**Problem Set 1**



Session: 2021 – 2024

**Submitted by:**

Ali Haider 2021-CS-38

**Submitted To:**

Sir Samyan Qayyum Wahla

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Insertion Sort**

1. Dry run the Insertion Sort on the following inputs. Use the code given in CLRS Section 2.1. Show the detailed iteration.

A= {5,43,76,2,98,23,12,32}

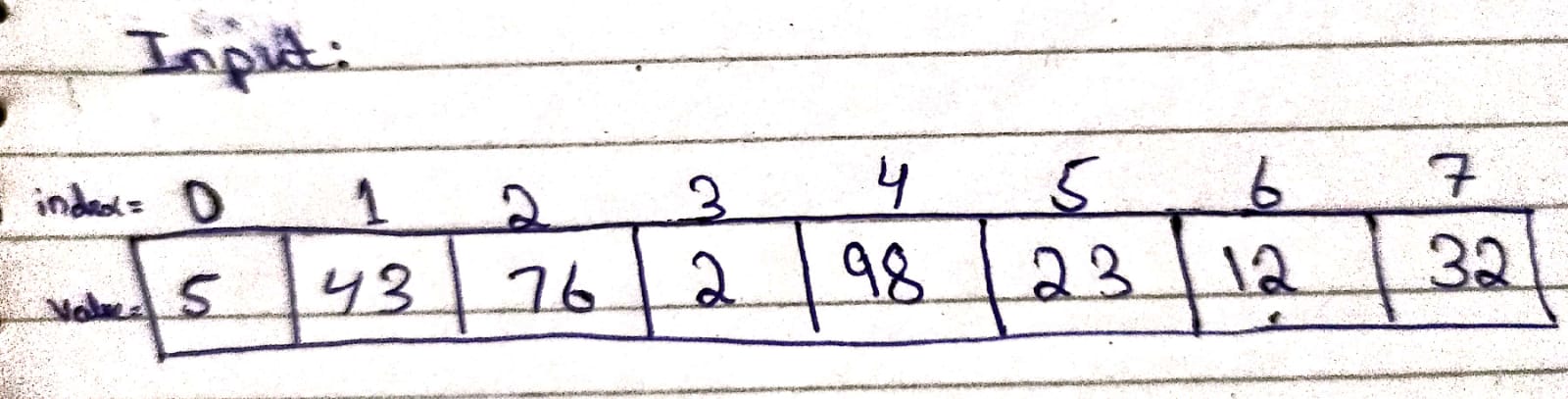
B= {6,7,8,9,10}

**Insertion Sort Dry Run:**

**Case 1:**

**Input:**

This is our Input Array. Let it be equal to A. In insertion sort we assume that our index is already sorted so we will start our loop from index 1

****

**Iteration 1:**

**A picture containing letter

Description automatically generated** In first Iteration, we will compare first index(1) to zero index and if first index is smaller than zero index(A[i+1]<A[i]) then swap them. In this case it is not true so we do not swap them and move to next Iteration.

**Iteration 2:**

**A picture containing letter

Description automatically generated** In second iteration, we will compare second index to first index and condition is true than swap them and then compare first and zero index and if condition is true swap them which is not in this case

**Iteration 3:**

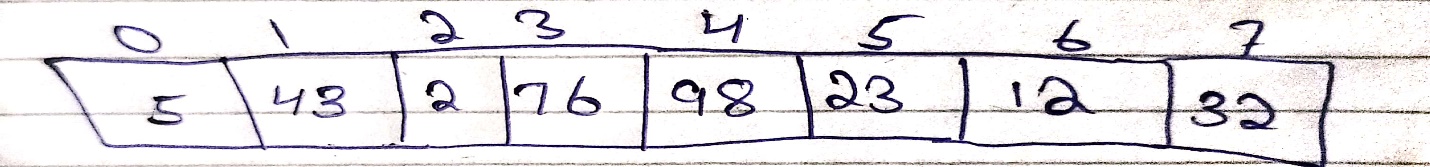
**A picture containing engineering drawing

