Data Analysis and Algorithm

Practical 2

Write a Program to implement Merge sort

&

Find the run time of the algorithm

Name – Yash Vasudeo Prajapati

Rollno - 022

MSc. Computer Science

→ Write a program to implement Merge sort & find the running time of the algorithm

→ For algorithms such as selection sort or insertion sort when we increase the input size the time taken also increase by a big margin.

Thus we look at algorithm like merge sort which has a runtime of $n \log n$ in all cases i.e with large input size the run time is much better than insertion sort.

Merge sort uses the Divide & Conquer technique to first divide a large array into smaller section particularly 2 equal sections & sorting those section independently & recursively & at last combining.

Thus, the algorithm has 3 parts.

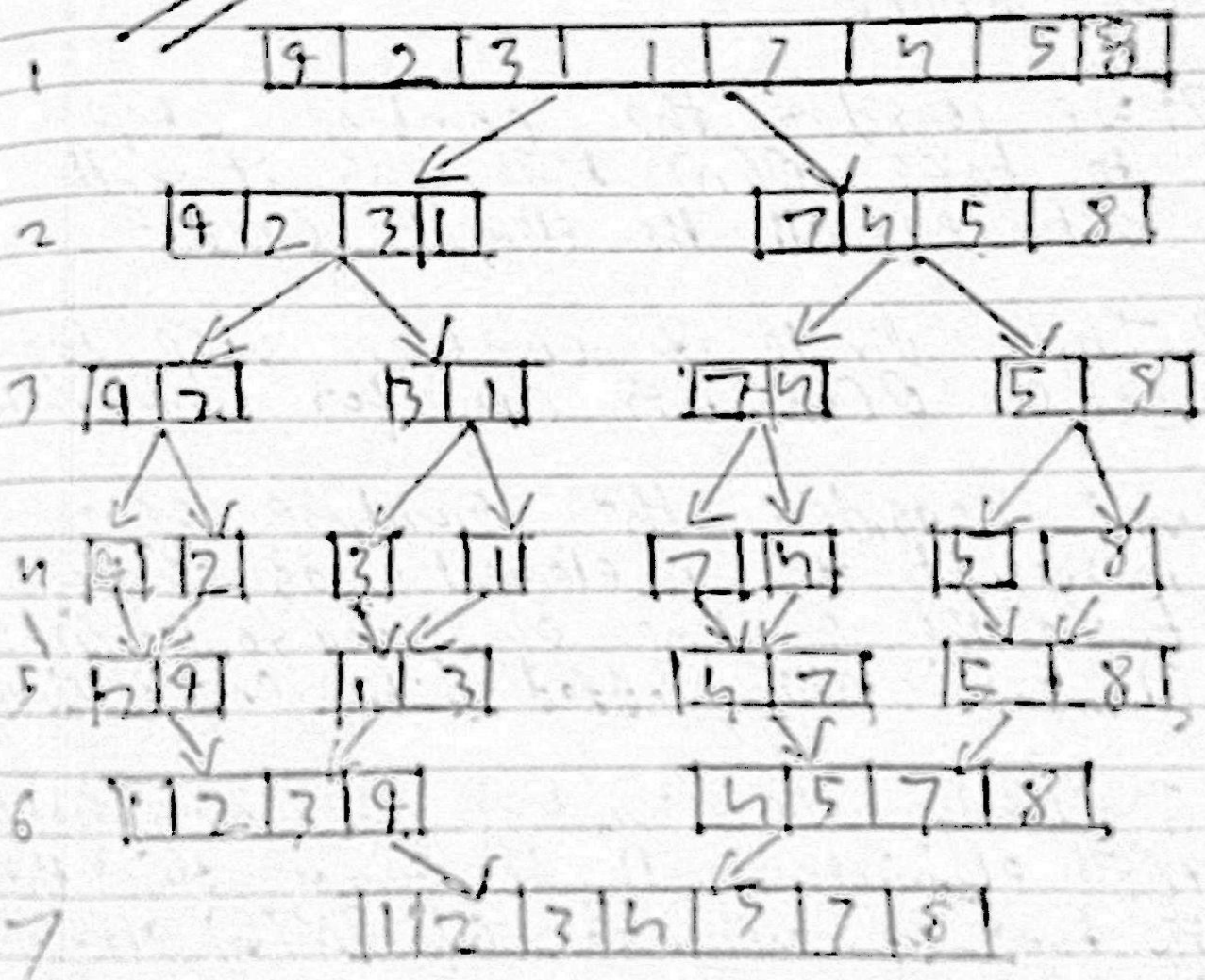
Divide, Conquer & combine

→ Divide → Divide the array into 2 i.e index[p..q] i.e ptr

& you get two arrays arr[p, ptr] & arr[p, ptr]

Combine:- Merge the sorted subarrays into one

Example



Where on steps 1-4 we divide & from 5-7 we conquer & merge

→ Runtime of merge sort algorithm

To calculate the runtime of merge sort we look at the 3 steps & the runtimes of the same

→ we consider the divide step to have a constant time because in the step all we do is calculate the midpoint of array.

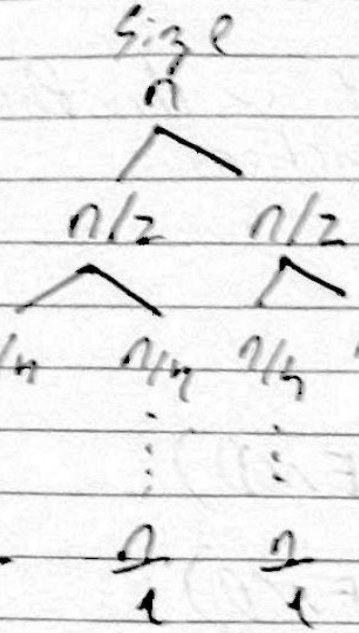
→ we consider the combine step to take $O(n)$ time as it will act on all the element once.

→ Thus divide & combine step together take $O(n)$ or cn for some constant

