# 1.2 Documentation

## 1.2.1 Demonstration of Enhanced Features

Since most of the program will work exactly same as the documentation in Assignment 1, the documentation in this Assignment 2 will be only demonstrating the new features which is sorting features (only for manufacturing country, processor type and hard disk capacity) and searching by using binary search (only for manufacturing country, hard disk capacity and internal memory capacity) in displaying product records.

After running the program and select desired view records in display records function, the system will prompt the sorting selection for the user to choose the desired sorting criteria for the records. For example, it will allow user to sort as processor type or hard disk capacity for display according to manufacturing country; to sort as manufacturing country or hard disk capacity for display according to processor type; to sort as manufacturing country or processor type for display according to hard disk capacity; to sort as manufacturing country, processor type or hard disk capacity for display according to internal memory capacity, display according to quantity and display all as shown in *Figure 1.1*, *Figure 1.2*, *Figure 1.3*, *Figure 1.4*, *Figure 1.5*, *Figure 1.6*.

Text

Description automatically generated

*Figure 1.1* Sorting Selection – Display According to Manufacturing Country

Text

Description automatically generated

*Figure 1.2* Sorting Selection – Display According to Processor Type

Text

Description automatically generated

*Figure 1.3* Sorting Selection – Display According to Hard Disk Capacity

Text

Description automatically generated

*Figure 1.4* Sorting Selection – Display According to Internal Memory Capacity

Text

Description automatically generated

*Figure 1.5* Sorting Selection – Display According to Quantity

Text

Description automatically generated

*Figure 1.6* Sorting Selection – Display All

After user selected the desired sorting criteria, the system will display the user’s desired record according to their selected sorting criteria. For example, user wish to view all products in the system and wish to sort them according to the manufacturing country. The system displays all product records and sorts them by country name in alphabetically, which is America, Japan and Malaysia as shown in *Figure 1.7*.

Text

Description automatically generated

*Figure 1.7* Output – Display All (Sort by Manufacturing Country)

Another example, user wish to view Malaysia records in the system and wish to sort them according to the processor type. The system displays Malaysia product records and sorts them by processor type, which is Intel i5, Intel i7 and Intel i9 as shown in *Figure 1.8*.

Text

Description automatically generated

*Figure 1.8* Output – Display Malaysia Records (Sort by Processor Type)

Moreover, if user wish to view Intel i5 records in the system and wish to sort them according to the hard disk capacity, the system displays Intel i5 product records and sorts them by hard disk capacity, which is 320 GB, 500 GB and 1 TB as shown in *Figure 1.9*.

Text

Description automatically generated

*Figure 1.9* Output – Display Intel i5 Records (Sort by Hard Disk Capacity)

## 1.2.2 Validation of Enhanced Features

The validation will only be showing the validation on new features as well as mentioned in 1.2.1 Demonstration of Enhanced Features.

If the user input integer that out of sorting selection range, the system will prompt the user to key in a valid sorting selection as shown in *Figure 2.1*, *Figure 2.2*.

Text

Description automatically generated

*Figure 2.1* Sorting Selection Out of Range – 2 Option

Text

Description automatically generated

*Figure 2.2* Sorting Selection Out of Range – 3 Option

If the user key in invalid sorting selection that is not an integer, the system will prompt the user to key in an valid integer sorting selection as shown in *Figure 2.3*.

