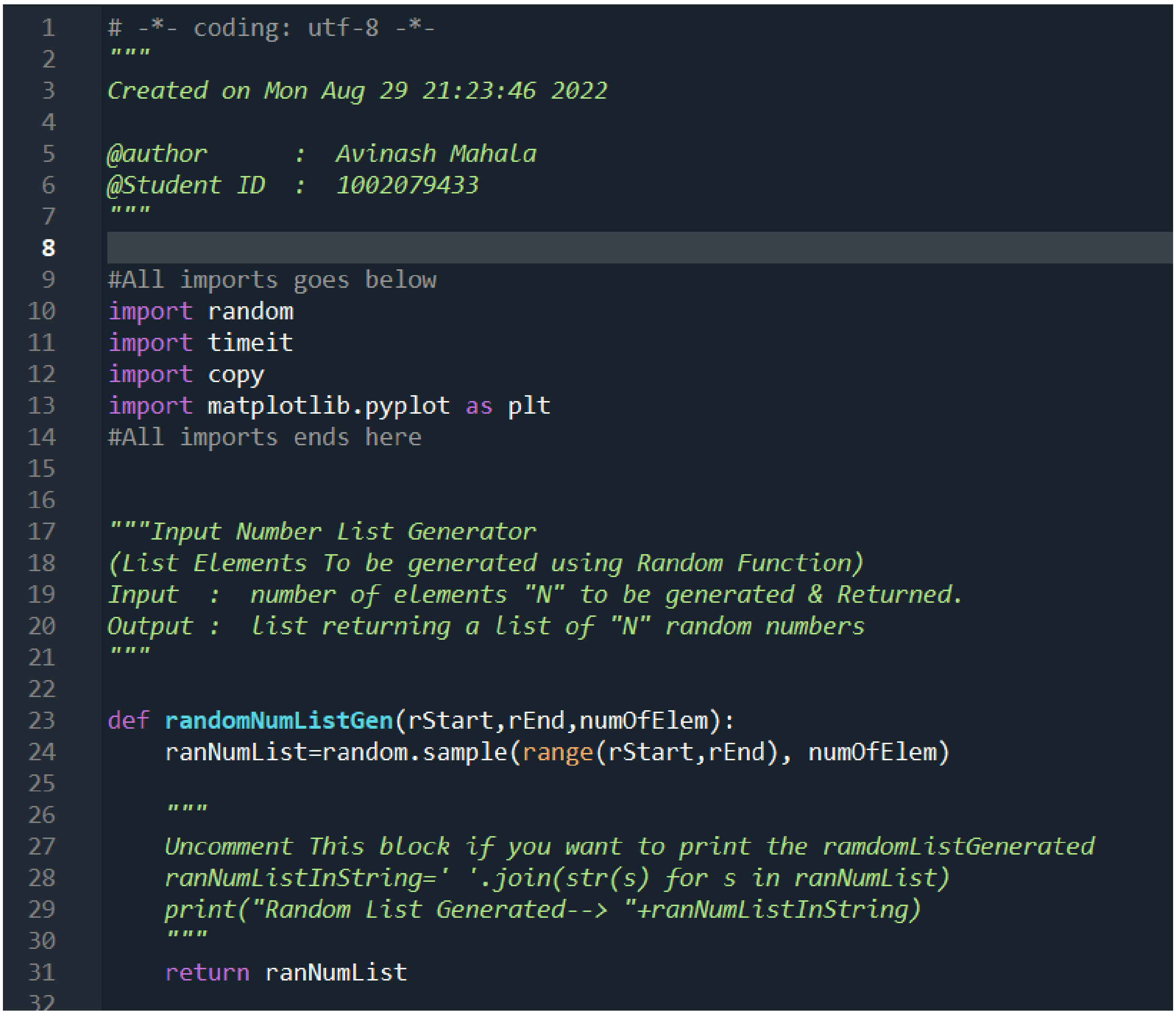
Q1) [60 points] Implement the insertion sort and merge sort algorithms with any programming language you choose and run them with the same input number list. Generate the list elements with a random function and increase the list size incrementally until you find the execution time of your merge sort program is consistently shorter. Plot the two curves in a figure (execution time vs input list size) about the two programs. Using the example to discuss why the asymptotic analysis is meaningful. Attach program codes in your submission.

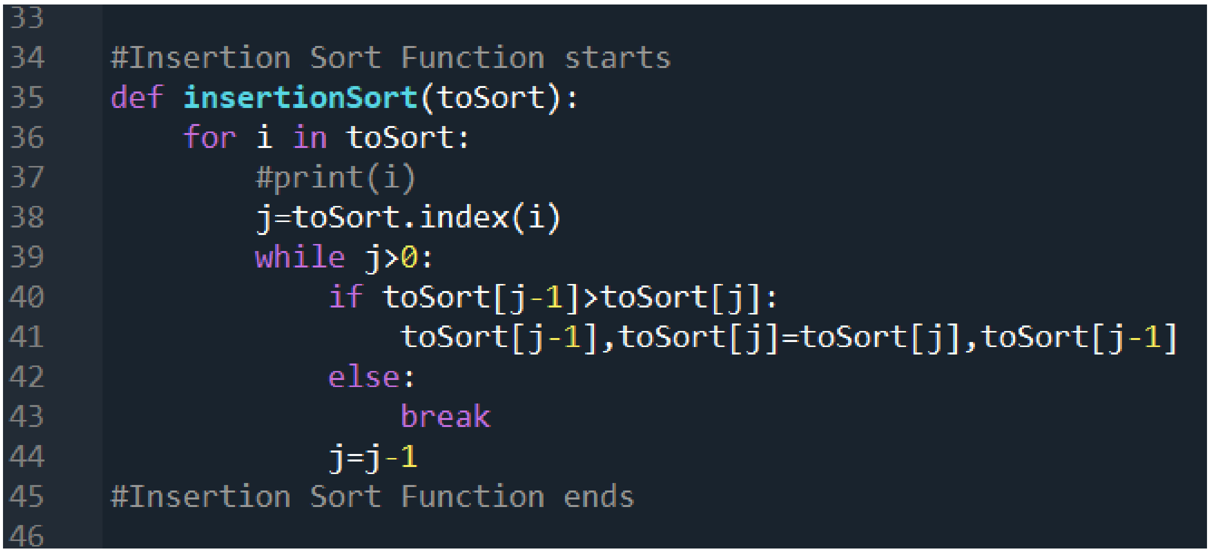
<u>Answer:-</u>

* Spyder version: 5.1.5 None
* Python version: 3.9.12 64-bit
* Qt version: 5.9.7
* PyQt5 version: 5.9.2
* Operating System: Windows 10