→ we consider the runtime of mergesort of n elements as sum of twice the runtime of mergesort of $(n/2)$ elements array added with cn i.e divide & combine

→ Similarly the runtime of mergesort of $(n/2)$ elements will be sum of twice the runtime of mergesort of $(n/4)$ elements + cn

Thus we see,



time
cn
$2 \cdot cn/2 = cn$
$4 \cdot cn/4 = cn$
$l \cdot cn/l = cn$
since l is also a constant

i.e total time of merge sort will be sum of merging time of each level.
i.e. for l levels we get l.cn

where l is given as

$$l = \log_2 n + 1$$

Eg:- $n = 8$

$$l = \log_2 8 + 1$$

$$l = 4$$

i.e. for $n = 8$ we have 4 levels.

thus we substitute $\log_2 n + 1$ in l.cn we get,

$$cn(\log_2 n + 1)$$

For Big O notation we discard constant c & low order term (+1)

thus get

$$O(n \log_2 n)$$

⇒ Algorithm for merge sort

- ⇒ Merge sort is implement with the use of recursive functions.

1) arr = input
   BEG = 0
   END = arr.length
   mergeSort (arr, BEG, END)

2) mergeSort (arr, BEG, END)
   if (BEG < END)
      mid = (BEG + END)/2
      mergeSort (arr, BEG, MID)
      mergeSort (arr, MID+1, END)
      Merge (arr, BEG, MID, END)

3) Merge (arr, BEG, MID, END)
   i = BEG
   j = MID+1
   x = 0
   while (i <= MID && j <= END)
      if arr[i] < arr[j]
         temp[x] = arr[i]
         i = i + 1
      else
         temp[x] = arr[j]
         j = j + 1
      x = x + 1

4) if i > MID
while j <= END
temp[x] = arr[j]
x = x + 1
j = j + 1
else
while i <= MID
temp[x] = arr[i]
x = x + 1
i = i + 1

