Digital Assignment-3.

Name:- Hari Krishna Shah.

VIT ID:- 21BCS0167.

1. Sort the elements 77, 49, 25, 12, 9, 33, 56, 81 using.

a) Insertion Sort.

Insertion sort algorithm places an unsorted element at its correct position in each iteration.

Algorithm:-

```
Void insertionsort(int array[], int size){
    for(int step = 1; step < size, step++){
        int key = array[step];
        int j = step - 1;
        while(key < array[j] && j >= 0){
            array[j+1] = array[j];
            --j;
        }
        array[j+1] = key;
    }
}.
```

=) Solution for given ~~array~~ array:-

Pass 1:

49 < 77 So Swap them.

=> | 49 | 77 | 25 | 12 | 9 | 33 | 56 | 81 |.

Pass:2

Again, 25 < 77 and 25 < 49. So, bring 25 before 49 and push other elements back.

=> | 25 | 49 | 77 | 12 | 9 | 33 | 56 | 81 |.

Pass:3 again, 12 < 77 and 12 < 49 and 12 < 25 So, bring 12 before 25 and push other elements one step back.

=> | 12 | 25 | 49 | 77 | 9 | 33 | 56 | 81 |.

pass:4. 9 < 77 and 9 < 49 and 9 < 25 and 9 < 12 so bring 9 before 12 and push other elements one step back.

=> | 9 | 12 | 25 | 49 | 77 | 33 | 56 | 81 |

pass:5 33 < 77 and 33 < 49 but 33 ✗ 25 So, bring 33 before 49 and push other elements back.

=> 9 12 25 33 49 77 56 81.

pass 6: 56 < 77 but 56 ✗ 49 So, bring 56 before 77 and push other elements one step back.

=> | 9 | 12 | 25 | 33 | 49 | 56 | 77 | 81 |.

pass 7: 81 ✗ 77 So, ~~no~~ Swap is not necessary.

Final sorted ~~arrg~~ array:-

=>

9	12	25	33	49	56	77	81

b) Selection Sort.

Selection sort is an sorting algorithm that sorts an array by repeatedly find minimum element of the array for each position.

Given, unsorted array:- 77, 49, 25, 12, 9, 33, 56, 81.

Iteration 1:

Fix the element for position 0 by repeatedly comparing first element i.e array[0] with other elements right of the array element at position 0. Swap, the number if array[0] is smaller than any other element.

~~After comparing 77 with all~~

After the iteration, we get sorted element for position 0.

i.e =>

9	49	25	12	77	33	56	81

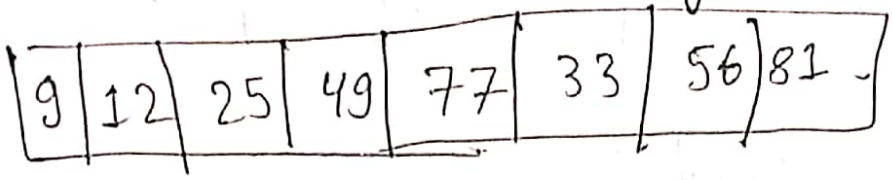
Iteration: 2.

Iteration 2 finds the correct element for positin 1 by comparing all the elements right of position 1 with position 1.

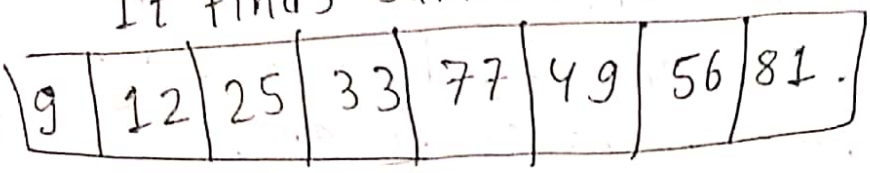
=>

9	12	25	49	77	33	56	81

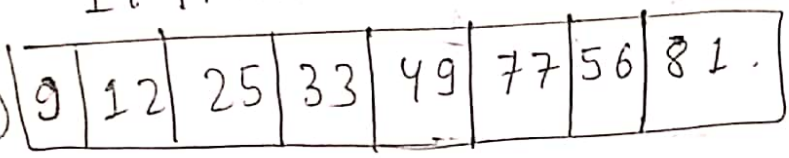
Iteration 3.
It finds suitable element for position 2 in ~~sinarl~~ similar way.

