**LAB 1**

**SEMISTER 3rd**

fall-2024

**SUBJECT**

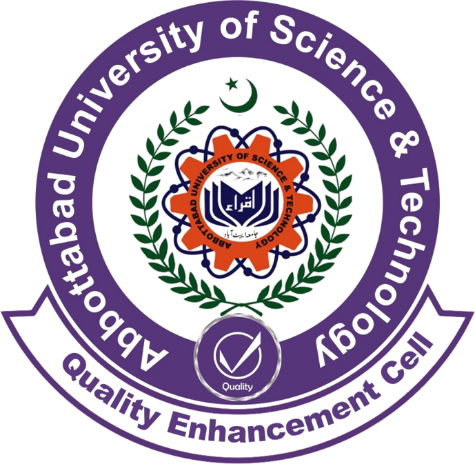
Database Structures

**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:**



**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

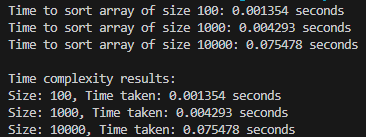
    print(f"Time to sort array of size {size}: {elapsed\_time:.6f} seconds")

print("\nTime complexity results:")

for size, elapsed in results.items():

    print(f"Size: {size}, Time taken: {elapsed:.6f} seconds")

**Output:**



**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:**