Step 1 → Calling code to initate the function

Step 2 → Recursively calling merge sort with array split in half & calling merge to organize the array sort

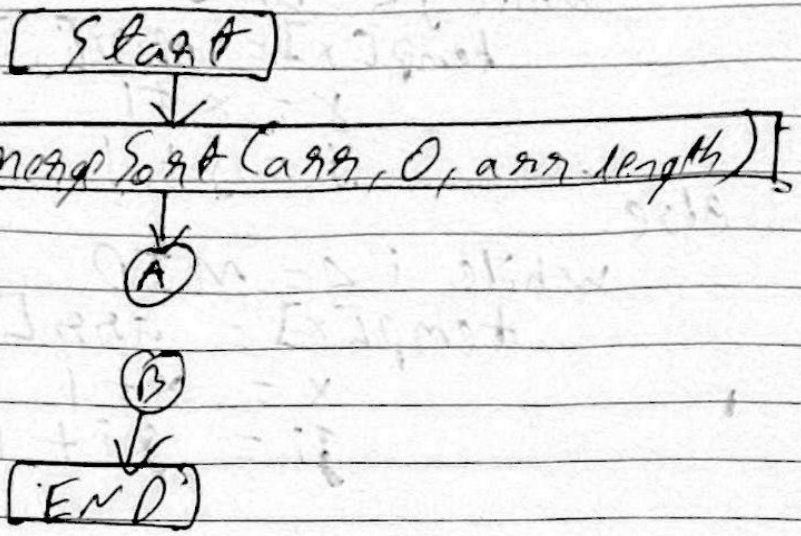
Step 3 → Sorting the sub array.

Step 4 → merge the remaing array

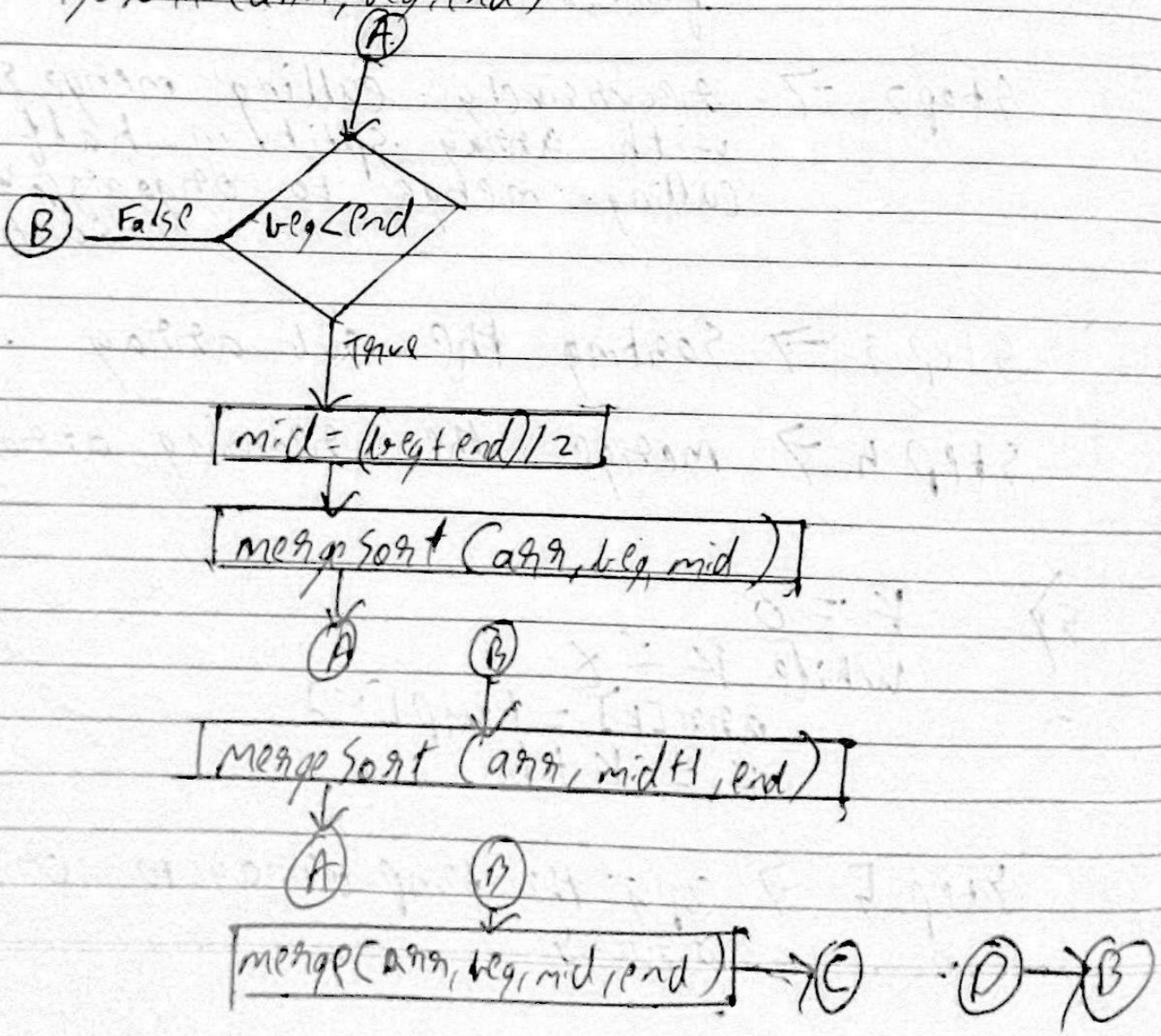
5) K = 0
while I <= x
arr[k] = temp[k]
K = K + 1

