Overview

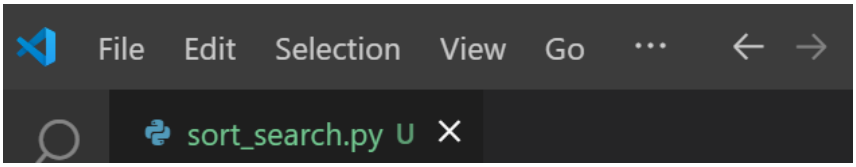
This lab practical is designed to focus on the fundamental concepts of sorting and searching elements within 2D arrays using Python. Through this walkthrough of the bubble sort algorithm, students will learn how to efficiently sort data in a two-dimensional structure. Following the sorting process, it will guide students through the development of a linear search function to locate specific elements within the sorted array.

Pre-requisites:

1. Github account
2. Git
3. VSCode
4. Python
5. Basic understanding of Python syntax.
6. Basic understanding of functions, sorting algorithm and searching algorithms
7. Familiarity with git commands

Instructions:

1. Open the working directory in the VSCode and create a file called sort_search.py:



2. Then write the following function for the bubble sort algorithm:

```python
def bubble_sort_2d(arr):
    n = len(arr)    # Number of rows in the 2D array
    m = len(arr[0]) # Number of columns in the 2D array; assumes all rows have equal length
    total_elements = n * m  # Total number of elements in the 2D array

    for i in range(total_elements - 1):
        # Outer loop: goes through all elements in the 2D array

        for j in range(total_elements - 1 - i):
            # Inner loop: goes through the elements, reducing the comparison range each time

            # Calculate current position in 2D terms
            row1 = j // m
            col1 = j % m

            # Calculate next position (right next to current position)
            row2 = (j + 1) // m
            col2 = (j + 1) % m

            # Compare and possibly swap elements
            if arr[row1][col1] > arr[row2][col2]:
                # If the current element is greater than the next, swap them
                arr[row1][col1], arr[row2][col2] = arr[row2][col2], arr[row1][col1]

```

This function sorts a 2D array arr in ascending order as if it was a 1D list, but maintains the 2D structure. It iterates through the array elements for the purpose of comparisons and swaps, using row and column calculations to map the 1D index back to 2D coordinates.

3. Now, program the following function for the linear search algorithm:

```python
def search_element(arr, element):
    found = False  # Initialize a flag to track if the element is found
    for i in range(len(arr)):
        for j in range(len(arr[i])):
            if arr[i][j] == element:
                print(f"Element found at position: row = {i}, column = {j}")
                found = True
                return  # Exit the function after finding the element
    if not found:
        print("Element not found in the given array.")

```

This function searches for a specific element in the 2D array arr. It iterates through each element of the array, using nested loops to go row by row and column by column. If the element is found, it prints the position (row and column) and exits the function immediately to avoid unnecessary searches. If the element is not found after checking the entire array, it prints a message indicating that the element is not in the array.

4. Then, write the following code:

```
# Example usage:
arr = [
    [9, 2, 3],
    [4, 5, 6],
    [7, 8, 1]
]

print(arr)
bubble_sort_2d(arr)
print(arr)

# Searching for an element
search = int(input("Enter the element to search: "))
search_element(arr, search)
```

Initialize a sample array to arr and then print it out. Call the bubble_sort_2d() function by passing the array called arr as an argument to the to sort the elements inside the array in ascending order. Print out the sorted array to visualize it. Then, ask the user to enter the element that they want to search for inside the sorted array. Lastly call the search_element() function by passing the sorted array arr and user input search as its arguments.

Exercise:
Create a Python program that sorts an array of integers using the quick sort algorithm and then searches for specific elements to replace them with other elements.

Instructions:

1. Prompt the user to input an array of integers.
2. Store the elements in an array in a Python list.
3. Implement the quick sort algorithm to sort the input array in ascending order.
4. Allow the user to specify a target element to search for in the sorted array.
5. If the target element is found, prompt the user to input a replacement element. Replace all occurrences of the target element with the replacement element in the array.
6. Display the modified array after replacing the elements.

Expected Output:

```
Array Sorting and Element Replacement

Enter the array elements separated by spaces: 0 12 10 3 4

Array: [0, 12, 10, 3, 4]
Sorted Array using Quick Sort: [0, 3, 4, 10, 12]

Enter the element to search for: 0
Enter the replacement element: 1

Modified Array after replacing 0 with 1: [1, 3, 4, 10, 12]
```

```
Array Sorting and Element Replacement

Enter the array elements separated by spaces: 0 12 3 4 10

Array: [0, 12, 3, 4, 10]
Sorted Array using Quick Sort: [0, 3, 4, 10, 12]

Enter the element to search for: 1
Enter the replacement element: 10
Element not found in the given array.
```