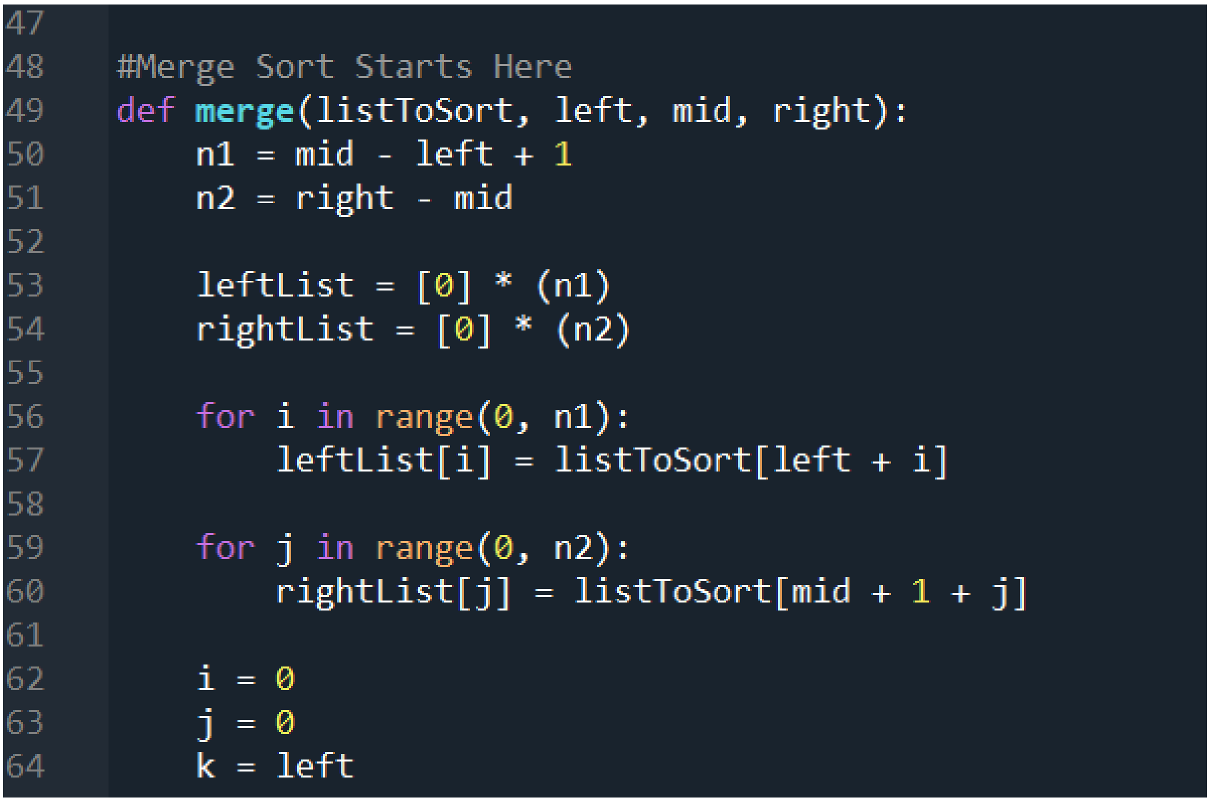
Code: - The Code was written and developed in Spyder by Anaconda IDE. Please make sure all the modules are successfully imported before running the below code.