Step 5 → copy the temp array to original array

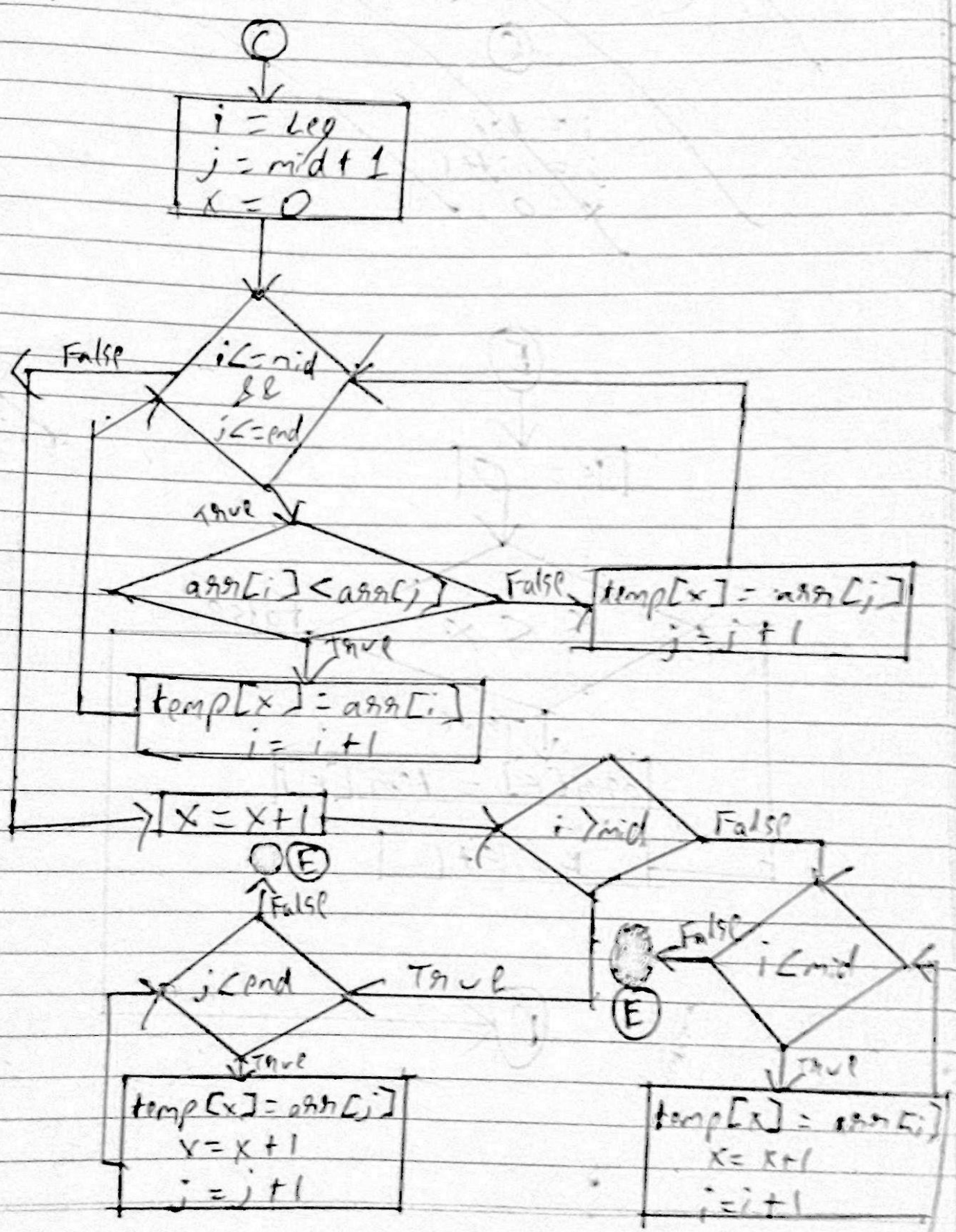
