

For a rugby club management system, I prefer to use a recursive Quicksort algorithm to sort the list of players and coaches. In terms of efficiency, Quicksort has an average time complexity of O(n log n), making it very efficient for sorting fairly large datasets. With this efficiency, it is very important to rank a large list of athletes in terms of quality.

And in general, Quicksort has much less space complexity compared to merge sort or other sorting algorithms. So only the data that requires extra logarithmic space due to the recursive call is sorted in place.

In terms of adaptability, Quicksort excels in its adaptability to other types of data sets, and briefly, it works well for different types of data distribution. Basically, quicksort fits perfectly for our system that has endless information.

This is very useful for sorting lists of people in a Rugby Club Management System that may have different attributes such as name, gender, email address etc.

Quicksort is a consistent sorting algorithm. That means the relative order of equal elements is preserved. Within a rugby club managing system, it may be desirable to maintain orders of equal elements example, order by name.

So if we compare it to alternatives, we will talk about merge sort. Merge sort is a worst-case time-complexity as well as recursive sorting algorithm. Even though merge sorting algorithm could offer great performance, its spat complexity is higher than quick sort. It is not ideal for large datasets. Generally, sorting in descending order is another sorting algorithm. That is based on O(n log n) theory that averages time complexity comparison. Nevertheless, its worst-case time complexity is same as Quicksort sorting. quicksort does better than heapsort practically, especially for random data distributions.

So, the recursive Quicksort algorithm is a great choice to sort the list of people in the "Rugby Club Managing System" because of its efficiency, space saving, time saving, adaptability and stability if we compare it to other recursive sorting algorithms. Other sorting algorithms would work, but the advantages we talked about will not be there.

A close-up of a text

Description automatically generated

Choosing an appropriate search algorithm is itself a very important requirement for easy, fast, and efficient data retrieval from large datasets. Choosing the right search algorithm for your people list application is critical to optimizing performance and ensuring a seamless user and user experience in the future, whether in a business context or at a more advanced level. After much thought, I decided to use the binary search algorithm used in the original CA1 based on its advantages and our new requirements.

Efficiency always comes first for every employee, especially in the computer industry. We look for the fastest and shortest way to do something. Among many reasons, the first reason I chose the binary search algorithm is the efficiency of searching an ordered data structure. we provide our users with quick and easy access to information. By looping the dividing the searching iteration in half, binary search shrinks the searching space at every iteration. And that results more efficient and greater search action. So, binary searching algorithm is the most used algorithm in other searching algorithms.

With binary search, the time taken by the search results naturally increases with the size of the dataset, but not proportionately. It is a piece of cake to implement and understand for binary search. And that makes it a good choice for our club form. So, this is the one of the reasons why it has been chosen for our system. Generally, birany search has got many advantages, there are other searching algorithm that have got many advantages as well. But I have no clue how much difference there would appear if I chose others. I can only say the difference between binary and linear searching algorithms.