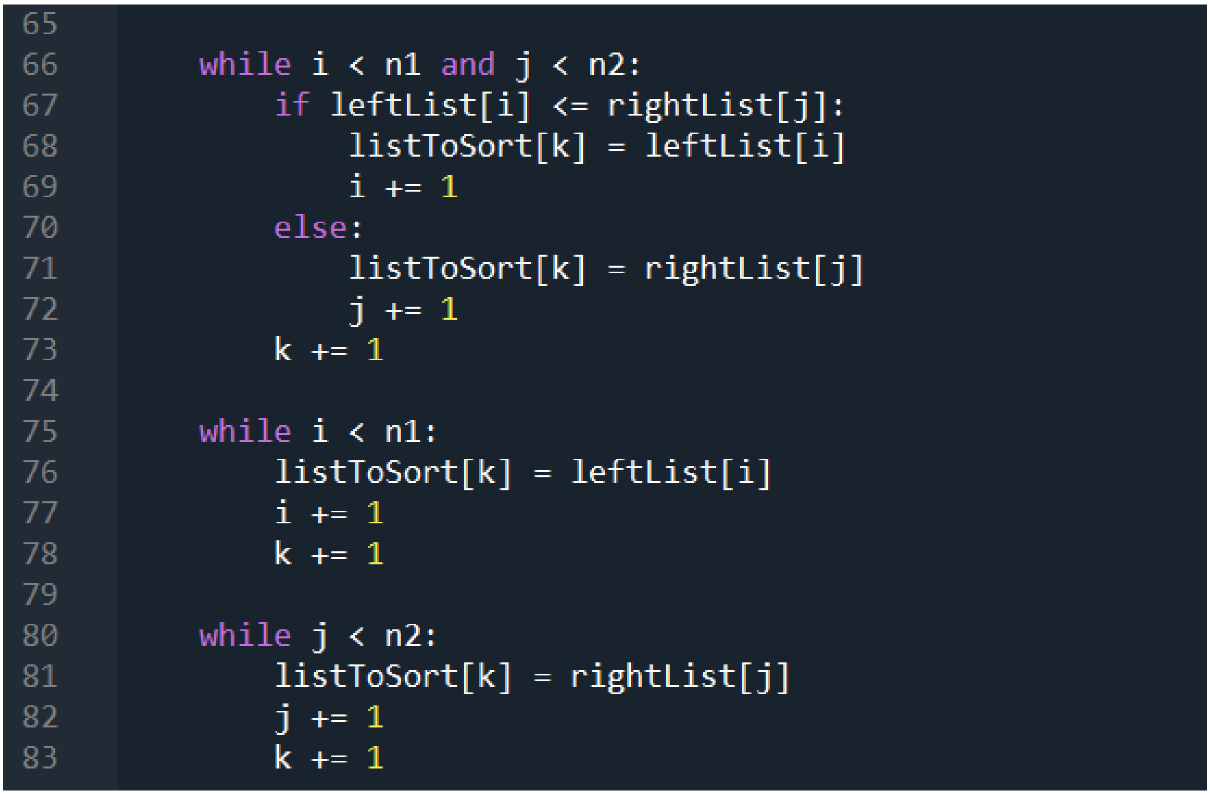
```python
# -*- coding: utf-8 -*-
"""
Created on Mon Aug 29 21:23:46 2022

@author      :  Avinash Mahala
@Student ID  :  1002079433
"""

#All imports goes below
import random
import timeit
import copy
import matplotlib.pyplot as plt
#All imports ends here


"""Input Number List Generator
(List Elements To be generated using Random Function)
Input   :  number of elements "N" to be generated & Returned.
Output :  list returning a list of "N" random numbers
"""

def randomNumListGen(rStart,rEnd,numOfElem):
    ranNumList=random.sample(range(rStart,rEnd), numOfElem)

    """
    Uncomment This block if you want to print the ramdomListGenerated
    ranNumListInString=' '.join(str(s) for s in ranNumList)
    print("Random List Generated--> "+ranNumListInString)
    """
    return ranNumList
```



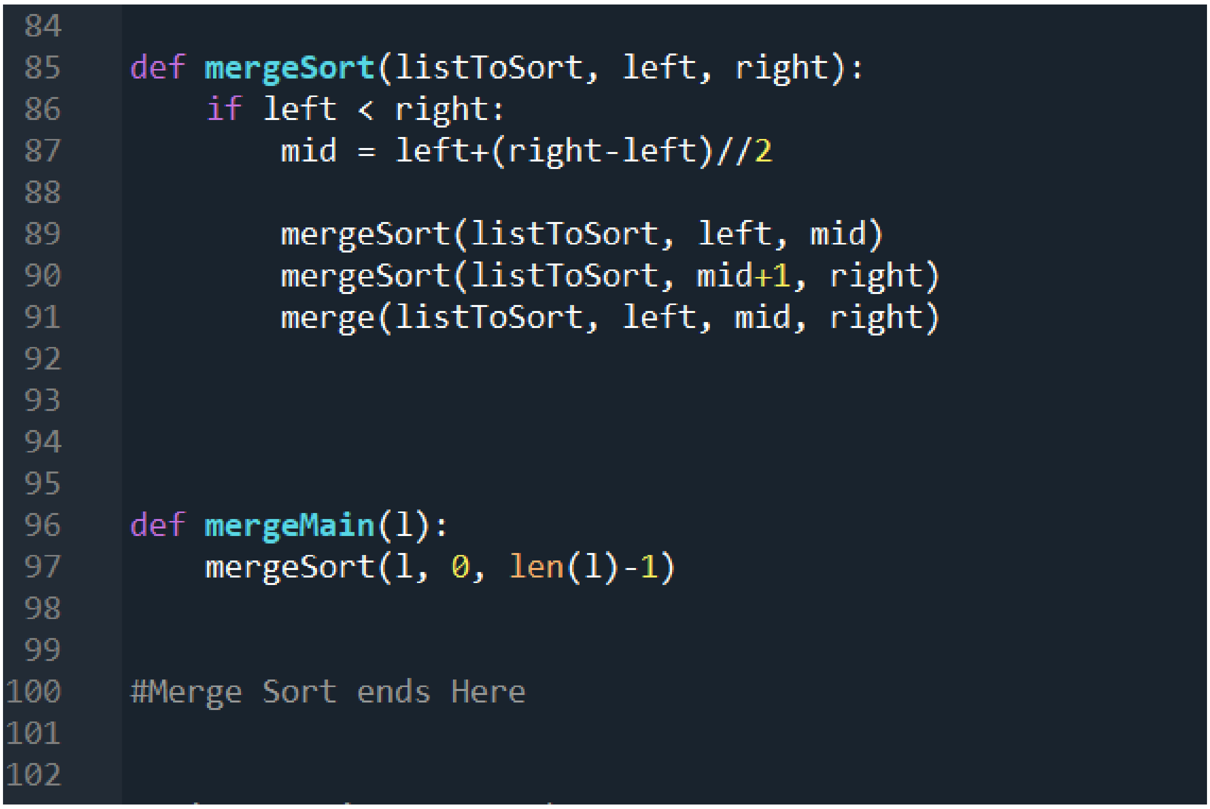
```python

#Insertion Sort Function starts
def insertionSort(toSort):
    for i in toSort:
        #print(i)
        j=toSort.index(i)
        while j>0:
            if toSort[j-1]>toSort[j]:
                toSort[j-1],toSort[j]=toSort[j],toSort[j-1]
            else:
                break
            j=j-1
#Insertion Sort Function ends

```



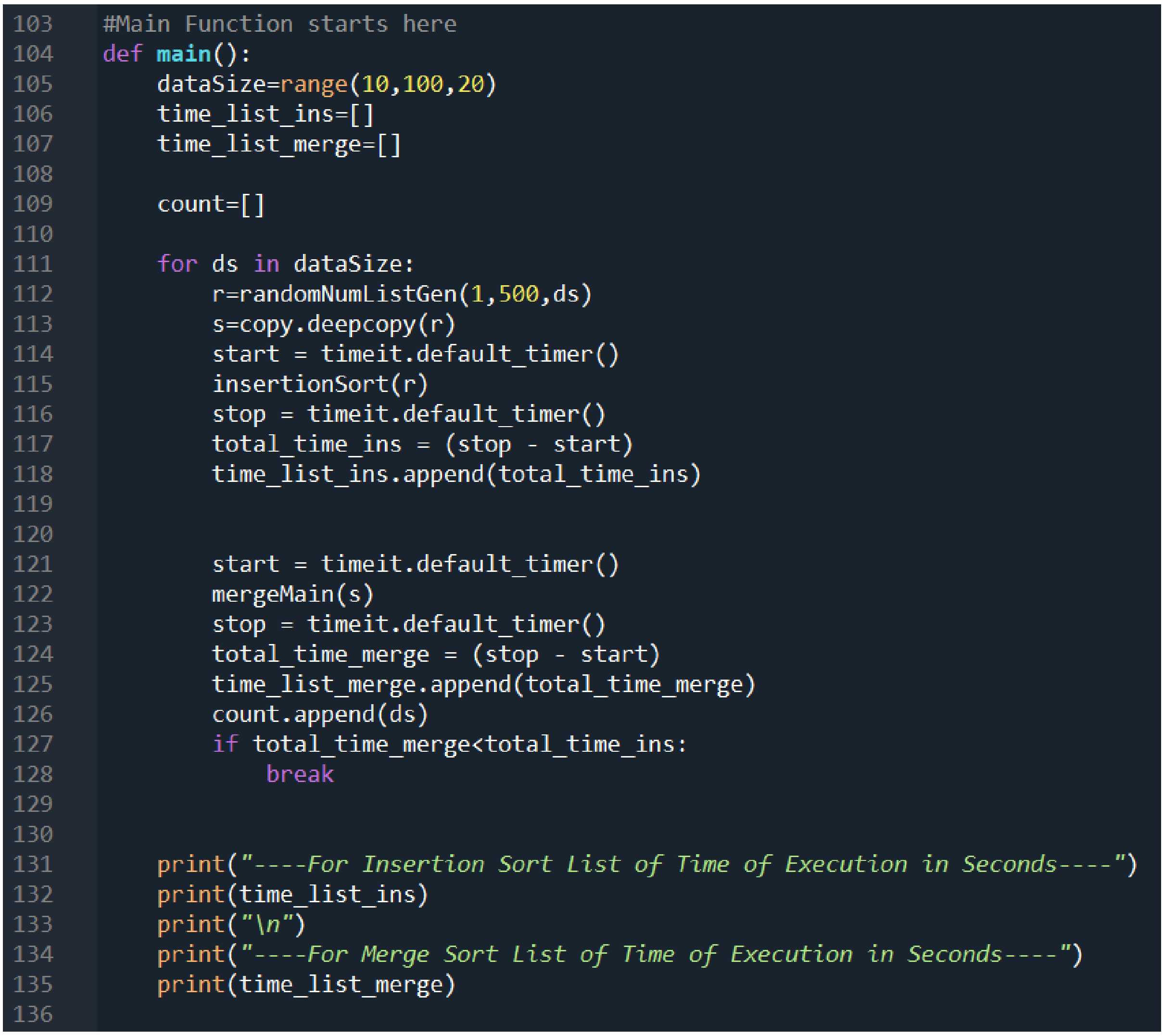
```python

#Merge Sort Starts Here
def merge(listToSort, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid

    leftList = [0] * (n1)
    rightList = [0] * (n2)

    for i in range(0, n1):
        leftList[i] = listToSort[left + i]

    for j in range(0, n2):
        rightList[j] = listToSort[mid + 1 + j]

    i = 0
    j = 0
    k = left
```