→ flowchart

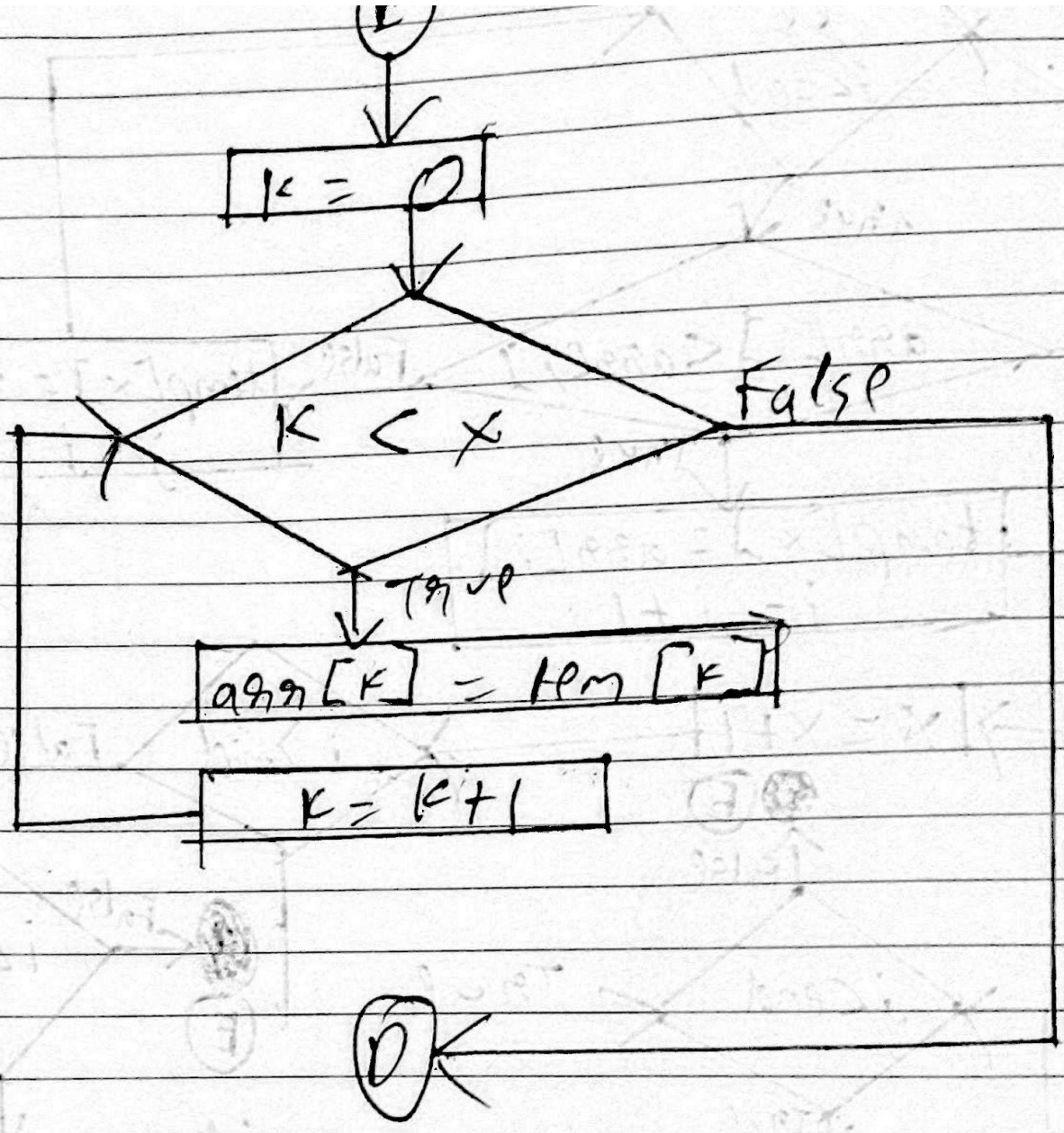


→ MergeSort(arr, beg, end)



