04/06/2023

PRG172

ASSIGNMENT 1

Lesedi Moholoeng
STUDENT NUMBER: 601390

Question 1: Practical

```
Sorting_Algorithm,
        Bubble_sort,
        Quick_sort,
        Merge_sort,

        Searching_Algorithm,
        Linear_search,
        Binary_search,
        Exit
    }
    static void Main()
    {

        Console.WriteLine("Please enter the size of the array you want to input:
");
        int size = int.Parse(Console.ReadLine());


        int[] Arr_Container = new int[size];

        for (int i = 0; i < size; i++)
        {
            Console.WriteLine($"Enter element {i + 1} of the array: ");
            Arr_Container[i] = int.Parse(Console.ReadLine());

Console.Clear();

        while (true)
        {

Console.WriteLine("**************************************************************
*******************************\n");

Console.WriteLine("******************************************************");
            Console.WriteLine("Please select an option in the Menu below: ");

Console.WriteLine("****************************************************** \r\n");

            Console.WriteLine("************************");
            Console.WriteLine(Menu.Sorting_Algorithm);
            Console.WriteLine("************************");

            Console.WriteLine("1. "+Menu.Bubble_sort);
            Console.WriteLine("2. "+Menu.Quick_sort);
            Console.WriteLine("3. "+Menu.Merge_sort);

            Console.WriteLine("\n************************");
            Console.WriteLine(Menu.Searching_Algorithm);
            Console.WriteLine("************************\r\n");
            Console.WriteLine("4. "+Menu.Linear_search);
            Console.WriteLine("5. "+Menu.Binary_search);
            Console.WriteLine("6. "+Menu.Exit + "\n");
            Console.Write("Option > : ");
            int choice = int.Parse(Console.ReadLine());

Console.WriteLine("\r\n**************************************************************
***********************************\n");



            Console.Clear();
```

```
            if (choice == 6)
            {
                Console.WriteLine("Thank you Good Bye!!! \r\n");
                break;

            }
            switch (choice)
            {
                case 1:
                    BubbleSort(Arr_Container);
                    Display_Array(Arr_Container);
                    break;
                case 2:
                    Quicksort(Arr_Container, 0, Arr_Container.Length - 1);
                    Display_Array(Arr_Container);
                    break;
                case 3:
                    MergeSort(Arr_Container, 0, Arr_Container.Length - 1);
                    Display_Array(Arr_Container);
                    break;
                case 4:
                    Console.Write("Please enter the value you want to search for
: ");
                    int value = int.Parse(Console.ReadLine());
                    int linearIndex = LinearSearch(Arr_Container, value);
                    if (linearIndex != -1)
                        Console.WriteLine("\nThe value requested is found at
index number: " + linearIndex);
                    else
                        Console.WriteLine("\nThe value requested is not found.");
                    break;
                case 5:

                    Console.Write("\nPlease enter the value you to search: ");
                    int binaryValue = int.Parse(Console.ReadLine());
                    int binaryIndex = BinarySearch(Arr_Container, binaryValue);
                    if (binaryIndex != -1)
                        Console.WriteLine("\nThe value requested is found at
index number: " + binaryIndex);
                    else
                        Console.WriteLine("\nThe value requested is found at
index number:");
                    break;
                default:
                    Console.WriteLine("The selected option is invalid. Please try
again.");
                    break;
            }
        }
    }

    // Bubble Sort
    static void BubbleSort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            { if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
```

```
                    arr[j + 1] = temp;
                }

// Quicksort
    static void Quicksort(int[] arr, int low, int high)
    {
        if (low < high)
        {
            int pivot = Partition(arr, low, high);
            Quicksort(arr, low, pivot - 1);
            Quicksort(arr, pivot + 1, high);
        }
    }

    static int Partition(int[] arr, int low, int high)
    {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++)
        {
            if (arr[j] < pivot)
            {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp2 = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp2;
        return i + 1;
    }

    // Merge Sort
    static void MergeSort(int[] arr, int left, int right)
    {
        if (left < right)
        {
            int middle = (left + right) / 2;
            MergeSort(arr, left, middle);
            MergeSort(arr, middle + 1, right)

   Merge(arr, left, middle, right);
        }
    }

    static void Merge(int[] arr, int left, int middle, int right)
    {
        int n1 = middle - left + 1;
        int n2 = right - middle;

        int[] leftArr = new int[n1];
        int[] rightArr = new int[n2];

        for (int i = 0; i < n1; i++)
            leftArr[i] = arr[left + i];
        for (int j = 0; j < n2; j++)

rightArr[j] = arr[middle + 1 + j];

        int k = left;
        int p = 0;
```

```
        int q = 0;

        while (p < n1 && q < n2)
        {
            if (leftArr[p] <= rightArr[q])
            {
                arr[k] = leftArr[p];

   p++;
            }
            else
            {
                arr[k] = rightArr[q];
                q++;
            }
            k++;
        }

        while (p < n1)
        {
            arr[k] = leftArr[p];
            p++;
            k++;
        }

else
            {
                arr[k] = rightArr[q];
                q++;
            }
            k++;
        }

        while (p < n1)
        {
            arr[k] = leftArr[p];
            p++;
            k++;
        }

        while (q < n2)
        {
            arr[k] = rightArr[q];
            q++;
            k++;
        }
    }

 // Linear Search
    static int LinearSearch(int[] arr, int value)
    {
        for (int i = 0; i < arr.Length; i++)
        {
            if (arr[i] == value)
                return i;
        }
        return -1;
    }

    // Binary Search
    static int BinarySearch(int[] arr, int value)
    {
        int left = 0;
```

```
        int right = arr.Length - 1;

        while (left <= right)
        {
            int middle = (left + right) / 2;

            if (arr[middle] == value)
                return middle;

            if (arr[middle] < value)
                left = middle + 1;
            else
                right = middle - 1;

}

        return -1;
    }

    // Print Array
    static void Display_Array(int[] arr)
    {
        Console.Write("Sorted Array: ");
        foreach (int element in arr)
            Console.Write(element + " ");
        Console.WriteLine();
    }
}
```