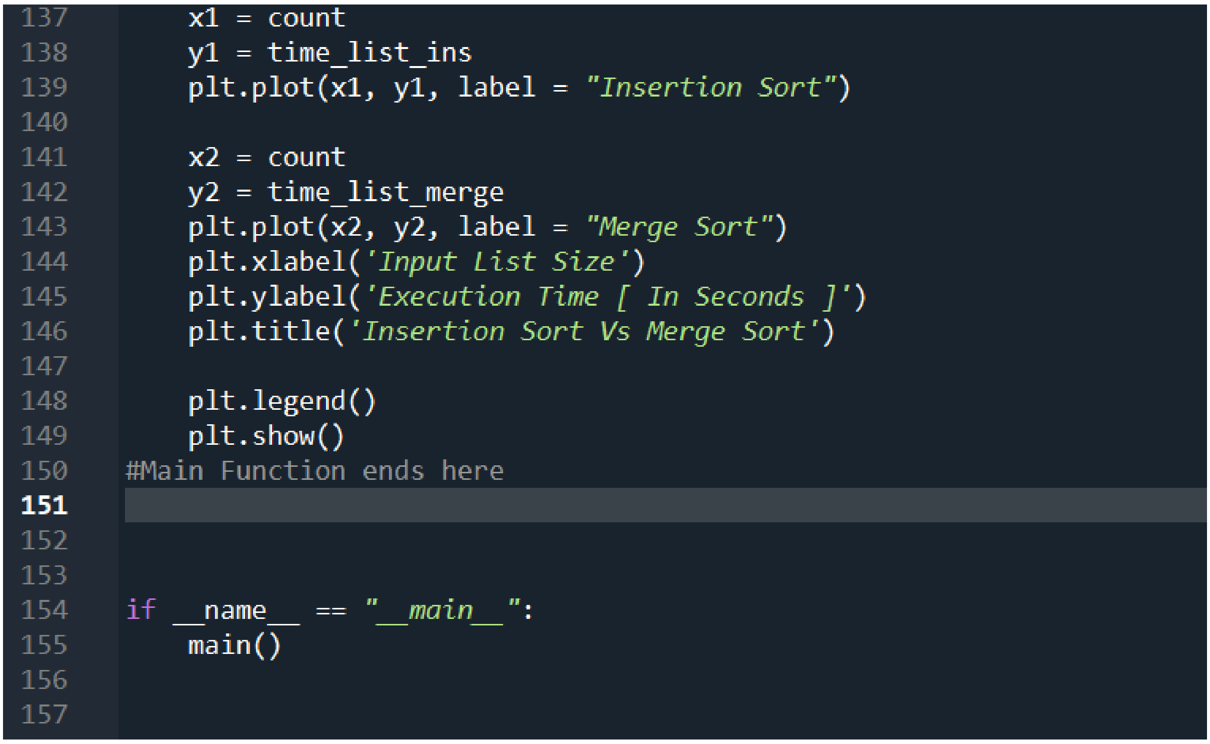
```

    while i < n1 and j < n2:
        if leftList[i] <= rightList[j]:
            listToSort[k] = leftList[i]
            i += 1
        else:
            listToSort[k] = rightList[j]
            j += 1
        k += 1

    while i < n1:
        listToSort[k] = leftList[i]
        i += 1
        k += 1

    while j < n2:
        listToSort[k] = rightList[j]
        j += 1
        k += 1
```



```

def mergeSort(listToSort, left, right):
    if left < right:
        mid = left+(right-left)//2

        mergeSort(listToSort, left, mid)
        mergeSort(listToSort, mid+1, right)
        merge(listToSort, left, mid, right)




def mergeMain(l):
    mergeSort(l, 0, len(l)-1)


#Merge Sort ends Here


```



```python
#Main Function starts here
def main():
    dataSize=range(10,100,20)
    time_list_ins=[]
    time_list_merge=[]

    count=[]

    for ds in dataSize:
        r=randomNumListGen(1,500,ds)
        s=copy.deepcopy(r)
        start = timeit.default_timer()
        insertionSort(r)
        stop = timeit.default_timer()
        total_time_ins = (stop - start)
        time_list_ins.append(total_time_ins)


        start = timeit.default_timer()
        mergeMain(s)
        stop = timeit.default_timer()
        total_time_merge = (stop - start)
        time_list_merge.append(total_time_merge)
        count.append(ds)
        if total_time_merge<total_time_ins:
            break


    print("----For Insertion Sort List of Time of Execution in Seconds----")
    print(time_list_ins)
    print("\n")
    print("----For Merge Sort List of Time of Execution in Seconds----")
    print(time_list_merge)

```



```python
    x1 = count
    y1 = time_list_ins
    plt.plot(x1, y1, label = "Insertion Sort")

    x2 = count
    y2 = time_list_merge
    plt.plot(x2, y2, label = "Merge Sort")
    plt.xlabel('Input List Size')
    plt.ylabel('Execution Time [ In Seconds ]')
    plt.title('Insertion Sort Vs Merge Sort')

    plt.legend()
    plt.show()
#Main Function ends here



if __name__ == "__main__":
    main()


```

Code With Line Numbers:-[Copy The below Code Without Line Numbers in Order to test in an IDE]

```
001: # -*- coding: utf-8 -*-
002: """
003: Created on Mon Aug 29 21:23:46 2022
004:
005: @author      :  Avinash Mahala
006: @Student ID  :  1002079433
007: """
008:
009: #All imports goes below
010: import random
011: import timeit
012: import copy
013: import matplotlib.pyplot as plt
014: #All imports ends here
015:
016:
017: """Input Number List Generator
018: (List Elements To be generated using Random Function)
019: Input  :  number of elements "N" to be generated & Returned.
020: Output :  list returning a list of "N" random numbers
021: """
022:
023: def randomNumListGen(rStart,rEnd,numOfElem):
024:     ranNumList=random.sample(range(rStart,rEnd), numOfElem)
025:
026:     """
027:     Uncomment This block if you want to print the ramdomListGenerated
028:     ranNumListInString=' '.join(str(s) for s in ranNumList)
029:     print("Random List Generated--> "+ranNumListInString)
030:     """
031:     return ranNumList
032:
033:
034: #Insertion Sort Function starts
```