=> 

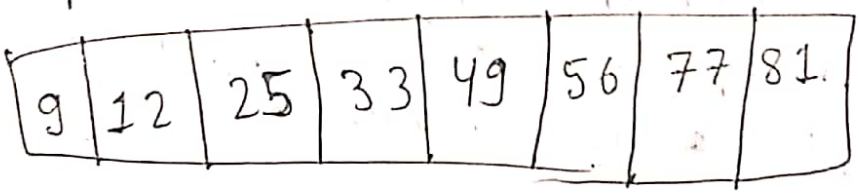
Iteration 4.
It finds suitable element for position 3.

=> 

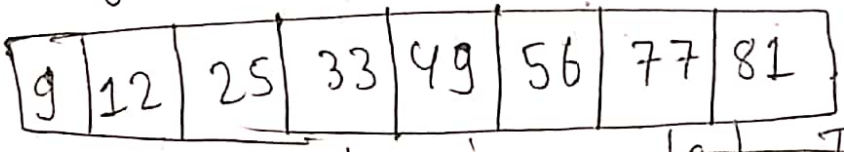
Iteration 5.
It finds suitable element for position 4.

=> 

Iteration 6.
It finds suitable element for position 5.

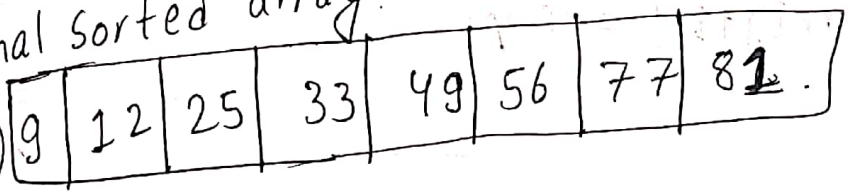
=> 

Iteration 7.
It finds suitable element for position 6.

=> 

No further iteration is required to sort the last element because it the last element can't be compared with right side element as it doesn't exist. Moreover, last element is sorted automatically at the end of selection sort.

Final sorted array:



Algorithm:

```
Void selectionsort (int array[], int size) {
    int temp;
    for(int i = 0; i<size-1; i++)
        for(int j = i+1; j<size; j++) {
            if(array[i] > array[j]) {
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
}
```

c) Bubble sort:

Bubble sort compares two adjacent elements and swaps them until they are not in the intended order.

Algorithm:

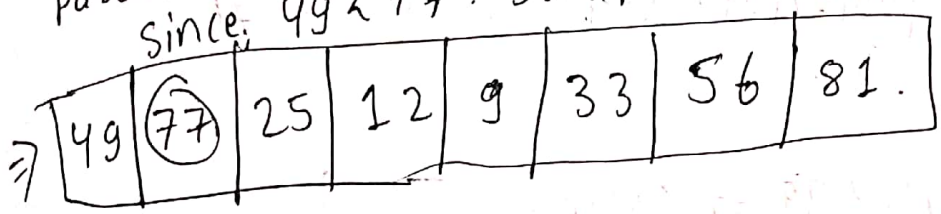
```
void bubblesort (int arr[], int n) {
    int i, j, temp;
    for(i=0; i<n-1; i++) {
        for(j=0; j<n-1-i; j++) {
            if(arr[j] > arr[j+1]) {
```

```
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
    }
}
}
```