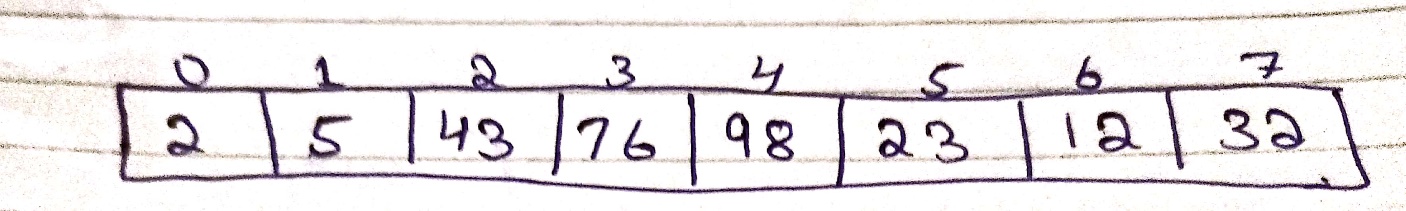
Description automatically generated** In third iteration, we will compare third index to second index and condition is true than swap them and then compare second and first index and if condition is true swap them and

so on. As in this iteration our condition satisfies so

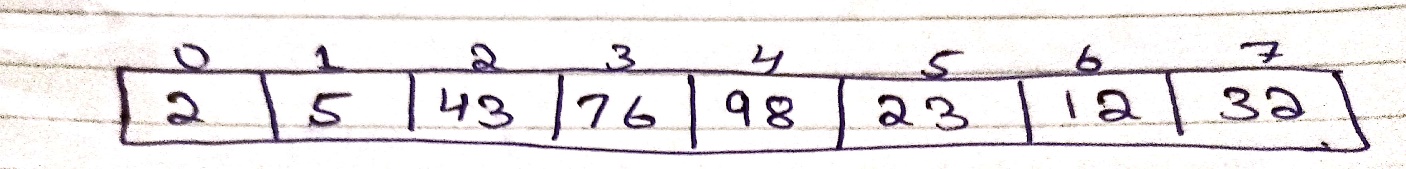
So after first swapping, our array becomes

A picture containing text

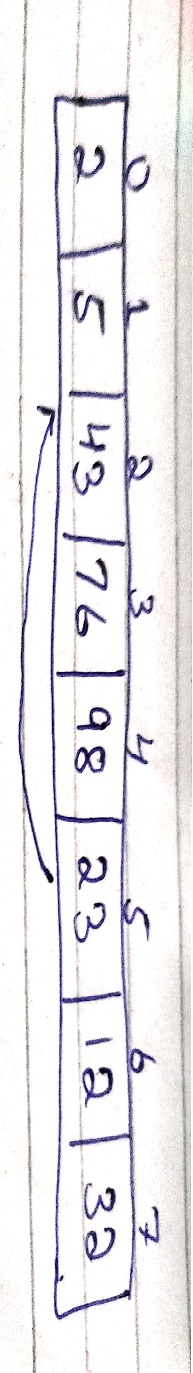
Description automatically generatedAfter second swapping our array becomes

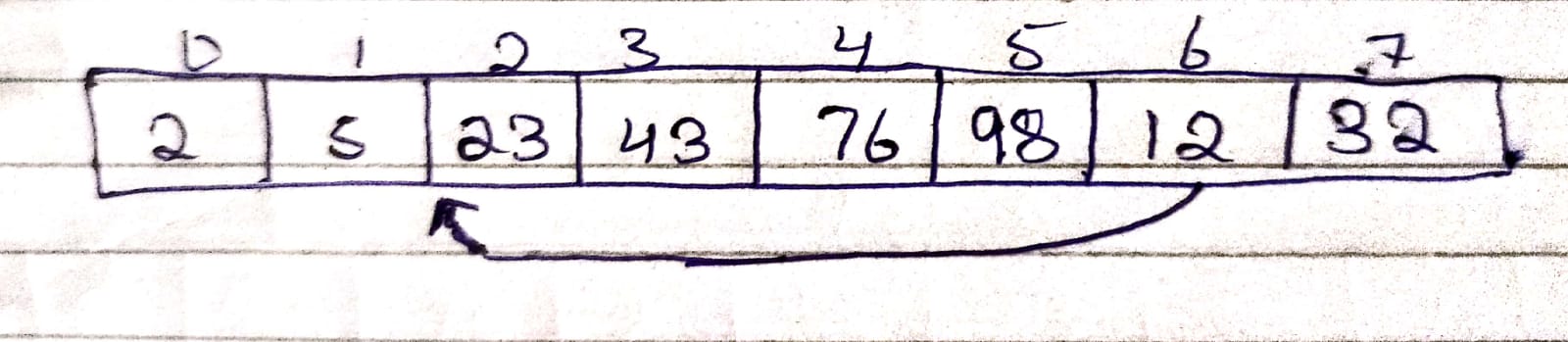
After third swapping and after Iteration our array becomes

**Iteration 4:**

Same process is repeated as present index is compared to previous index and condition satisfies then previous index to before previous index and so on

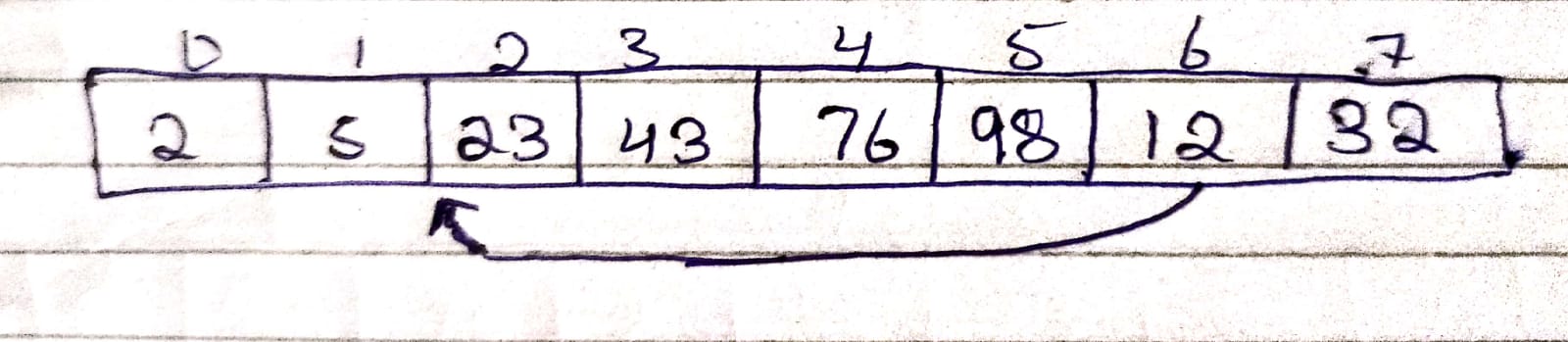
**Iteration 5:**

Same process. As in this case our condition satisfies so we swap them So

So After iteration our array becomes

**Iteration 6:**

Condition Satisfies so we swap them till condition become false

After Iteration array becomes

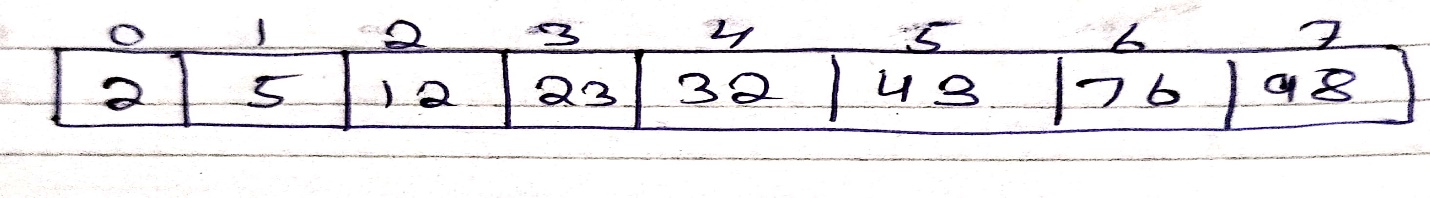
A picture containing text

Description automatically generated**Iteration 7:**

A picture containing text

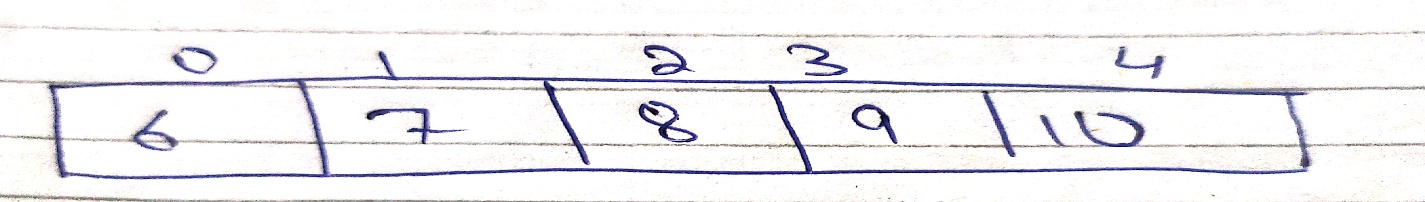
Description automatically generatedSame Process

