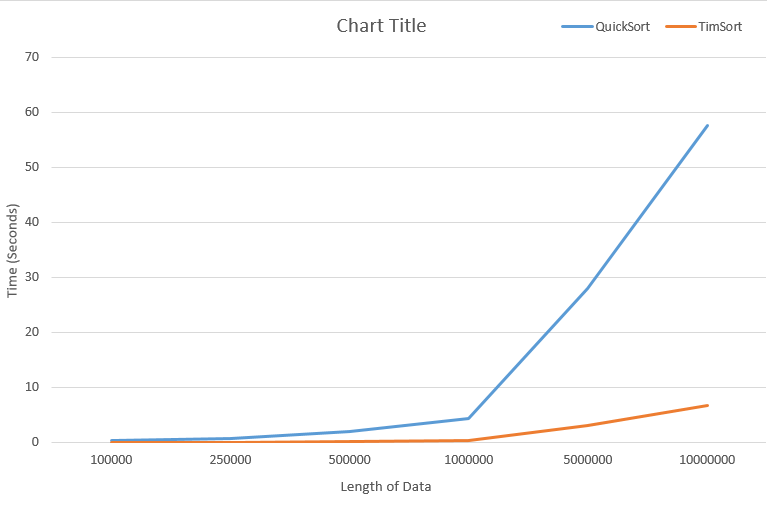


## Evaluate the efficiency of your software

For this section, I will be evaluating the efficiency of my software in terms of the size of code, speed of running and the efficient use of functions and specialist algorithms. Here, I implemented the use of **cProfile** to track the performance of functions I ran and the algorithms I used, and included some benchmarks of the comparisons between the algorithm I am using (Quick Sort), and Python’s in-built **sorted** algorithm (Tim Sort), however I stuck with Quick Sort as we were advised to use our own algorithm.

Here are the benchmarks of the differences between Quick Sort and Tim Sort; had I the ability to use Python’s in-built algorithm, I would’ve due to its amazing efficiency and the fact that in Python, Tim Sort is implemented in C. Having Tim Sort executed via C allows it to have a slight boost in performance in comparison to other sorting algorithms written in Python.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **100,000** | **250,000** | **500,000** | **1,000,000** | **5,000,000** | **10,000,000** |
| **Quick Sort** | 0.401s | 0.824s | 1.968s | 4.429s | 27.965s | 57.691s |
| **Tim Sort** | 0.03s | 0.09s | 0.199s | 0.461s | 3.06s | 6.752s |



We can see here, that Tim Sort (BigO notation of O(n)), it represents that in the graph quite accurately. If I was to do the tests with a standard iteration (i.e. iterations of 1,000,000), the graph would represent that including with Quick Sort (BigO notation of O(n log n)), it also represents that in the above graph.

I then moved on to using **cProfile** that is a Python in-built library that is used to evaluate the time taken to execute a partition of code. The code I had written in order to view the ranking points and prize money includes the use of the Quick Sort algorithm I had implemented:

**Pseudo Code:**

****

**Python Code:**



***[ The next page contains the cProfile Statistics ]***

**cProfile Statistics for “view\_prize\_money()” of a specific tournament:**



**cProfile Statistics for “view\_ranking\_points()” of a specific tournament:**

****

**cProfile Statistics for Viewing Matches with Particular Score of All Tournaments within a Season:**

****

**cProfile Statistics for Viewing Player with Win Percentage of All Tournaments within a Season:**

****

**cProfile Statistics for Finding Player with Highest Amount of Wins of All Tournaments within a Season:**

****

**cProfile Statistics for Finding Player with Highest Amount of Loses of All Tournaments within a Season:**

****

Design a solution for the second season (pseudo code)

Due to this task being very confusing to read at first, I have separated and broken down the specification of this task into sub categories, and this is how I will list the pseudo code for this task.

**First Round Pairs (Manual Input Checks & Validation)**

**Second Round and Above Pairs (Manual Input Checks & Validation)**

**Points Difficulty Factor For Achievement**

**Combination of Season Overall Leaderboards**