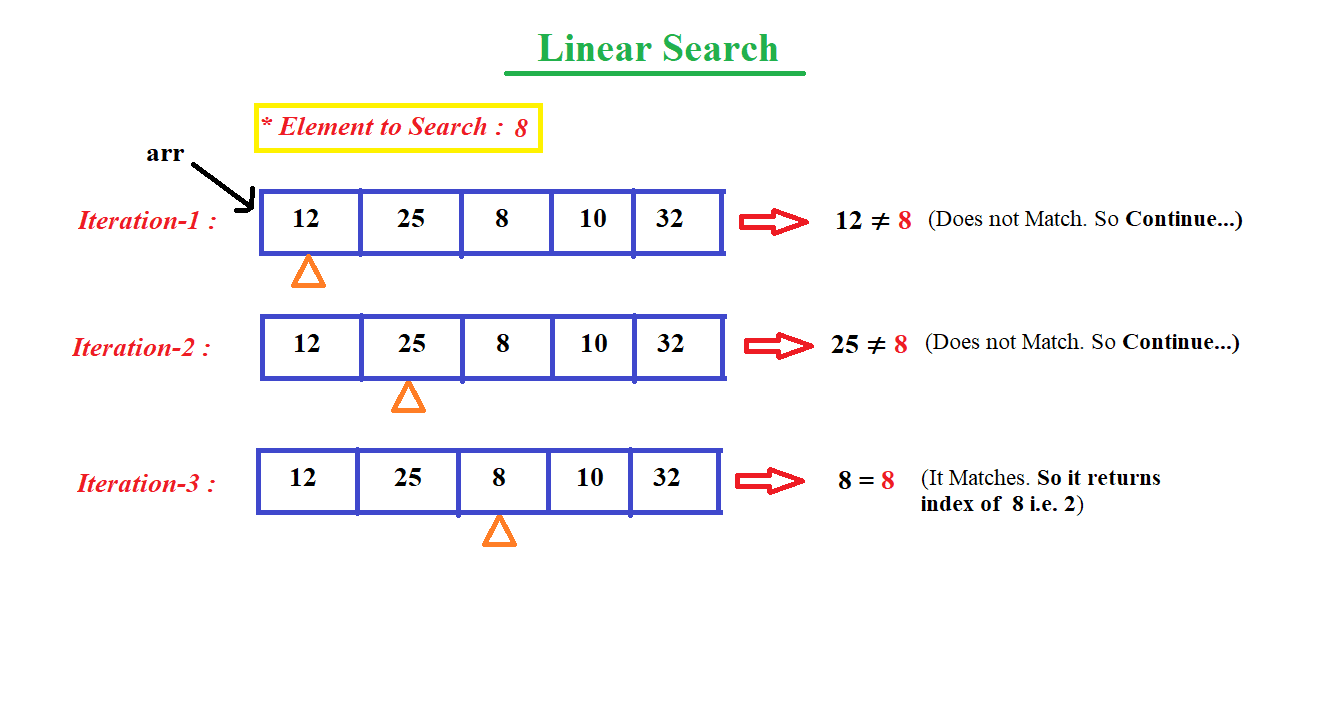
Text

Description automatically generated

*Figure 2.3* Sorting Selection Not Numeric

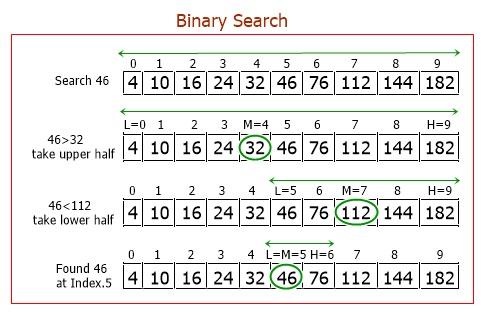
# Task B - Individual Report [Chan Seow Fen (0207368)]