After last Iteration our array becomes

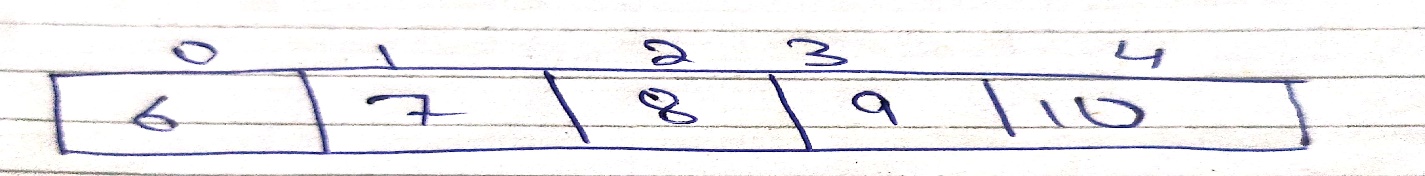
So after 7 iteration(size of array) our array becomes sorted

**Case 2(Sorted Array):**

B= {6,7,8,9,10}

** Input:**

In case of sorted array, time complexity of insertion sort becomes best as in first iteration first index is already greater than zero index so no swapping is done. Similarly in second iteration second iteration is greater than first condition so swapping is done so we can move to next iteration and so on so time complexity of Insertion sort in sorted array becomes O(n). After alteration our array remains same

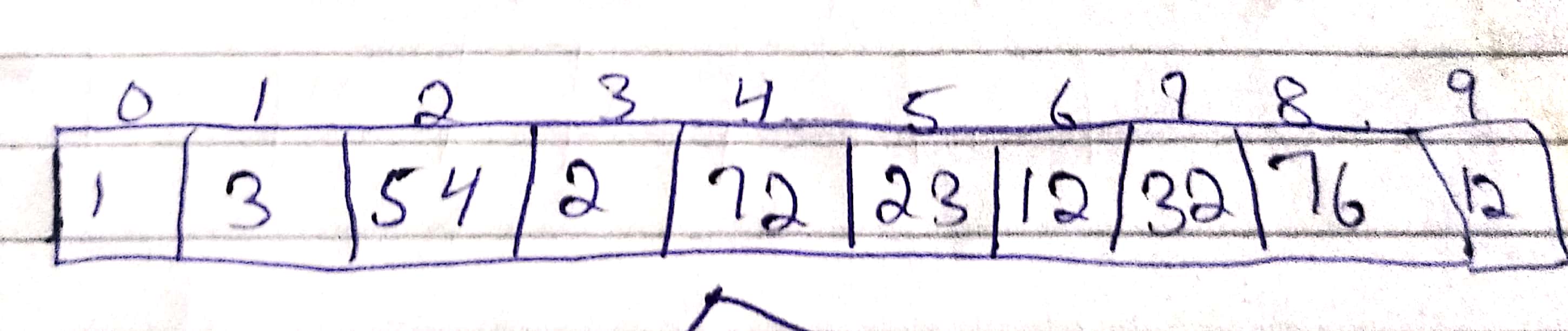
****

**Merge Sort**

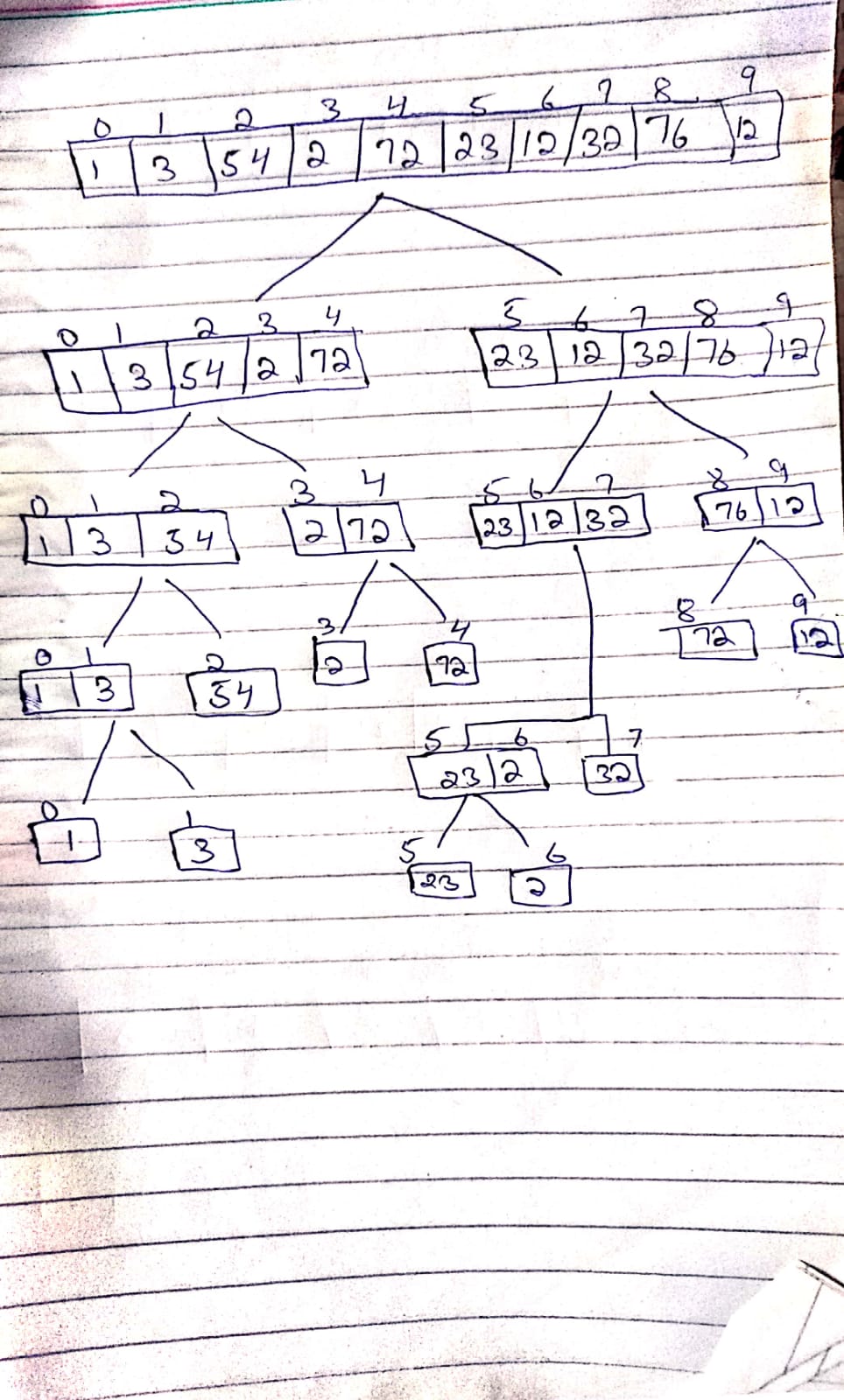
1. Dry run the Merge ;Sort on the following inputs. Use the code given in CLRS Section 2.3. Show in detail what happens in each recursive function.

A= {1,3,54,2,72,23,12,32,76,12}

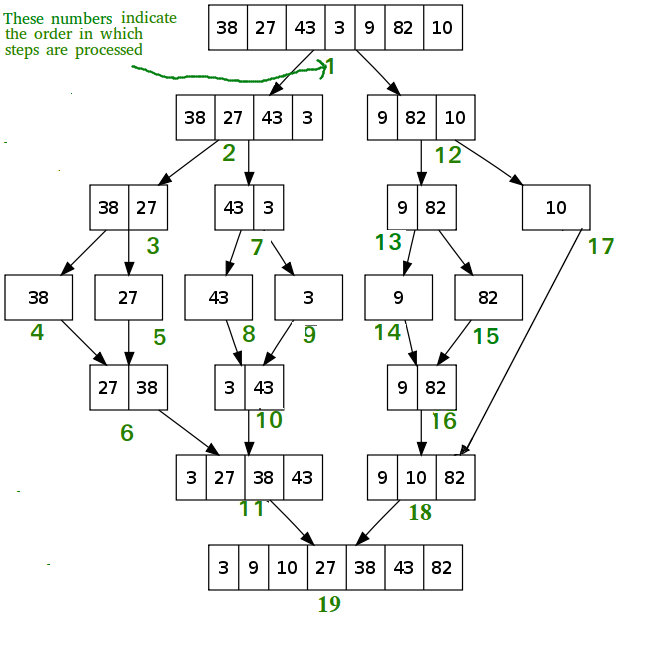
**Input:**

****In Merge Sort, we use divide and conquer approach

* **Divide** : We divide the problem in 2 sub problem which continue until the problem set is left with one element only
* **Recursively solve:** Here, the sub-arrays are sorted using recursion.
* **Conquer**: Conquer basically merges the 2 sorted array into the original array

**** **Merge Sort Dry Run:**

**Process :**

In this, first array is divided in 2 sub array. As array is of 10 size so after first division 2 sub array of 5 index forms. We then divide first sub array again so new array array becomes of 3 index and 2 indexs then we again divide 3 index aray in two array of size two and one then size two array is again divide in two array of size one and in this condition if condition becomes false and merge sort will sort them and and then second sub array of size will be will divide same like first sub array and after it size becomes 1 and if condition becomes false and the merge sort will be called and it will sort the both array comparing them

3. Visualize the function calls in the memory for the above input

We have a array of size 10 so we divide it in two halves as in above hand written picture. Then first first array of size 5 is divided again in two halves. Then first part array is divided by “Merge Sort” until array element become 1 and same process is repeated for second part till second part array element becomes 1 then all the elements are sorted using “Merge-Sort” and joined them with the help of function “Merge”, thus giving a sorted array

4.Do you think that Merge Sort always takes same number of operations in sorted and unsorted array. Justify your answer.

Yes, merge sort always takes same of operation whether it is sorted or unsorted array. Even in best, worst or average case it will always takes O(n log n) time because the depth of tree will always be same. We always divide the array whether it is sorted or unsorted array until it size becomes 1 so number of comparison remains same so same time will taken for both cases. So, Merge sort is more suitable for bigger n values

5. Change the pseudo code of Merge Sort to arrange elements in decreasing order rather than increasing order. Write down the modified pseudo code.

MERGE(A, p, q, r)

1- n1=q-p+1

2- n2=r-q

3- let L[1….. n1 + 1] and R[1…..n2+1 ] be new arrays

4- for i= 1 to n1

5- L[i]=A[p+i-1]

6- for j= 1 to n2

7- R[j]=A[q+j]

8- L[n+1]=infinity

9- R[n+1]=infinity

11- i=1

12- j=1

13- for k= p to r

13- if L[i]>R[j]

14- A[k]=L[i]

15- i=i+1

16- else A[k]= R[j]

17- j=j+1

**Selection Sort**

6. Consider the following code of Selection Sort. Provide the detailed asymptotic analysis of code similar to that of Insertion Sort discussed in class.

Function Selection-Sort(A,n): Cost Times

1. for i=1 to n-1 do c1 n-1
2. Min<I c2 n-1
3. For j =i+1 to n do c3 (n-j+1)
4. If A[j]<A[min] then c4 (n-j)
5. Min<j c5 (n-j)
6. End if
7. End for
8. Swap A[i] ,A[min] c6 n-1
9. End for
10. End function

It has O(n^2) time complexity

7. Discuss the Loop Invariant of Selection Sort

There are in fact two loops. There are invariants in both. In the outer loop we have the invariant that

* a[0...i-1] is sorted
* all entries in a[i..n-1] are larger than or equal to the entries in a[0..i-1]

Then in the inner loop, we still have the above invariant, but then we also have

* All entries in a[i..j-1] are greater than or equal to a[best].

When the inner loop terminates, that last invariant tells us that the variable best indexes a minimum element of the unsorted array, so we can swap it with the item at index i.