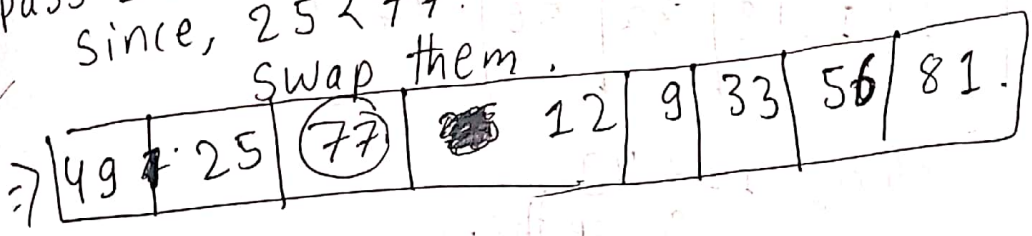
To Sort: 77, 49, 25, 12, 9, 33, 56, 81

Iteration 1:

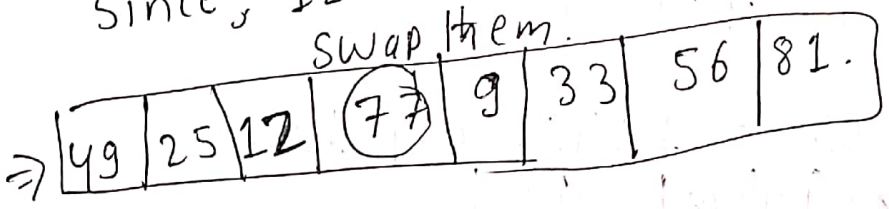
pass: 1.
since, 49 < 77. swap them.



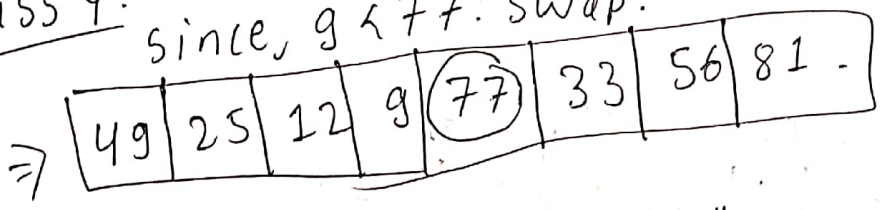
pass 2:
since, 25 < 77.
swap them.



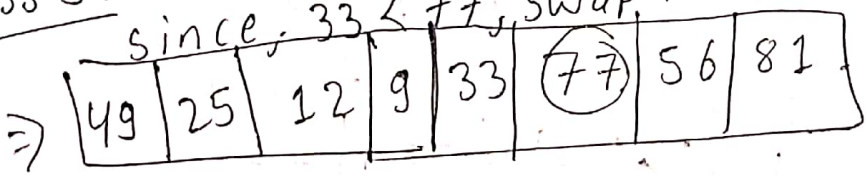
pass 3:
since, 12 < 77
swap them.



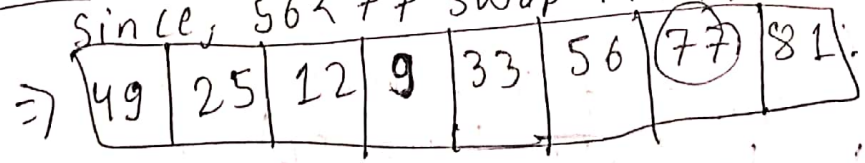
pass 4:
since, 9 < 77. swap.



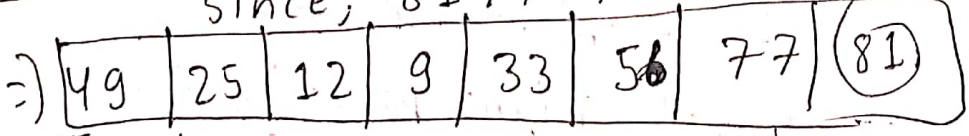
pass 5:
since, 33 < 77, swap them.



pass 6:
since, 56 < 77 swap them.



pass 7:
since, 81 ⩾ 77. No swap needed.



The last element i.e 81 is now sorted at right position i.e 7.

Now,
Again, doing the same process for elements from 0 to n-i-1 i.e 8-1-1
= 8-2
= 6 #

we have, Array

25	49	12	9	33	56	77

Iteration 2.

Pass 1:
25 < 49 So swap.

=>

25	(49)	12	9	33	56	77

Pass 2:
since, 12 < 49
Swap them.

=>

25	12	(49)	9	33	56	77

Pass 3:
since 9 < 49, Swap them.

=>

25	12	9	(49)	33	56	77

Pass 4:
since, 33 < 49, Swap.

25	12	9	33	(49)	56	77

Pass 5:
since, 56 7 49, don't swap.

25	12	9	33	49	(56)	77

Pass 6:
since, 77 7 56, don't swap.

25	12	9	33	49	56	(77)

The last element i.e. 77 is Sorted at it's right position i.e 6.

Iteration 3

Repeat process from index 0 to 5
i.e $n-1-i$
$= 8-1-2$
$= 8-3$
$= 5$.

we have,

25	12	9	33	49	56

pass: 1

Since, $12 < 25$.
swap.

=>

12	(25)	9	33	49	56

Pass: 2

since, $9 < 25$, swap.

12	9	(25)	33	49	56

pass: 3

since, $33 > 25$ no swap needed.

12	9	25	(33)	49	56

Pass: 4

since, $49 > 33$, don't swap.

=>

12	9	25	33	(49)	56

pass: 5

since, $56 > 49$, don't swap.

12	9	25	33	49	(56)

The element for position 5 is now sorted correctly as 56.

Iteration 4

Repeat ~~for~~ process from index 0 to n-1-i;
i.e from 0 to 8-1-3
= 8-4
= 4.

we have,

=> | 12 | 9 | 25 | 33 | 49 |

Pass 1:

since, 9 < 12 swap.

| 9 | (12) | 25 | 33 | 49 |

pass 2:

since, 25 > 12, don't swap.

| 9 | 12 | (25) | 33 | 49 |

pass 3:

since, 33 > 25, don't swap.

| 9 | 12 | 25 | (33) | 49 |

pass 4:

since, 49 > 33, don't swap.

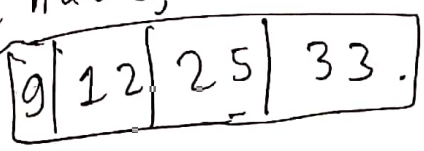
| 9 | 12 | 25 | 33 | (49) |

The correct element for position 4 is now sorted ~~for~~ as 49.

Iteration 5.

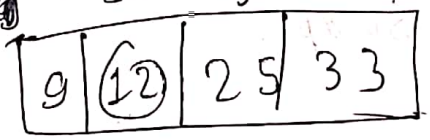
Repeat process from index 0 to 3.

We have,

=> 

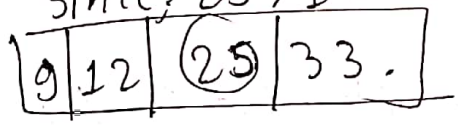
Pass 1:

. Since, 1279, don't swap.

=> 

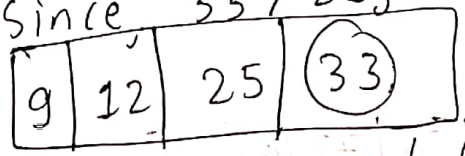
pass: 2

Since, 25712 don't swap.

=> 

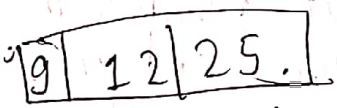
pass: 3

Since, 33725, don't swap.

=> 

The correct element for position 3 is now sorted as 33.

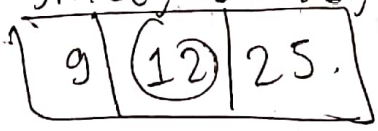
Iteration 6:

we have, 

Repeat process from index 0 to 2.

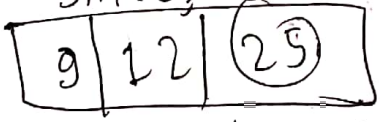
Pass 1

Since, 1279, don't swap.



pass: 2

Since, 25712, don't swap.



The correct element for position 2 is now sorted as 25.

Iteration 7.

We have,

=> | 9 | 12 |

Repeat process from 0 to 1.

Pass 1 Since, 12 > 9, don't Swap.

=> | 9 | (12) |

The correct element for position 1 is sorted as 12.

The ~~remaing~~ remaining element i.e 9 is sorted as correct element for position 0.

Final sorted array

=> | 9 | 12 | 25 | 33 | 49 | 56 | 77 | 81 |

d) Merge Sort.

Merge sort is a divide and conquer algorithm that divides the array into two halves and merges them while sorting simultaneously.

C code for Merge sort.

```
void merge(int arr[], int l, int m, int r){
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
```

```
for (i = 0; i < n1; i++) {
    L[i] = arr[l+i];
}
for (j = 0; j < n2; j++) {
    R[j] = arr[m+1+j];
}
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
```

```
while(j<n2){
    arr[k]=R[j];
    j++;
    k++;
}
}
void mergesort(int arr[], int l, int r){
    if(l<r){
        int m=l+(r-1)/2;
        merge(arr, l, m, r);
    }
}
```

To Sort:-

0	1	2	3	4	5	6	7
77	49	25	12	9	33	56	81

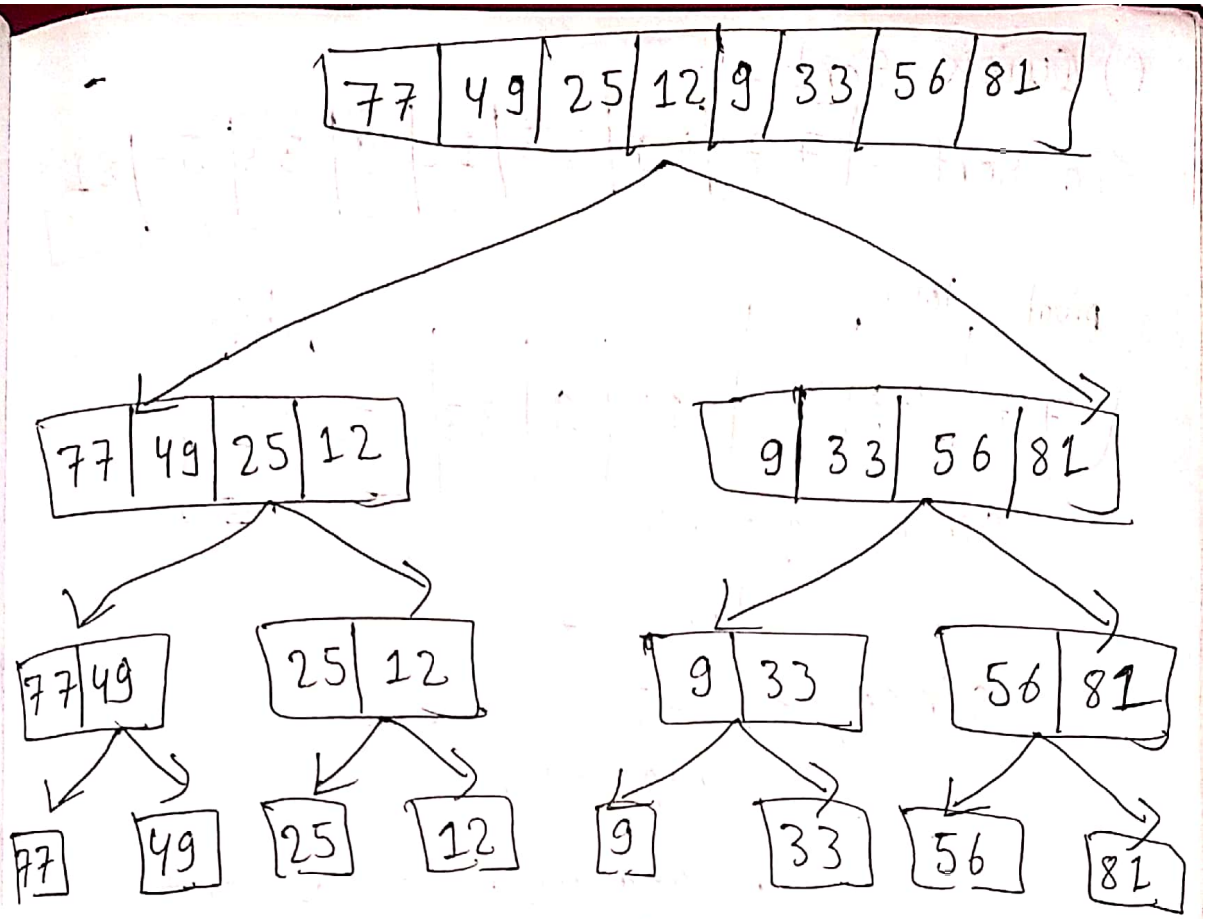
Dividing the array into halves using the formula $m = \frac{\text{Left index} + \text{right index}}{2}$