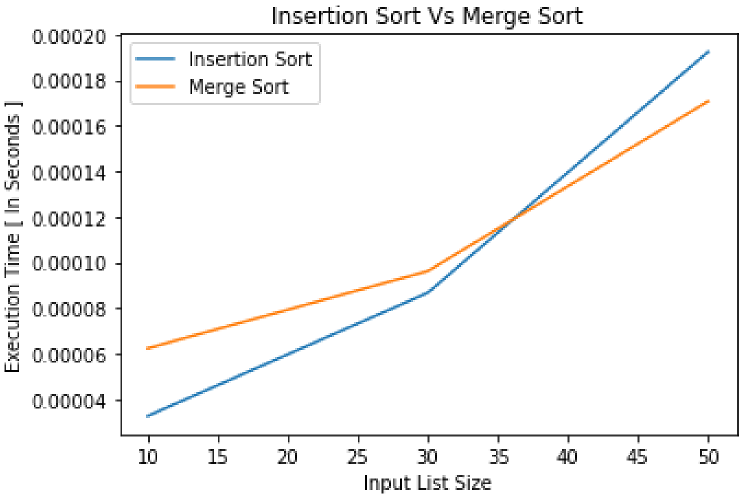
```
def insertionSort(toSort):
    for i in toSort:
        #print(i)
        j=toSort.index(i)
        while j>0:
            if toSort[j-1]>toSort[j]:
                toSort[j-1],toSort[j]=toSort[j],toSort[j-1]
            else:
                break
            j=j-1
#Insertion Sort Function ends


#Merge Sort Starts Here
def merge(listToSort, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid

    leftList = [0] * (n1)
    rightList = [0] * (n2)

    for i in range(0, n1):
        leftList[i] = listToSort[left + i]

    for j in range(0, n2):
        rightList[j] = listToSort[mid + 1 + j]

    i = 0
    j = 0
    k = left

    while i < n1 and j < n2:
        if leftList[i] <= rightList[j]:
            listToSort[k] = leftList[i]
            i += 1
        else:
            listToSort[k] = rightList[j]
            j += 1
        k += 1

    while i < n1:
        listToSort[k] = leftList[i]
        i += 1
        k += 1

    while j < n2:
        listToSort[k] = rightList[j]
        j += 1
        k += 1

def mergeSort(listToSort, left, right):
    if left < right:
        mid = left+(right-left)//2

        mergeSort(listToSort, left, mid)
        mergeSort(listToSort, mid+1, right)
        merge(listToSort, left, mid, right)




def mergeMain(l):
    mergeSort(l, 0, len(l)-1)


#Merge Sort ends Here


#Main Function starts here
def main():
```

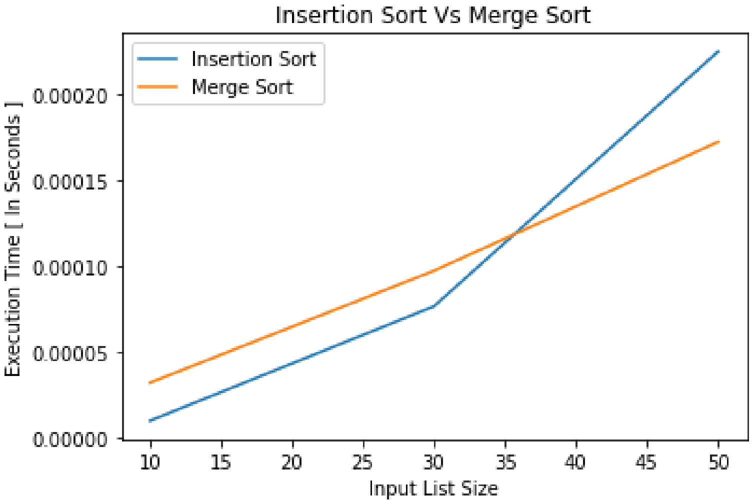
```
dataSize=range(10,100,20)
time_list_ins=[]
time_list_merge=[]

count=[]

for ds in dataSize:
    r=randomNumListGen(1,500,ds)
    s=copy.deepcopy(r)
    start = timeit.default_timer()
    insertionSort(r)
    stop = timeit.default_timer()
    total_time_ins = (stop - start)
    time_list_ins.append(total_time_ins)


    start = timeit.default_timer()
    mergeMain(s)
    stop = timeit.default_timer()
    total_time_merge = (stop - start)
    time_list_merge.append(total_time_merge)
    count.append(ds)
    if total_time_merge<total_time_ins:
        break


print("----For Insertion Sort List of Time of Execution in Seconds----")
print(time_list_ins)
print("\n")
print("----For Merge Sort List of Time of Execution in Seconds----")
print(time_list_merge)

x1 = count
y1 = time_list_ins
plt.plot(x1, y1, label = "Insertion Sort")
```

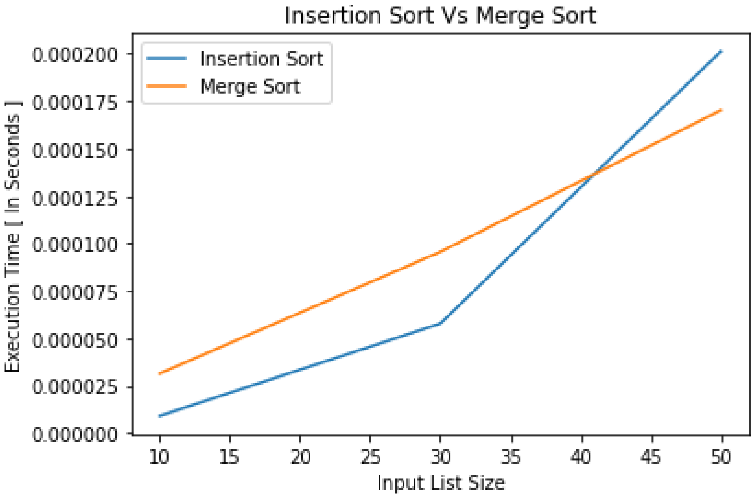
```

    x2 = count
    y2 = time_list_merge
    plt.plot(x2, y2, label = "Merge Sort")
    plt.xlabel('Input List Size')
    plt.ylabel('Execution Time [ In Seconds ]')
    plt.title('Insertion Sort Vs Merge Sort')

    plt.legend()
    plt.show()
#Main Function ends here



if __name__ == "__main__":
    main()
```

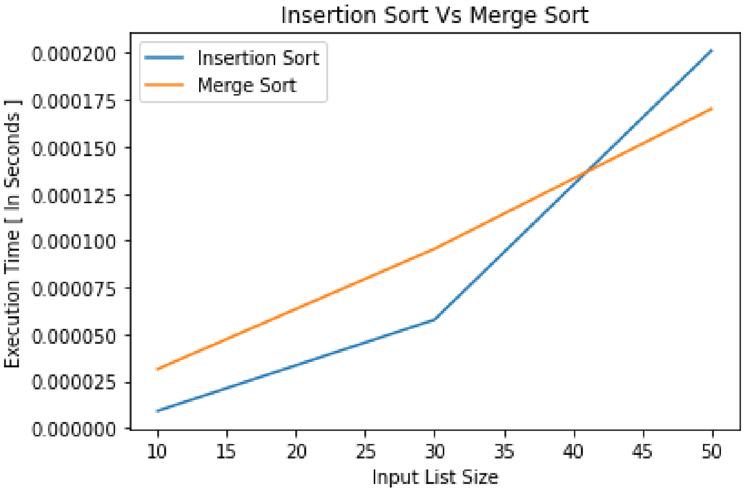
Multiple Graphs When Executed at Different Times:-