Search algorithms are used to perform checking or retrieving an element from any data structure that contains that element. They look in the search space for a target (key) (Krishna, 2022). One of common searching algorithms is linear search. Linear search is also known as sequential search, it is a simple and straightforward search method that systematically examines each element of an array or list, beginning with the first element, until the desired element is located. Although it is the simplest and most straightforward search technique, it is not the most effective for huge datasets. The linear search algorithm works by start with the array's or list's first element. Next, up until the target element is located or the array's end is reached, compare each element with the target element. Finally, return the element's index if the desired element can be located. If not, return -1. For a better illustration, in real life, suppose Ali have a bookcase full of titles and Ali is trying to locate a certain title. To find the book Ali is looking for, start with the first book on the left and scan each one. Similar to the linear search searching algorithm, this procedure involves systematically scanning each element of an array or list until he locates the desired element. As an illustration, suppose Ali is looking for "Data Structures and Algorithms" on his bookshelf, which also contains the following titles: Foundation of HCI, Object Oriented Programming, Data Structures and Algorithms, System Analysis and Design, Computer Networks. If Ali wanted to find " Data Structures and Algorithms " using linear search, Ali would start from "Foundation of HCI," the first book on the left, and scan each book until he found it. To locate the target book in this example, scan the second book ("Object Oriented Programming") first. Although this procedure is straightforward and easy to understand, it gets less effective as the bookshelf fills up with more titles. Other real-life examples for linear search are finding a name on a phonebook, finding a person without knowing his or her appearance in a queue and finding a specific document in an unsorted document shelf. The time complexity of linear search is O(1), when the target element is discovered at the start of the array or list, which is the best case situation for linear search. In contrast, the worst-case situation involves finding the target element at the end of the array or list, which results in an O(n) time complexity. The target element is equally likely to be present at any place in the array or list in the average case scenario, resulting in an average time complexity of O(n) (S, 2021). The space complexity of linear search algorithm is O(1) which means it has a constant memory need (Ue Kiao, PhD, 2021). The algorithm just utilizes a few variables to hold the index and target element, and it does not need to store any other data structures. As a result, the magnitude of the input data has no bearing on how complex the linear search's space is.



*Figure 3.1* Linear Search Algorithm (Github.io, 2023)

Another searching algorithm is binary search. For sorted arrays, binary search is a more effective searching algorithm than linear search. The search period is split in half repeatedly until the target element is located or the search interval is empty. The binary search algorithm work by first, determine the array's middle index. Then, return the target element's index if the middle element is identical to it. It will repeat the method on the right half of the array if the center element is less than the target element and discard the left half. It will repeat the method on the left half of the array if the middle element is greater than the target element and discard the right half. It will continue doing this until the search interval is empty or the target element is located. To illustrate it, consider a library with a big collection of books where Amin is trying to find a specific book. All of the books in the library are ordered alphabetically by last name of the author. Amin is trying to locate the book as soon as he can, and he is aware of the author's last name. In this case, binary search might be used to find the book. He would begin by scanning the middle of the library's book selection. He would then search the right half of the collection if the book he is looking for appears after the book in the middle of the collection alphabetically. Amin would search the left half of the collection if the book came before the one in the middle. The search space would be divided in half again and again until the desired book was located. Because the books are grouped alphabetically, which is a sorted dataset, binary search is especially helpful in this situation. Amin may rapidly find the book he is seeking for without having to browse through every book in the library by using binary search. Overall, binary search can be applied to a variety of real-world situations where it is necessary to search a sorted dataset in order to locate a specific item, such as looking up a certain name in a phone book or a specific item in the inventory of an online retailer. The time complexity of binary search in the best case scenario will be O(1) if the target element being located in the center of the sorted array or list as binary search start from the middle of an array. However, in the worst situation, the target element is not present in the array or list, leading to an O(log n) time complexity as binary search need to search over the array until it make sure the element does not exist. Nevertheless, the target element is equally likely to be present in any place in the sorted array or list which is the average case scenario, leading to an average time complexity of O(log n) (Jana, 2022). The space complexity of binary search will be also O(1) as well. The reason is that binary search only requires a constant amount of memory to execute as to keep track of the variety of components that must be examined, only two variables are required. No other information is required. However, that is only applicable for common use of binary search which is iterative implementation of binary search, if it is recursive implementation of binary search, the space complexity will be O(log n) as in the worst situation, log n recursive calls will occur and that they will all be piled in memory. In fact, if I comparisons are required, I recursive calls will be stacked in memory, and based on our analysis of the average case time complexity, it can infer that the average memory will also be O(log n) (Ue Kiao, PhD, 2021).



*Figure 3.2* Binary Search Algorithm (Codescracker.com, 2023)