LAB 1

SEMISTER 3rd

fall-2024

SUBJECT

Database Structures

COURSE CODE

CC210

PROGRAMME

BS (4 year)

ABBOTTABAD UNIVERSITY OF SCIENCE AND TECHNOLOGY ISLAMABAD

SUBMITTED TO

Mr. Jamal Abdul Ahad

SUBMITTED BY

Haroon Imran

STUDENT ROLL NO

14723

SUBMISSION DATE

15 October 2024

Exercise 1

Task 1:

Sort an array of 10 random integers using the merge sort function and observe the output.

```
def merge_sort(arr):
    if len(arr) > 1:
        # Find the middle point to divide the array into two halves
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        # Recursively sort the two halves
        merge_sort(left_half)
        merge_sort(right_half)

        # Merge the sorted halves
        merge(arr, left_half, right_half)
def merge(arr, left_half, right_half):
    i = j = k = 0

    # Copy data to temporary arrays left_half[] and right_half[]
    while i < len(left_half) and j < len(right_half):
        if left_half[i] < right_half[j]:
            arr[k] = left_half[i]
            i += 1
        else:
            arr[k] = right_half[j]
            j += 1
        k += 1
```

```
    # Check for any remaining elements
    while i < len(left_half):
        arr[k] = left_half[i]
        i += 1
        k += 1

    while j < len(right_half):
        arr[k] = right_half[j]
        j += 1
        k += 1

# Driver code to test the merge_sort function
if __name__ == "__main__":
    arr = [12, 11, 13, 5, 6, 7,5,6,8,3]
    print("Original array:", arr)
    merge_sort(arr)
    print("Sorted array:", arr)
```

Output:

```
Original array: [12, 11, 13, 5, 6, 7, 5, 6, 8, 3]
Sorted array: [3, 5, 5, 6, 6, 7, 8, 11, 12, 13]
```

Task 2:

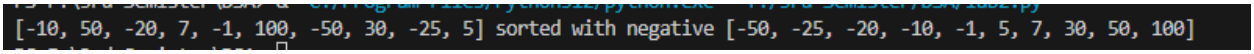
Modify the input array to include negative numbers and test the function.

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]
```

```
    # Recursive calls for each half
    merge_sort(left_half)
    merge_sort(right_half)

    # Merging the two halves
    i = j = k = 0

    while i < len(left_half) and j < len(right_half):
        if left_half[i] < right_half[j]:
            arr[k] = left_half[i]
            i += 1
        else:
            arr[k] = right_half[j]
            j += 1
        k += 1

    while i < len(left_half):
        arr[k] = left_half[i]
        i += 1
        k += 1

    while j < len(right_half):
        arr[k] = right_half[j]
        j += 1
        k += 1

  return arr

# Task 2: Modify the array to include negative numbers and test the merge_sort function.
```

```
# # # Array with negative numbers

if __name__ == "__main__":

    test_array_with_negatives = [-10, 50, -20, 7, -1, 100, -50, 30, -25, 5]

# Sorting the array with merge_sort

    sorted_array_with_negatives = merge_sort(test_array_with_negatives[:])  # Copying to avoid
modifying the original


    print(test_array_with_negatives,"sorted with negative", sorted_array_with_negatives)
```

Output:



Task 3:

Measure the time complexity of the merge_sort function using the time module for an array of size n where n = 100, 1000, 10000.

```
import random

import time


# Define the merge_sort function again (since the environment was reset)

def merge_sort(arr):

    if len(arr) > 1:

        mid = len(arr) // 2

        left_half = arr[:mid]

        right_half = arr[mid:]


        # Recursive calls for each half

        merge_sort(left_half)

        merge_sort(right_half)
```

```
        # Merging the two halves
        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
def measure_time(n):
    arr=[random.randint(0,10000) for _ in range(n)]
    start_time=time.time()
    merge_sort(arr)
    end_time=time.time()
    return end_time - start_time

sizes=[100,1000,10000]
results={}
```

```
for size in sizes:
    elapsed_time=measure_time(size)
    results[size]=elapsed_time
    print(f"Time to sort array of size {size}: {elapsed_time:.6f} seconds")

print("\nTime complexity results:")
for size, elapsed in results.items():
    print(f"Size: {size}, Time taken: {elapsed:.6f} seconds")
```

Output:

```
Time to sort array of size 100: 0.001354 seconds
Time to sort array of size 1000: 0.004293 seconds
Time to sort array of size 10000: 0.075478 seconds

Time complexity results:
Size: 100, Time taken: 0.001354 seconds
Size: 1000, Time taken: 0.004293 seconds
Size: 10000, Time taken: 0.075478 seconds
```

Exercise 2

Task 1:

Sort an array of 10 random integers using the insertion_sort function and observe the output.

Task 2:

Modify the input array to include negative numbers and test the function.

Task 3:

Measure the time complexity of the insertion_sort function using the time module for different array sizes (e.g., 100, 1000, 10000).

```
import random
import time

# Define the insertion sort function
```

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr


# Task 1: Sorting a random array of 10 integers
test_array_1 = random.sample(range(1, 101), 10)  # Array with random integers between 1 and 100
sorted_array_1 = insertion_sort(test_array_1[:])  # Copy to prevent in-place sorting during the test
print("Task 1 - Original array:", test_array_1)
print("Task 1 - Sorted array:", sorted_array_1)


# Task 2: Modify the array to include negative numbers and test
test_array_2 = random.sample(range(-50, 51), 10)  # Array with random integers between -50 and 50
sorted_array_2 = insertion_sort(test_array_2[:])
print("\nTask 2 - Original array (with negatives):", test_array_2)
print("Task 2 - Sorted array:", sorted_array_2)


# Task 3: Measure time complexity with different array sizes
def measure_time_complexity(array_size):
    test_array = [random.randint(-1000, 1000) for _ in range(array_size)]  # Random integers between -1000 and 1000
```

```
    start_time = time.time()
    insertion_sort(test_array)
    end_time = time.time()
    return end_time - start_time

# Measure for 100, 1000, and 10000 elements
time_100 = measure_time_complexity(100)
time_1000 = measure_time_complexity(1000)
time_10000 = measure_time_complexity(10000)

print("\nTask 3 - Time complexity results:")
print("Time for 100 elements: {:.6f} seconds".format(time_100))
print("Time for 1000 elements: {:.6f} seconds".format(time_1000))
print("Time for 10000 elements: {:.6f} seconds".format(time_10000))
```

Output: