Experiment no 12

Merge and Quick Sort

OBJECTIVES:

- To learn about the concept of Merge and Quick Sort.
- To learn different ways for implementing a Merge and Quick Sort.

1. MERGE SORT:

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

The provided Python code implements the Merge Sort algorithm, a divide-and-conquer sorting technique. It breaks down an array into smaller subarrays, sorts them individually, and then merges them back together to create a sorted array. The code includes two main functions: merge, responsible for merging two subarrays, and mergeSort, which recursively divides and sorts the array. The merge function combines two sorted subarrays into a single sorted array. The mergeSort function recursively splits the array in half until each subarray has a single element, then merges them to achieve the final sorted result. The example sorts an array using Merge Sort and prints both the initial and sorted arrays.

```
# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]


def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0     # Initial index of first subarray
    j = 0     # Initial index of second subarray
    k = l     # Initial index of merged subarray
```

```
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Copy the remaining elements of L[], if there
    # are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there
    # are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
```

```python
def mergeSort(arr, l, r):
    if l < r:

        # Same as (l+r)//2, but avoids overflow for
        # large l and h
        m = l+(r-l)//2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)


# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is")
for i in range(n):
    print("%d" % arr[i],end=" ")

mergeSort(arr, 0, n-1)
print("\n\nSorted array is")
for i in range(n):
    print("%d" % arr[i],end=" ")

# This code is contributed by Mohit Kumra
```

Figure 1: Merge Sort Example

2. QUICK SORT:

Just unlikely merge Sort, QuickSort is a **divide and conquer algorithm**. It picks an element as a pivot and partitions the given array around the picked pivot.
There are many different versions of quickSort that pick pivot in different ways.

1. Always pick the first element as a pivot
2. Always pick the last element as a pivot
3. Pick a random element as a pivot
4. Pick median as a pivot

Here we will be picking the last element as a pivot. The key process in quickSort is partition(). Target of partitions is, given an array and an element ‘x’ of array as a pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

```python
# Function to find the partition position
def partition(array, low, high):

    # choose the rightmost element as pivot
    pivot = array[high]

    # pointer for greater element
    i = low - 1

    # traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:

            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with the greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])

    # Return the position from where partition is done
    return i + 1
```

```python
def quickSort(array, low, high):
    if low < high:

        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quickSort(array, pi + 1, high)


data = [1, 7, 4, 1, 10, 9, -2]
print("Unsorted Array")
print(data)

size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')
print(data)
```

Figure 2: Quick Sort Example

3. ACTIVITIES:

A1) Write a code for Merge Sort.

A2) Write a code for Quick Sort.

A3) Write a code in which merge and quick sort should be implemented.

IMPORTANT: All the activities` code should be attached to the manual before summary section.

Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature:_____ **Date:**_____

LABORATORY SKILLS ASSESSMENT (Psychomotor)

Total Marks:100

Criteria (Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< 90%	Level 4 90%≤ S ≤100%	Score (S)
Procedural Awareness (30)	Selects inappropriate skills and/or strategies Required by the task.	Selects and applies appropriate skills and/or strategies required by the task with major errors.	Selects and applies the appropriate strategies and/or skills specific to the task without significant errors.	Selects and applies appropriate strategies and/or skills specific to the task without any error.	
Practical Implementation (30)	Makes major critical errors in applying procedural knowledge related to python Merge and Quick Sort Algorithm.	Makes numerous critical errors in applying procedural knowledge related to python Merge and Quick Sort Algorithm.	Makes some non-critical errors in applying procedural knowledge related to python Merge and Quick Sort Algorithm.	Applies the procedural knowledge in optimized ways related to python Lists, Merge and Quick Sort Algorithm.	
Use of Tool/Equipment (30)	Uses tools, equipment and materials with limited competence.	Uses tools, equipment and materials with some competence.	Uses tools, equipment and materials with considerable competence.	Uses tools, equipment and materials with a high degree of competence.	
Safety (10)	Requires constant reminders to follow safety procedures.	Requires some reminders to follow safety procedures.	Follows safety procedures with only minimal reminders.	Routinely follows safety procedures.	
				Marks Obtained	

Instructor Name:_____ **Sign:** _____

LABORATORY SKILLS ASSESSMENT (Affective)

Total Marks: 40

Criteria (Max. Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < 90%	Level 4 90% ≤ S ≤ 100%	Score
Introduction (5)	Very little background information provided or information is incorrect	Introduction is brief with some minor mistakes	Introduction is nearly complete, missing some minor points	Introduction complete and well-written; provides all necessary background principles for the experiment	
Procedure (5)	Many stages of the procedure are not entered on the lab report.	Many stages of the procedure are entered on the lab report.	The procedure could be more efficiently designed but most stages of the procedure are entered on the lab report.	The procedure is well designed and all stages of the procedure are entered on the lab report.	
Data Record (10)	Data is brief and missing significant pieces of information.	Data provides some significant information and has few critical mistakes.	Data is almost complete but has some minor mistakes.	Data is complete and relevant. Tables with units are provided. Graphs are labeled. All questions are answered correctly.	
Data Analysis (10)	Data is presented in very unclear manner.	Data is presented in ways that are not clear enough.	Data is presented in ways that can be understood and interpreted.	Data is presented in ways that best facilitate understanding and interpretation.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Cognitive)

Total Marks: 10

(If any) **Marks Obtained**	

Instructor's Signature:_____ **Date:**_____