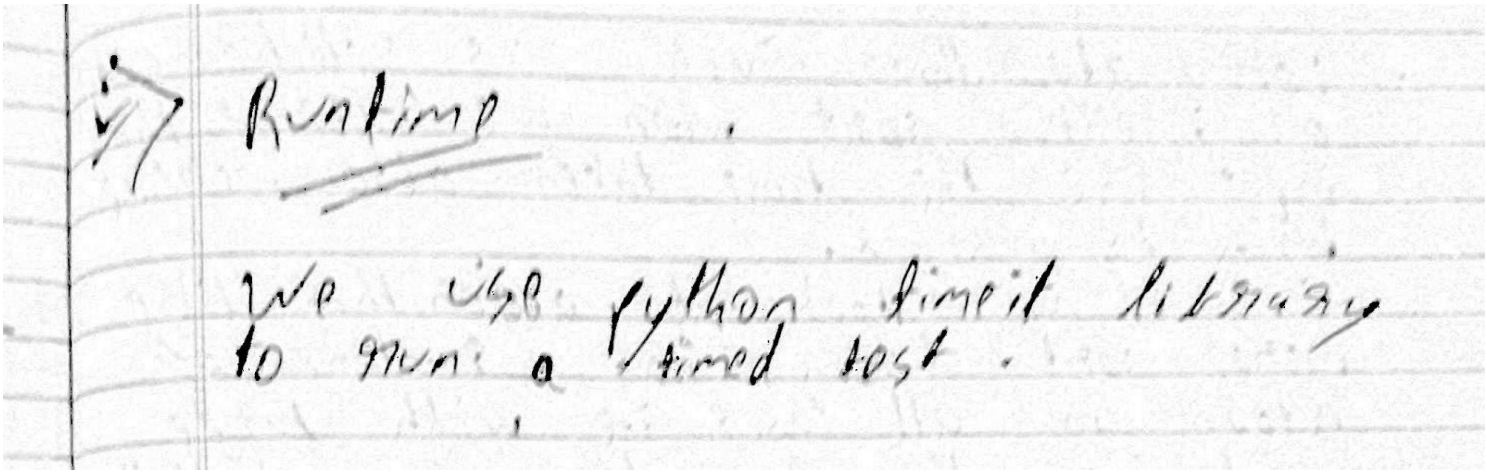
-7 Merge(arr, leg, mid, end)





