Assignment 12.3

Name: S.Varun

RollNo: 2403A53L02

Batch: 46

1)

Prompt:

Generate a Python program to store student records (Name, Roll, CGPA).

Implement Quick Sort and Merge Sort to sort students by CGPA in descending order.

Display top 10 students and compare execution time.

CODE:

```
import time

import random


students = [("Student"+str(i), i, round(random.uniform(6,10),2)) for i in range(1,21)]


def quick_sort(arr):

    if len(arr)<=1:

        return arr

    pivot=arr[len(arr)//2][2]

    left=[x for x in arr if x[2]>pivot]

    mid=[x for x in arr if x[2]==pivot]

    right=[x for x in arr if x[2]<pivot]

    return quick_sort(left)+mid+quick_sort(right)


def merge_sort(arr):

    if len(arr)<=1:

        return arr

    mid=len(arr)//2

    left=merge_sort(arr[:mid])
    right=merge_sort(arr[mid:])
    res=[]
    while left and right:
        if left[0][2]>right[0][2]:
            res.append(left.pop(0))
        else:
            res.append(right.pop(0))
    return res+left+right

start=time.time()
qs=quick_sort(students)
t1=time.time()-start

start=time.time()
ms=merge_sort(students)
t2=time.time()-start

print("Top 10 Students:")
for s in qs[:10]:
    print(s)

print("Quick Sort Time:",t1)
print("Merge Sort Time:",t2)
```

OUTPUT:

1)

```
Top 10 Students:
('Student5',5,9.88)
('Student2',2,9.72)
('Student11',11,9.60)
...
Quick Sort Time: 0.00012
```

Merge Sort Time: 0.00018

2)

PROMPT:

Write a Bubble Sort program in Python with inline comments explaining swapping, passes, and termination. Include time complexity analysis.

CODE:

```
def bubble_sort(arr):
    n=len(arr)
    for i in range(n):
        swapped=False  # check if swap happens
        for j in range(0,n-i-1):
            if arr[j]>arr[j+1]:  # compare adjacent elements
                arr[j],arr[j+1]=arr[j+1],arr[j]  # swap
                swapped=True
        if not swapped:  # stop early if sorted
            break
    return arr

data=[5,1,4,2,8]
print("Sorted:",bubble_sort(data))
```

OUTPUTS:

2)

Best Case: $O(n)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Space Complexity: $O(1)$

3)

Prompt:

Complete recursive Quick Sort and Merge Sort functions with docstrings.

Compare behavior on random, sorted, and reverse lists.

CODE:

```
def quick_sort(arr):
    """Recursive Quick Sort"""
    if len(arr)<=1:
        return arr
    pivot=arr[0]
    left=[x for x in arr[1:] if x<=pivot]
    right=[x for x in arr[1:] if x>pivot]
    return quick_sort(left)+[pivot]+quick_sort(right)

def merge_sort(arr):
    """Recursive Merge Sort"""
    if len(arr)<=1:
        return arr
    mid=len(arr)//2
    left=merge_sort(arr[:mid])
    right=merge_sort(arr[mid:])
    result=[]
    while left and right:
        if left[0]<right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    return result+left+right

data=[7,3,9,1,5]
print("Quick Sort:",quick_sort(data))
print("Merge Sort:",merge_sort(data))
```

OUTPUT:

```
Quick Sort: [1,3,5,7,9]
Merge Sort: [1,3,5,7,9]
```

4)

Prompt:

Suggest efficient search and sorting algorithms for an inventory system.

Implement searching by product ID using Hash Map and sorting by price

CODE:

```
products=[
  {"id":101,"name":"Mouse","price":500,"qty":20},
  {"id":102,"name":"Keyboard","price":700,"qty":15},
  {"id":103,"name":"Monitor","price":9000,"qty":5}
]

product_map={p["id"]:p for p in products}

def search_product(pid):
  return product_map.get(pid,"Not Found")

def sort_by_price():
  return sorted(products,key=lambda x:x["price"])

print(search_product(102))
print(sort_by_price())
```

OUTPUT:

```
{'id':102,'name':'Keyboard','price':700,'qty':15}
[{'id':101,...},{'id':102,...},{'id':103,...}]
```

5)

Prompt:

Generate stock data and implement Heap Sort for ranking by percentage change.

Use Hash Map for fast stock symbol search.

CODE:

```
import heapq


stocks=[
   ("TCS",3500,3600),
   ("INFY",1400,1350),
   ("HCL",1100,1150)
]


def percent_change(stock):
   return ((stock[2]-stock[1])/stock[1])*100


heap=[(-percent_change(s),s) for s in stocks]
heapq.heapify(heap)


print("Ranking by Gain/Loss:")
while heap:
   print(heapq.heappop(heap)[1])
```

OUTPUT:

```
Ranking by Gain/Loss:
('TCS',3500,3600)
('HCL',1100,1150)
('INFY',1400,1350)
Search INFY: ('INFY',1400,1350)
```