Here While executing the code at different time instances, I found that the graphs produced were not constant or same all the time, but there were similarities in terms of best case, average case and worst case time scenarios. Here every time the code ran, the run time performance was dependent on multiple external factors apart from input set of numbers. Hence, asymptotic analysis is meaningful in order to conclude the mathematically bounded run time performance. Asymptotic analysis is input bound. It means the algorithms run time complexity depends on only the input to the algorithm. All other factors are considered to be constant. Hence if there is no input to the algorithm, it is established to work in a constant time.


Insertion Sort Vs Merge Sort
Insertion Sort
Merge Sort
Execution Time [In Seconds]
0.00020
0.00018
0.00016
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
10
15
20
25
30
35
40
45
50
Input List Size


Insertion Sort Vs Merge Sort
Insertion Sort
Merge Sort
Execution Time [In Seconds]
0.00020
0.00015
0.00010
0.00005
0.00000
10
15
20
25
30
35
40
45
50
Input List Size


Insertion Sort Vs Merge Sort
Insertion Sort
Merge Sort
Execution Time [In Seconds]
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
10
15
20
25
30
35
40
45
50
Input List Size


Insertion Sort Vs Merge Sort
Insertion Sort
Merge Sort
Execution Time [In Seconds]
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
10
15
20
25
30
35
40
45
50
Input List Size

s

Q2) [40 points] Problem 2-1 on Page 39 of the CLRS textbook. ("2-1 Insertion sort on small arrays in merge sort")

2-1 Insertion sort on small arrays in merge sort

Although merge sort runs in ,.n lg n/ worst-case time and insertion sort runs in ,.n2/ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to ***coarsen*** the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n=k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

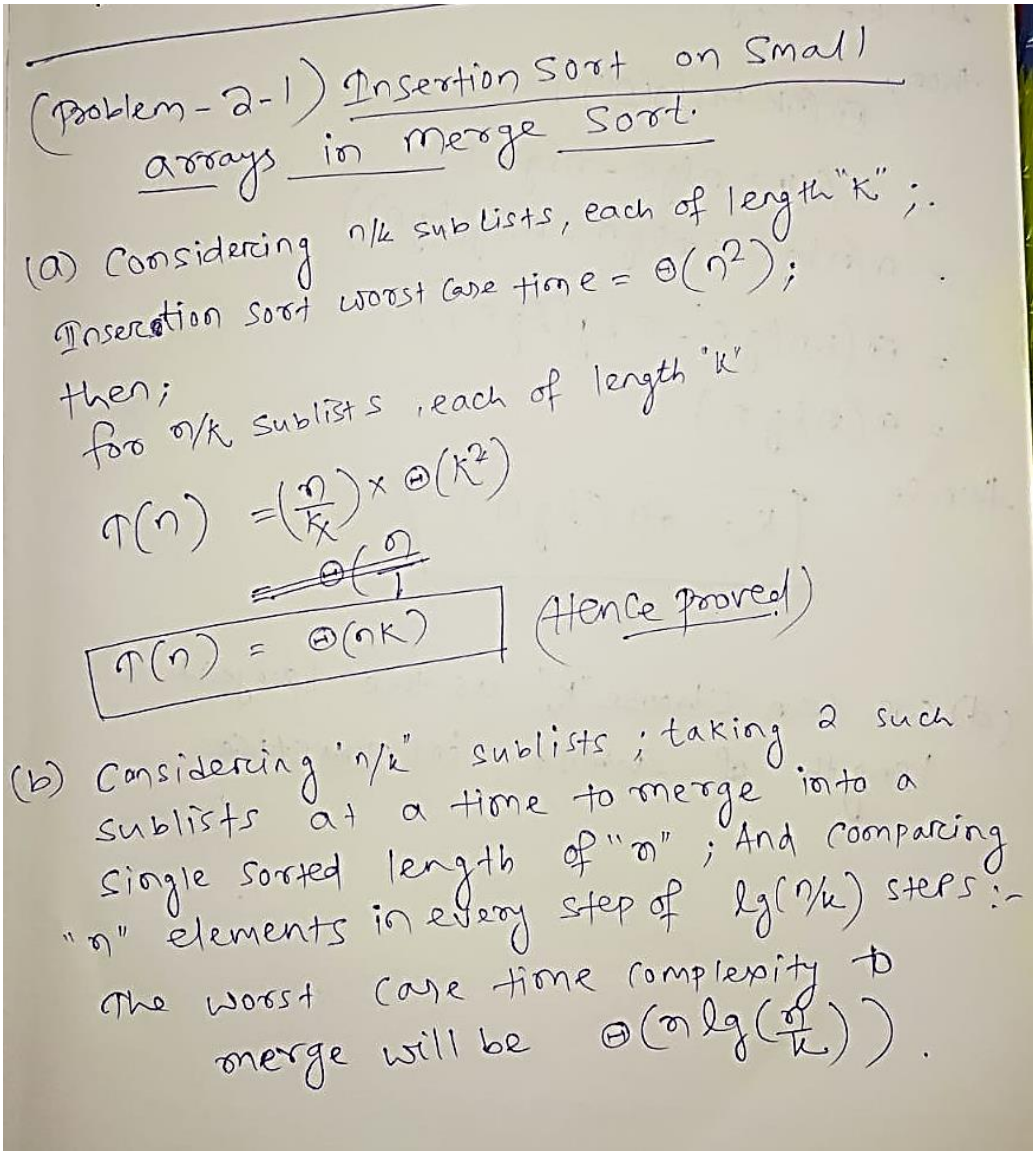
a. Show that insertion sort can sort the n=k sublists, each of length k, in ,.nk/ worst-case time.

b. Show how to merge the sublists in ,.n lg.n=k// worst-case time.

c. Given that the modified algorithm runs in ,.nk C n lg.n=k// worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of ,-notation?

d. How should we choose k in practice?