Part 2: Definitions

1. Bubble Sort:

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. It continues to iterate through the list until no more swaps are needed, indicating that the list is sorted. Bubble Sort has a time complexity of O(n^2).

2. Merge Sort:

In merge sort, the input array is divided into two halves, sorted recursively, and then merged. It divides the array until it reaches individual elements, then merges them back in a sorted order. The time complexity of merge sort is O(n log n).

3. Quick Sort:

Quick Sort is also a divide-and-conquer algorithm, in which a pivot element is selected, and the other elements are partitioned according to whether they are smaller or larger than the pivot. After that, the subarrays are sorted recursively. If the pivot is poorly chosen, Quick Sort can have a time complexity of O(n log n).

4. Linear Search:

In linear search, each element in the list is sequentially checked until a match is found or the list reaches the end. This means that it will take a longer time to search for an element in a large list than it will for a smaller list.

5. Binary Search:

Binary Search is a search algorithm that works on sorted lists by dividing the search interval in half at each step. It compares the target value with the middle element of the list and decides to continue the search in the lower or upper half. Binary Search has a time complete.

Citations and References

- Sorting Algorithm Animations at the Wayback Machine (archived 3 March 2015).
- Sequential and parallel sorting algorithms – Explanations and analyses of many sorting algorithms.
- Dictionary of Algorithms, Data Structures, and Problems – Dictionary of algorithms, techniques, common functions, and problems.
- Slightly Skeptical View on Sorting Algorithms – Discusses several classic algorithms and promotes alternatives to the quicksort algorithm.
- 15 Sorting Algorithms in 6 Minutes (Youtube) – Visualization and "audibilization" of 15 Sorting Algorithms in 6 Minutes.
- A036604 sequence in OEIS database titled "Sorting numbers: minimal number of comparisons needed to sort n elements" – Performed by Ford–Johnson algorithm.
- Sorting Algorithms Used on Famous Paintings (Youtube) – Visualization of Sorting Algorithms on Many Famous Paintings.
- A Comparison of Sorting Algorithms – Runs a series of tests of 9 of the main sorting algorithms using Python timeit and Google Colab.