**Bubble Sort**

8. Write down one paragraph description of Bubble Sort in your own words.

In bubble sort, we start from zero index and compare with adjacent index and if condition satisfies then swap them and compare first index to second index and if condition satisfies swap them again so on comparing with adjacent indexes. Then in second pass again adjacent array are compares till the last and swapping them. Bubble Sort is not suitable for larger n because it worst and average time complexity is quite high(O(n^2))

9. Provide pseudo code of Bubble Sort.

bubbleSort(arr):

    n = len(arr)

    for i=0 to n

        for j=0 to n-i-1

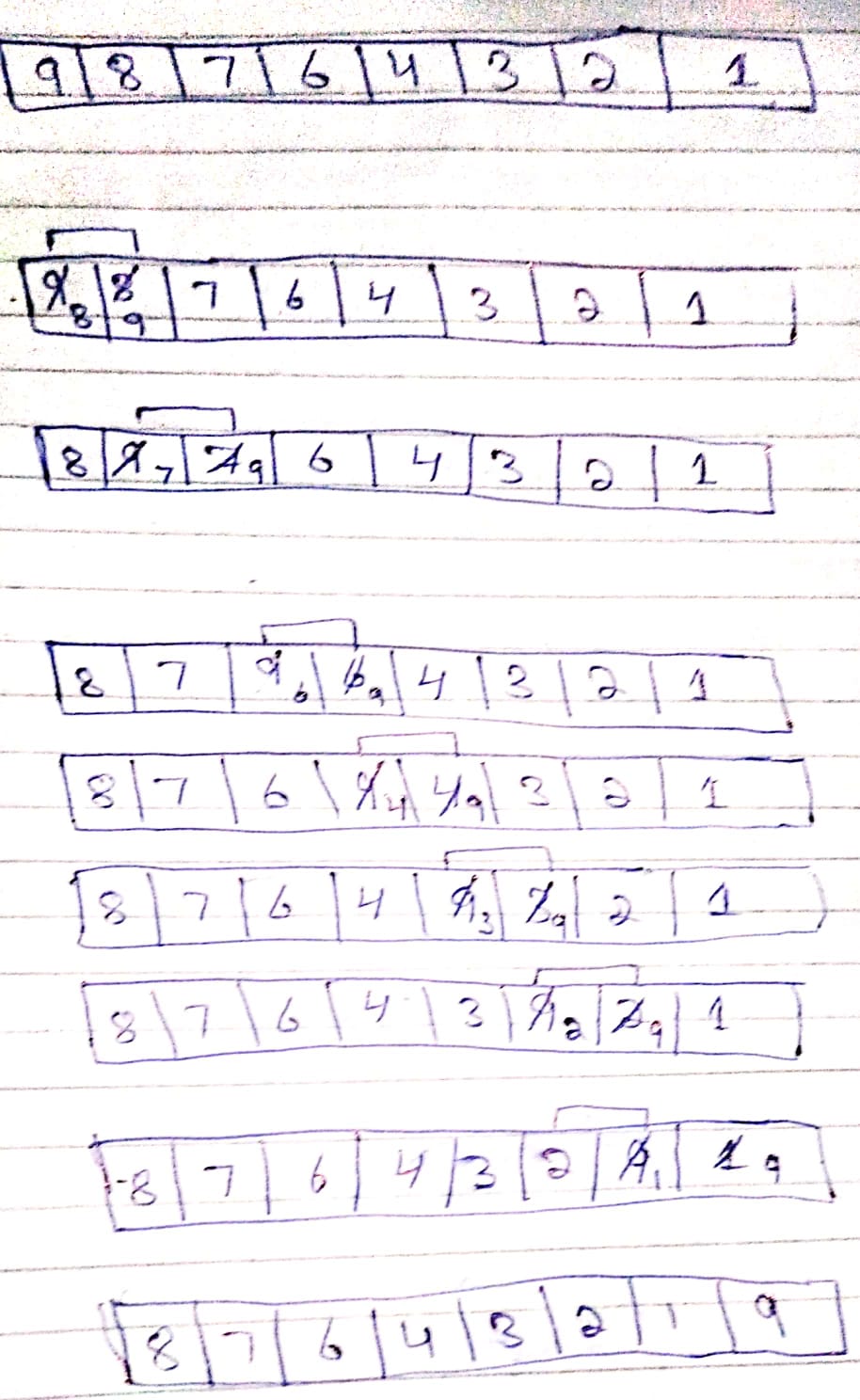
             if arr[j] > arr[j+1]:

                arr[j] = arr[j+1],

arr[j+1]= arr[j]

