Part A

Aim:
1. Design and analysis algorithms for merging two sorted list.
2. Design and analysis algorithms for merge sort.

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:
Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.

Procedure:
1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Query statement
1	Design and analysis algorithm for merging two sorted list.
2	Applying divide and conquer method to sort n elements using merge sort. a) Elements can be read from a file or can be generated using random number generator. b) Determine time required to sort the elements. c) Repeat experiment for different values of n and plot the graph of the time taken versus n.
3	For both the algorithm, give case-wise complexity

Instructions:
1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

1) **CODE:**
a) **Algorithm:**

Input: two sorted lists
Output: merged list of 2 input lists
Algorithm:

Merging 2 lists:

```
def mergesort(arr,left,right):
   i=0
   j=0
   k=0
```

```
arr=[0]*(len(left)+len(right))
while i<len(left) and j<len(right):
    if left[i]<=right[j]:
        arr[k]=left[i]
        k+=1
        i+=1
    else:
        arr[k]=right[j]
        k+=1
        j+=1
while(i<len(left)):
    arr[k]=left[i]
    k+=1
    i+=1
while(j<len(right)):
    arr[k]=right[j]
    k+=1
    j+=1
return arr
```

Program:

```
def mergesort(arr,left,right):
    i=0
    j=0
    k=0
    arr=[0]*(len(left)+len(right))
    while i<len(left) and j<len(right):
        if left[i]<=right[j]:
            arr[k]=left[i]
            k+=1
            i+=1

        else:
            arr[k]=right[j]
            k+=1
            j+=1

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

    while(j<len(right)):
        arr[k]=right[j]
        k+=1
        j+=1
    return arr
```

```
arr=[]
left=list(map(int,input("Input 1st sorted list").split()))
right=list(map(int,input("Input 1st sorted list").split()))
arr=mergesort(arr,left,right)
print("Sorted list :",arr)
```

Input and output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week3> python .\merge2lists.py
Input 1st sorted list 1 4 6 9 12 19 23 28
Input 1st sorted list 5 8 11 16 22 27 35 38 88
Sorted list : [1, 4, 5, 6, 8, 9, 11, 12, 16, 19, 22, 23, 27, 28, 35, 38, 88]
PS E:\books and pdfs\sem4 pdfs\DAA lab\week3>
```

Case-wise complexity:

Best case: O(m+n)
Average case: O(m+n)
Worst case: O(m+n)
where m,n are lengths of each list

2)

a) **Algorithm:**

Input: n list of random elements
Output: Sorted lists of elements with plot of time complexity of merge sort
Algorithm:

Sorting:

```
def mergesort(arr,left,right):
    i=0
    j=0
    k=0
    global com
    while i<len(left) and j<len(right):
        com+=1
        if left[i]<=right[j]:
            arr[k]=left[i]
            k+=1
            i+=1

        else:
            arr[k]=right[j]
            k+=1
            j+=1
```

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

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

Merge function:

