Zachary Meisner

Professor Atkison

9/26/21

4-1 Programming Journal

What were the **strengths** of the pair in solving the problem? In other words, why do you think the provided data structure and algorithm made sense as a pair for the given task? Support your claims with specific evidence from the course materials.

The provided data structure and algorithm makes sense as a pair for the given task due to the number of utilities that we have available. Not only are we able to use an array of algorithms such as Selection Sort, Quick Sort, or Recursive Algorithms, but we can also utilize functions specific for each of these types of algorithms such as partitions, and pivots. The Data Structures that we must focus on are data structures that essentially act as either permanent or temporary "homes" for our data as the encapsulation is pertinent to the livelihood of the algorithms implemented. The more popular example we use in this week are linked lists, stacks, and queues. Stacks and Queues specifically can be thought more of as Abstract Data Types, in other words, if you were to create a linked list, and impose a certain ruleset on the structure of the linked list, you would be creating an expression of an abstract instance almost as if you had a square and pulled one of the corners out to create a rhombus. The square may still be quadrilateral, but the new shape, the rhombus, cannot be a square. The ADT in this sense, realistically is more refined specifically for the outcome of the program and the way that you want it to work, which is one of the big reasons that the pairs chosen can be so strong with one another. Selection sort for example, is easier to use and specifically for more generalized problems that you are trying to solve. If you were to run a smaller program then you may want to use selection sort as it cuts the square into two halves, and then properly puts the values in the correct order based on the logical deductions. On the other hand, Quick Sort, is referred to as being one of the best sorting algorithms we have available. Not only is Quick Sort able to compute bigger data sets faster, but more specifically uses recursion in order to help accomplish this task by partitioning and compartmentalizing the data within the data set imposed onto the program

What were the **weaknesses** in the pair for solving the given problem? In other words, what was it about either the data structure or the algorithm that might make you consider alternatives? Alternatively, if you feel there were no significant weakness in the pairing, provide a circumstance where this pairing would be inadequate for solving a problem. Support your claims with specific evidence from the course materials.

The significant weaknesses of these pairs are ironically the strengths that it has as well. Not only are these instances created for very specific expressions and circumstances, but they might not work as well with other data sets or algorithms, leading you to waste your time, or even come to an incorrect conclusion. The reason I am saying this is because if you were to run a Quick Sort Algorithm on a pre-sorted data set, you would get its worst-case performance comparatively to something like Bubble Sort, which is not only easier to implement within a program. In addition to that, if you were to compare Selection Sort and Bubble Sort with Quick Sort, neither do well with big lists of data, and Selection Sort is much slower to use. So we have used our linked list, to narrow down to parameters where we are able to implement any amount of ADT's, only to have the possibility to potentially implement these environments incorrectly, compute something an in a misleading way, and pushing us into a hole where it is more difficult to come back from in the case there are issues, meaning that it would be harder to implement security functions and features unless they in addition to the abstract data types, followed some of the same parameters of the linked list, or even added more parameters to the abstract data types, making the program take longer, making it a less efficient choice overall for the big picture of the software.

Brainstorm **other possible applications** for the pair. How could the pair be used to solve problems relating to your own personal or professional interests? Illustrate your response with specific details.

Other possible applications for the pair I think are good for bigger sets of data, (Quick Sort), or specified instances of small amounts of data (Selection Sort) In other words, if I had wanted to use Quick Sort to help create an inventory system for Amazon, that maybe optimal, because not only would I be able to help take inventory and keep track of everything inside of my warehouse, but I could compartmentalize each parts of the list with Quick Sort to run an additional algorithm on the compartmentalized bits of the program using Selection Sort to divvy up the packages in a correct way in order to make it more efficient for customers in specific regions. I think that the pair if utilized together in the correct way can be incredibly strong for business and that it can pave the way for revolutionary utilizations and instances of different types of algorithms and data structures.