to divide the array into halves: Divide the array until each sub-array contains only 1 element.

Here, for 1st half,

$$m = \frac{0+7}{2} = 3.5.$$

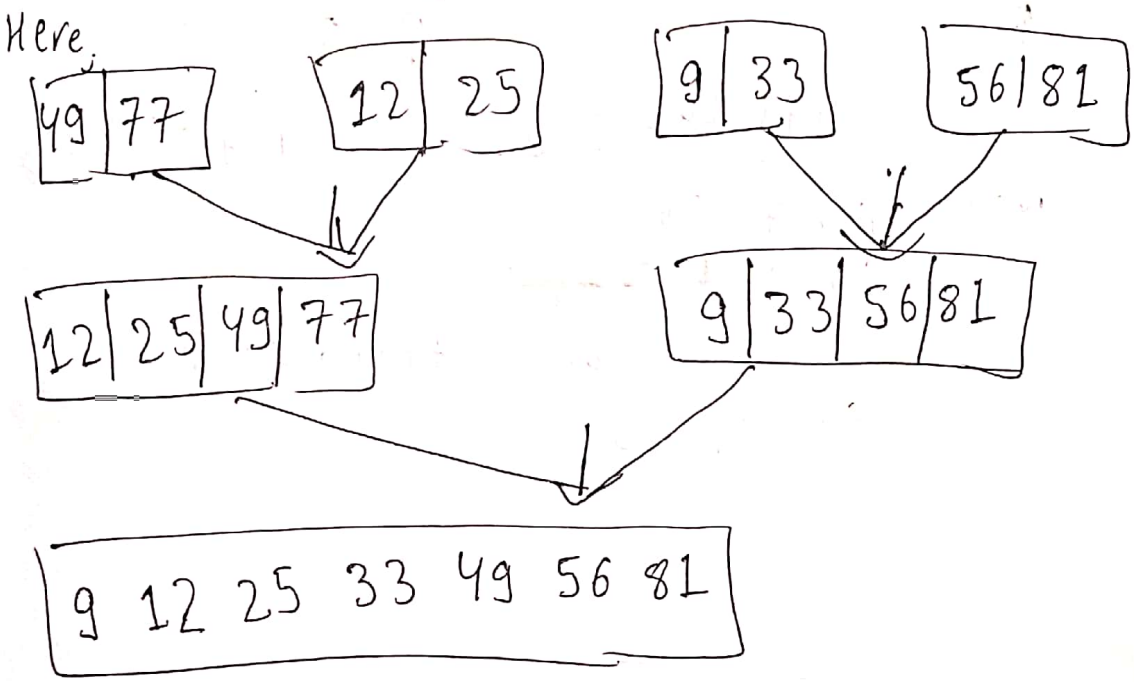
So, 3 is the middle after flooring



Conquer

Compare the Sub-array and combine them by putting the smaller sub-array element in front.

Here,



e) Quick Sort.