```
def merge(arr):
    if len(arr)>1:
        mid=math.ceil(len(arr)//2)
        left=arr[:mid]
        right=arr[mid:]
        merge(left)
        merge(right)
        mergesort(arr,left,right)
```

b) Program:

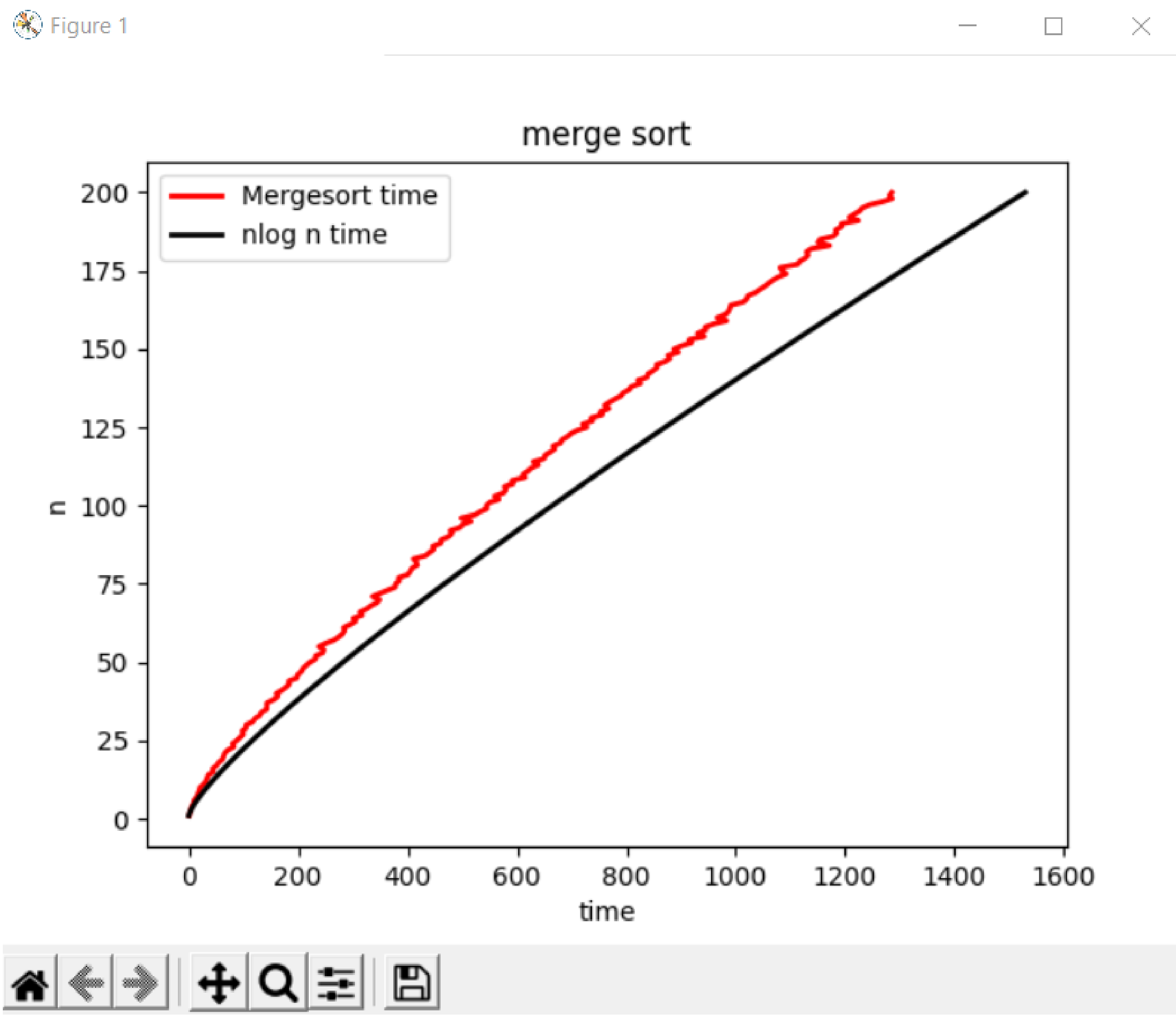
```
import math
import random
import time
import matplotlib.pyplot as plt

def mergesort(arr,left,right):
    i=0
    j=0
    k=0
    global com
    while i<len(left) and j<len(right):
        com+=1
        if left[i]<=right[j]:
            arr[k]=left[i]
            k+=1
            i+=1

        else:
            arr[k]=right[j]
            k+=1
```

```
            j+=1

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

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


def merge(arr):
    if len(arr)>1:
        mid=math.ceil(len(arr)//2)
        left=arr[:mid]
        right=arr[mid:]
        merge(left)
        merge(right)
        mergesort(arr,left,right)

complist=[]
timeList=[]
n=int(input("number of lists: "))
for i in range(1,n+1):
    arr1=[]
    for j in range(i):
        arr1.append(random.randint(1,100))

    com=1
    start = time.time()
    merge(arr1)
    total=time.time()-start
    timeList.append(total)
    complist.append(com)

print("Average time taken to sort n lists using merge
sort",sum(timeList)/len(timeList))


actual=[*range(1,n+1)]
actual1=[]
for i in actual:
    actual1.append(i*math.log(i,2))