Answer:-



(Problem - 2-1) Insertion Sort on Small arrays in merge Sort.

(a) Considering n/k sublists, each of length "k";.

Insertion Sort worst case time = $\Theta(n^2)$;

then;

for n/k sublists, each of length 'k'

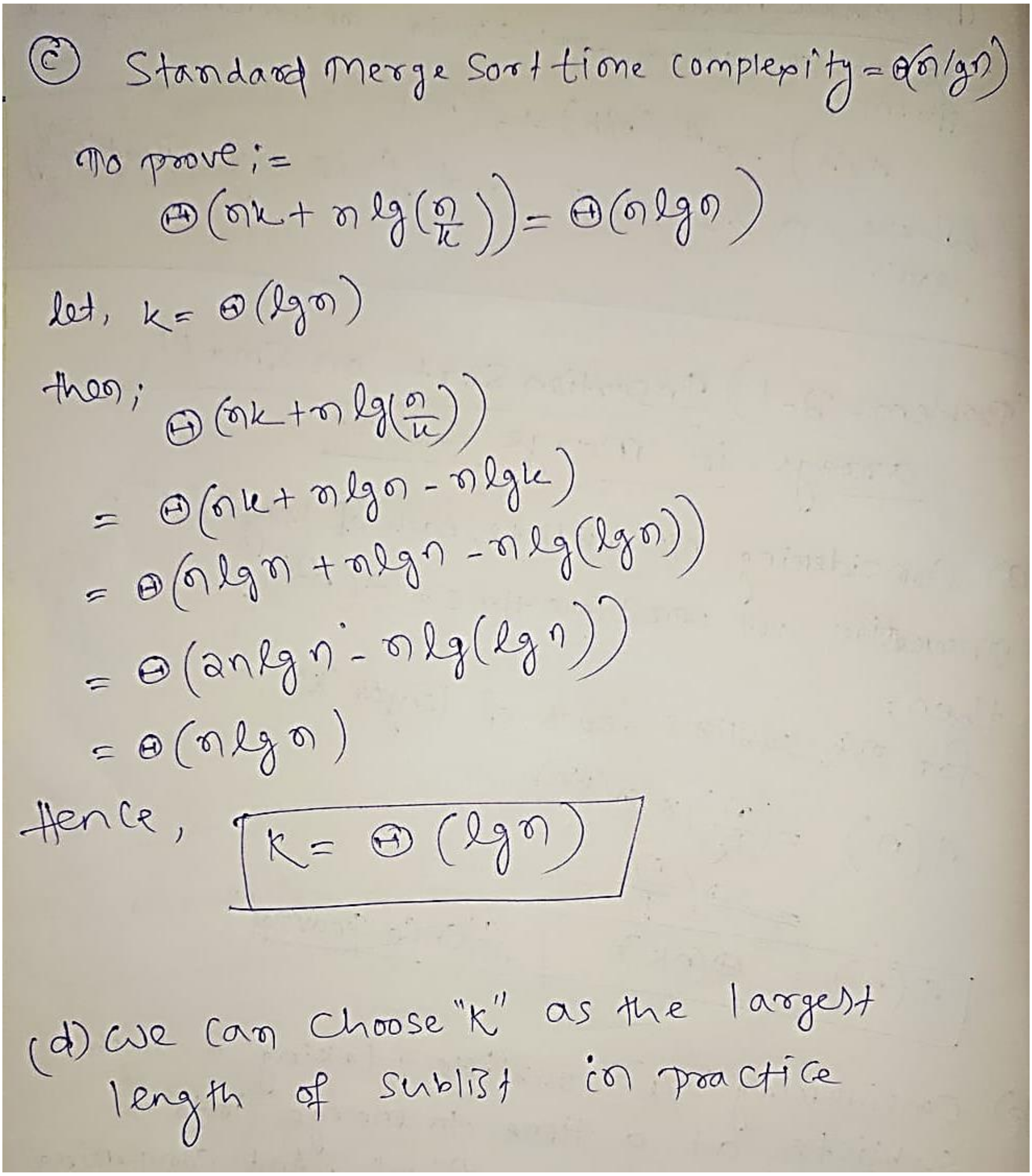
$$T(n) = \left(\frac{n}{k}\right) \times \Theta(k^2)$$

$$\cancel{= \Theta\left(\frac{n}{1}\right.}$$

$$\boxed{T(n) = \Theta(nk)}$$ (Hence proved)

(b) Considering 'n/k' sublists; taking 2 such sublists at a time to merge into a single sorted length of "n"; And comparing "n" elements in every step of $lg(n/k)$ steps:-

The worst case time complexity to merge will be $\Theta(n\,lg(\frac{n}{k}))$.



(c) Standard Merge Sort time complexity = $\Theta(n \lg n)$

To prove :-

$$\Theta(nk + n\lg(\tfrac{n}{k})) = \Theta(n\lg n)$$

let, $k = \Theta(\lg n)$

then; $\Theta(nk + n\lg(\frac{n}{k}))$

$= \Theta(nk + n\lg n - n\lg k)$

$= \Theta(n\lg n + n\lg n - n\lg(\lg n))$

$= \Theta(2n\lg n - n\lg(\lg n))$

$= \Theta(n\lg n)$

Hence, $\boxed{k = \Theta(\lg n)}$

(d) we can choose "k" as the largest length of sublist in practice

--------------------------------End of HomeWork-1----------------------------------