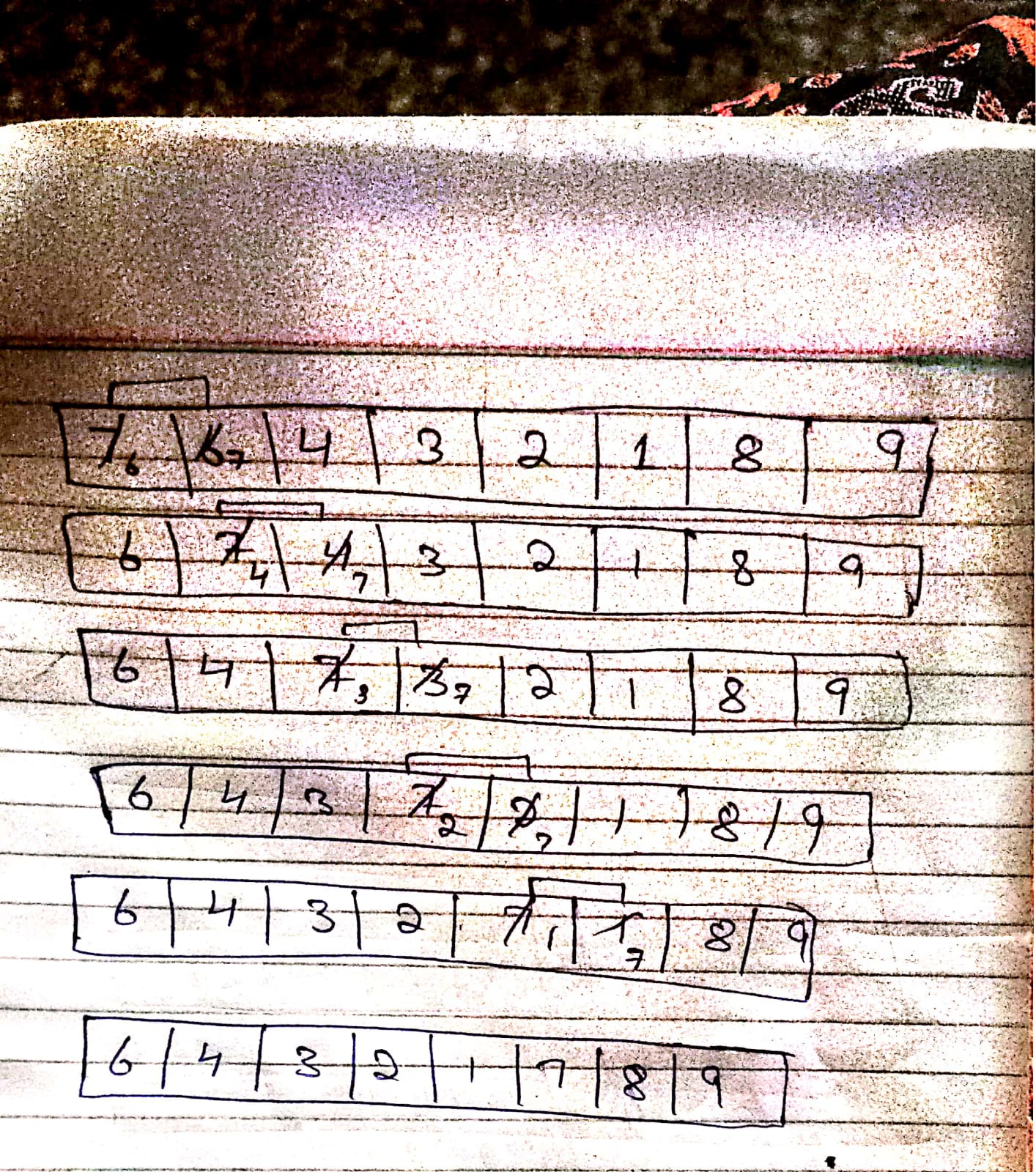
10. Run your algorithm on the following input.

A= {9,8,7,6,4,3,2,1}

 Pass 1:

Text, letter

Description automatically generatedPass 2:



Pass 3:

Pass 4 :

Text, letter

Description automatically generatedPass 5:

A picture containing text, document

Description automatically generated

Pass 6:

A picture containing text, document

Description automatically generated

Pass 7:

A picture containing text, stove, appliance, kitchen appliance

Description automatically generatedThus, Array is Sorted.