To sort:-

77	49	25	12	9	33	56	81

pivot	low						high
77	49	25	12	9	33	56	81

Since, 81 > 77.

low = 81.

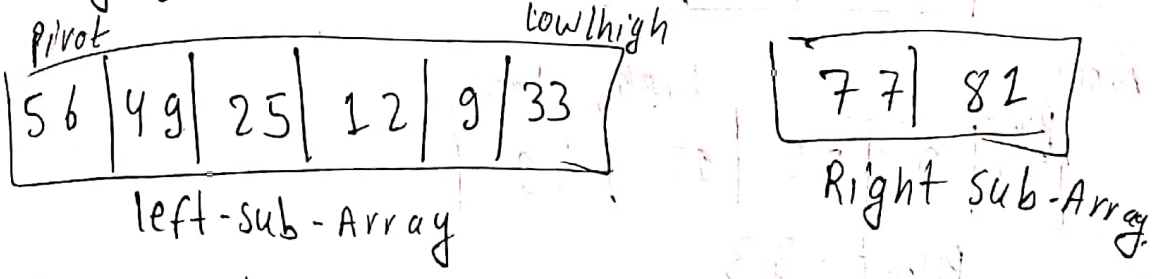
Since, 56 < 77.

high = 56,

Since, low and high are adjacent

Swap pivot and high.

we get,



For left-sub-Array.

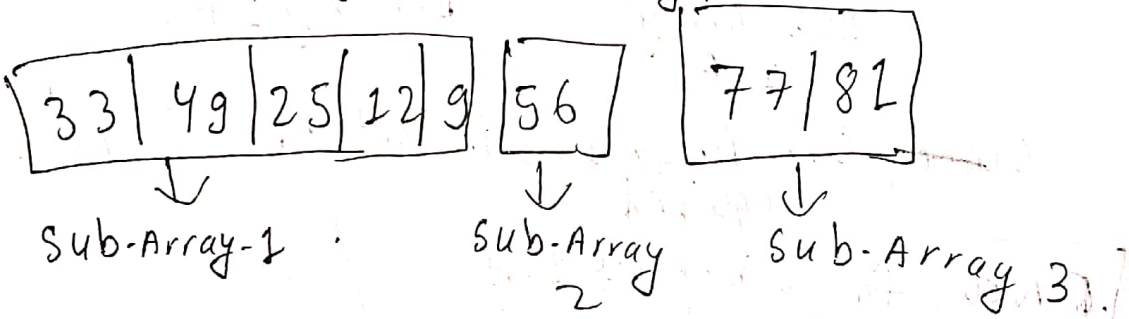
pivot = 56.

Since, there is no element greater than 56.

low = 33

high = 33.

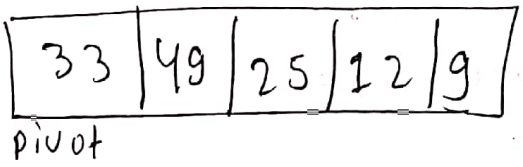
since, low and high are crossing swap pivot and high.



Sub-Array 3 is already sorted.

Now, for Sub-Array 1

pivot = 33.



since, 49 ≥ 33

low = 49

since, 9 < 33

high = 9.

Since, low and high not crossing exchang low and high position.

33	9	25	12	49

Now

pivot			high	low
33	9	25	12	49

pivot = 33
low = 49 since, 49 > 33
high = 12 since, 12 < 33

since, low and high crossing
exchange high
and pivot position.

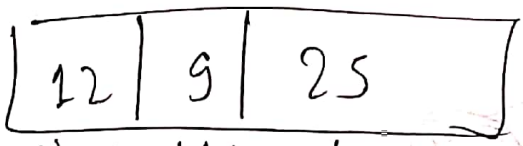
12	9	25	33	49

Sub-Array 1.A

Sub-Array 1.B.

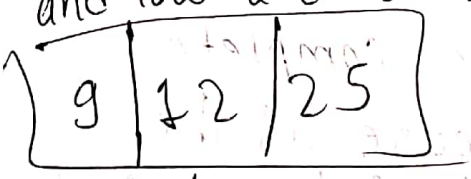
Since, Sub-Array 1.B is sorted, no Comparison required.