plt.plot(complist,actual,color='red', linewidth=2,label='Mergesort time')
```

```python
plt.plot(actual1,actual,color='black', linewidth=2,label='nlog n time')
plt.title('merge sort')
plt.xlabel('time')
plt.ylabel('n')
plt.legend()
plt.show()
```

Input & Output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week3> python .\merge_sort.py
number of lists: 200
time list is  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0010132789611816406, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0009875297546386719, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0010116100311279297, 0.0, 0.0, 0.0, 0.0, 0.0, 0.00099945068359375, 0.0, 0.0, 0.0, 0.0, 0.0010013580322265625, 0.0, 0.0, 0
.0, 0.0, 0.0010001659393310547, 0.0, 0.0, 0.0, 0.0, 0.0009987354278564453, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0009996891021728516, 0.0, 0.0, 0.0, 0.0010001659393310547, 0
.0, 0.0, 0.0, 0.0009882450103759766, 0.0, 0.0010004043579101562, 0.0010004043579101562, 0.0009999275207519531, 0.0, 0.0009996891021728516, 0.0009999275207519531, 0.0, 0.00100088
11950683594, 0.0, 0.0, 0.0010113716125488281, 0.0, 0.0, 0.0011088848114013672, 0.0, 0.0008904933929443359, 0.0, 0.0009899139404296875, 0.0, 0.0, 0.0, 0.0, 0.0010006427764892578,
 0.0, 0.0, 0.0010006427764892578, 0.0, 0.0, 0.0, 0.0, 0.0009999275207519531, 0.0, 0.0, 0.0, 0.0010006427764892578, 0.0, 0.0010087490081787 11, 0.0, 0.0010004043579101562, 0.0, 0.
0, 0.0009884834289550781, 0.0, 0.0010166168212890625, 0.0, 0.0009920597076416016, 0.0, 0.0010008811950683594, 0.00099945068359375, 0.0, 0.0010015964508056 64, 0.0, 0.001000165939
3310547, 0.0, 0.0010001659393310547, 0.0, 0.0009999275207519531, 0.0, 0.0010023117065429688, 0.0, 0.0009870529174804688, 0.0, 0.0010123252868652344, 0.0, 0.0009875297546386719,
0.0010130405426025 39, 0.0, 0.00099945068359375, 0.0, 0.0010004043579101562, 0.0009999275207519531, 0.0, 0.0010001659393310547, 0.0, 0.0009996891021728516, 0.0, 0.0, 0.0009880065
91796875, 0.0, 0.0010130405426025 39, 0.0009989738464355469, 0.0, 0.0010004043579101562, 0.0009999275207519531, 0.0, 0.0010001659393310547, 0.0, 0.0, 0.0010001659393310547, 0.0,
0.0, 0.0009999275207519531, 0.0, 0.0, 0.0010120868682861328, 0.0, 0.0, 0.0009999275207519531, 0.0009999275207519531, 0.0, 0.0010004043579101562, 0.0009999275207519531, 0.0009891
986846923828, 0.0009992122650146484, 0.0009999275207519531, 0.0009996891021728516, 0.0010006427764892578, 0.0010006427764892578, 0.00099945068359375, 0.0009999275207519531, 0.00
09999275207519531, 0.0010008811950683594, 0.0009992122650146484, 0.0010008811950683594, 0.0, 0.0, 0.0009999275207519531, 0.0009999275207519531, 0.0, 0.0010006427764892578, 0.0010001659393310547, 0.00100970
26824951172, 0.0009992122650146484, 0.0009999275207519531, 0.0010006427764892578, 0.00099945068359375]
Average time taken to sort n lists using merge sort 0.0003901326656341553
```



Case-wise complexity for merge sort:
Best case : O(nlog n)
Average case : O(nlog n)
Worst case : O(nlog n)
where n comes from merging procedure while log n comes from dividing the list into 2 halves.

Observation & Learning:
I have observed that:
i)Case wise time complexities for merge sort are same i.e O(nlog n)
ii)comparatively slower than other sort algorithms for smaller tasks
iii)the runtime increases approximately linearly with the number of elements

Conclusion:
I have designed, analyzed and implemented the algorithms of merging 2 lists and merge sort and plotted the running time of the merge sort algorithm.

Questions:
1. Is mergesort stable sorting?
2. Is mergesort internal sorting?
3. Is mergesort in-place sorting?

Answers:
1. **Yes,**
   Merge sort is stable as elements of array maintain their original positions with respect to each other
2. **No,**
   Merge sort is not internal sorting since there is a need for external memory for sorting list using merge sort,i.e, sorting process can't be adjusted in main memory
3. **No**
   Merge sort isn't in-place sort as it requires additional memory to store auxiliary arrays.