Input

We use Python random library to create a list of n number to serve as an input



Program

```
1.  import random
2.  def merge(arr, l, m, r):
3.      n1 = m - l + 1
4.      n2 = r - m
5.
6.      # create temp arrays
7.      L = [0] * (n1)
8.      R = [0] * (n2)
9.
10.     # Copy data to temp arrays L[] and R[]
11.     for i in range(0, n1):
12.         L[i] = arr[l + i]
13.
14.     for j in range(0, n2):
15.         R[j] = arr[m + 1 + j]
16.
17.     # Merge the temp arrays back into arr[l..r]
18.     i = 0     # Initial index of first subarray
19.     j = 0     # Initial index of second subarray
20.     k = l     # Initial index of merged subarray
21.
22.     while i < n1 and j < n2:
23.         if L[i] <= R[j]:
24.             arr[k] = L[i]
25.             i += 1
26.         else:
27.             arr[k] = R[j]
28.             j += 1
29.         k += 1
30.
31.     # Copy the remaining elements of L[], if there
32.     # are any
33.     while i < n1:
34.         arr[k] = L[i]
35.         i += 1
36.         k += 1
37.
38.     # Copy the remaining elements of R[], if there
39.     # are any
40.     while j < n2:
41.         arr[k] = R[j]
42.         j += 1
43.         k += 1
44.
45. # l is for left index and r is right index of the
46. # sub-array of arr to be sorted
47. def mergeSort(arr, l, r):
48.     if l < r:
49.         # Same as (l+r)//2, but avoids overflow for
50.         # large l and h
51.         m = l+(r-l)//2
52.         # Sort first and second halves
```

```
53.         mergeSort(arr, l, m)
54.         mergeSort(arr, m+1, r)
55.         merge(arr, l, m, r)
56.         print(arr)
57.      return(arr)
58.
59. #driver code
60. if __name__ == "__main__":
61.     arr=[]
62.     for i in range (1,10):
63.         n = random.randint(0,1000)
64.         arr.append(n)
65.     mergeSort(arr, 0, len(arr)-1)
66.           print(mergeSort)
```

```
================== RESTART: C:\Users\ADMIN\Desktop\
[527, 771, 932, 266, 393, 59, 781, 553, 949]
[527, 771, 932, 266, 393, 59, 781, 553, 949]
[527, 771, 932, 266, 393, 59, 781, 553, 949]
[266, 393, 527, 771, 932, 59, 781, 553, 949]
[266, 393, 527, 771, 932, 59, 781, 553, 949]
[266, 393, 527, 771, 932, 59, 781, 553, 949]
[266, 393, 527, 771, 932, 59, 553, 781, 949]
[59, 266, 393, 527, 553, 771, 781, 932, 949]
[59, 266, 393, 527, 553, 771, 781, 932, 949]
>>>
```

Timing code

```
1.  SETUP_CODE = '''
2.  from __main__ import mergeSort,merge
3.  import random
4.  '''
5.
6.  TEST_CODE = '''
7.  arr=[]
8.  input_size = 10
9.  for i in range (1,input_size):
10.     n = random.randint(0,1000)
11.     arr.append(n)
12. mergeSort(arr, 0, len(arr)-1)'''
13.
14. times = timeit.timeit(setup = SETUP_CODE,
15.                       stmt = TEST_CODE,
16.                       number = 1)
17.
18. print(times)
19.
```

Output time with array of size 10

```
============== RESTART: C:/Users/ADMIN/Deskto
[4, 279, 452, 588, 670, 775, 785, 820, 980]
0.020413899999999985
```

Output time with array of size 100000

```
0.8353626
>>>
```

7 Conclusion

We observe that merge sort take very small amount of time when the input size in increased as compared to other linear algorithms.