Now, for Sub-Array 1.A.



Pivot = 12, low = 25 since 25 > 12 and low = 9, since, 9 < 12.

Now, swap Pivot and high since high and low are crossing

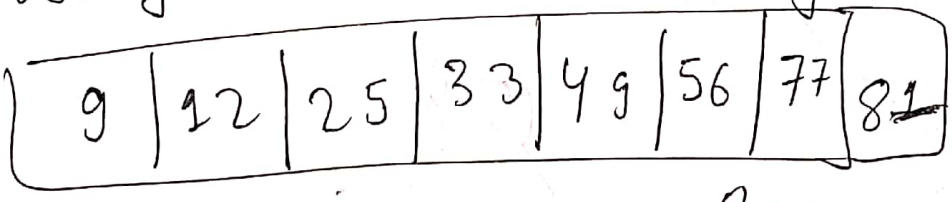


Sub-Array 1.A

Hence, all sub-arrays are now sorted successfully.

combine all sub-arrays to get the final sorted array.

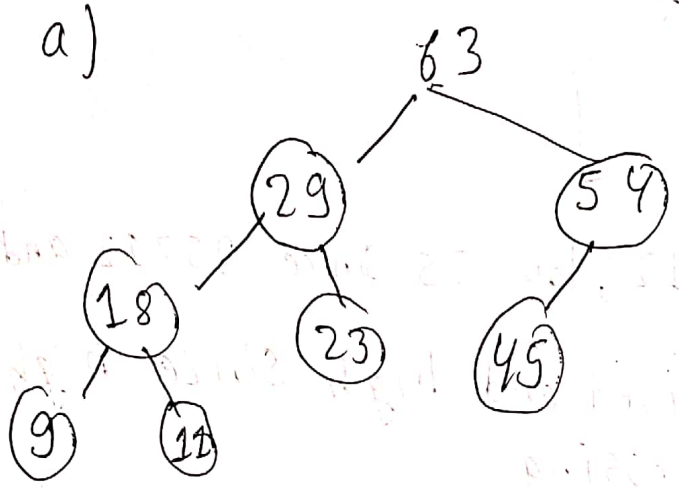
We get final sorted array:-



Answer

Q2.

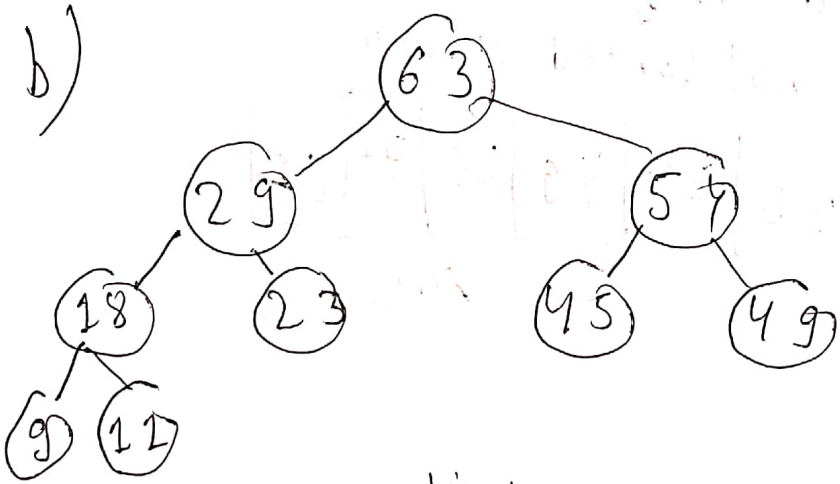
a)



The given binary tree is a complete binary tree because all the levels are completely filled except the lowest level i.e 54 has only one child

The given binary tree isn't a full binary tree because the element 54 has only one child.

b)



The given binary tree is both complete and full binary tree because all the levels are completely filled with two children.