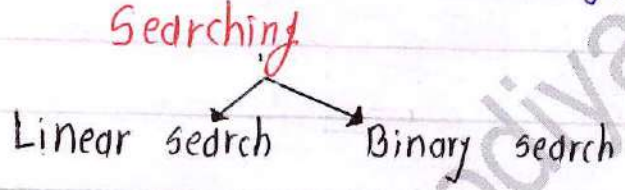
2.1 Searching :

i) Searching is a fundamental operation which is perform on the Data structure.

ii) Searching is the process of finding the location of a given data element.

iii) Searching is classified in two types.



- **Linear search :**

Linear search is also known as sequential search. This method traverses a list sequentially to locate a key element or key value.

Algorithm for linear search.

Step 1: Set Flag = 0

Step 2: Set index = 0

Step 3: Begin from index at the first record to the end of the list & if the required record is found make Flag = 1

Step 4: At the end, if Flag is 1 the record is found otherwise the search is failure/ unsecessful.

Ex:
(1) Search the key value = 14 in the given sequence by using linear search.

0	1	2	3	4	5
6	1	3	14	20	19

for i = 0
value = key element
6 ≠ 14

For i = 1
value = key element
1 ≠ 14

for i = 2
value = key element
3 ≠ 14

for i = 3
value = key element
14 = 14

The search key value is found at index location = 3
Search is successful.

Example:

(2) Consider the array arr[] = {10, 50, 30, 70, 80, 20, 90, 40} and key = 30.

⇒ The given array is

0	1	2	3	4	5	6	7
10	50	30	70	80	20	90	40

For i=0,
Value = key element
10 ≠ 30

For i=1,
Value = key element
50 ≠ 30

For i=2,
Value = key element
30 = 30

The search key value is found at index location = 2
Search is successful

Advantages of linear search

1) It is a simple and easy method of searching.
2) It can be applied over an sorted / unsorted array.
3) It is highly efficient for small list.

Disadvantages of linear search

1) It is high inefficient for large data / sequence
2) High Time complexity.

Binary Search :

1) Binary search is a very fast searching technique.
2) It can be applied only on sorted list / array.

How binary search works

In this method first calculate the mid position in an array and compare the mid position element with the search element.

If a match is found, the complete search otherwise divide the input list into two parts.

First part contains all the numbers less than mid element.

And second part contains all the numbers greater than mid element.

To calculate mid element perform

$$mid = \frac{(lower + upper)}{2}$$

The binary search performs the comparison till the element is found.

Algorithm :

step 1: Define start & end.
step 2: Find middle, mid = $\left[\frac{start + end}{2}\right]$
step 3: A] IF A [mid] < element, search to the right of the middle element.

→ start = mid + 1 & end = end.

B] If A [mid] element search to the left of the middle element.

→ start = start & end = mid - 1

C] If A [mid] = element, element / search element found return mid
search successful.

Example :

1) Find the position of element 88 using binary search method in an sequence given below.

arr [] = { 77, 33, 44, 11, 88, 22, 66, 55 }

Given sequence is

	0	1	2	3	4	5	6	7
start →	11	22	33	44	55	66	77	88

→ end

$$mid = \frac{start + end}{2}$$

$$= \frac{0+7}{2}$$

$$= 3.5$$

floor / base = 3

∴ mid = 3

A [mid] < element

44 < 88

Move towards right side of array

Start = 3+1 = 4 End = End End = 7

Start = 4 ↓ start, End

0	1	2	3	4	5	6	7
11	22	33	44	55	66	77	88

$mid = \frac{start + end}{2}$

$= \frac{4+7}{2}$

$= 5.5$

mid = 5

A [mid] < element

66 < 88

Move toward right side of array

start = mid + 1 = 5 + 1 = 6 & end = end, end = 7

start, End

0	1	2	3	4	5	6	7
11	22	33	44	55	66	77	88

$mid = \frac{start + End}{2} = \frac{6+7}{2} = 6.5$

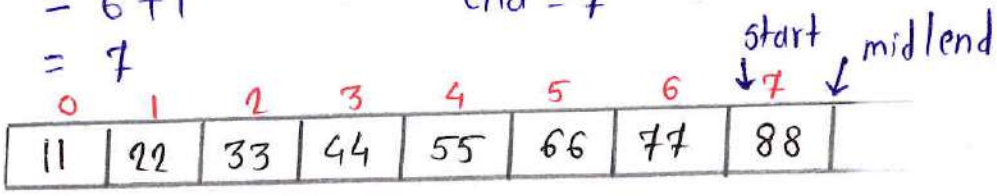
mid = 6

A [mid] < element

77 < 88

Move towards right side of array

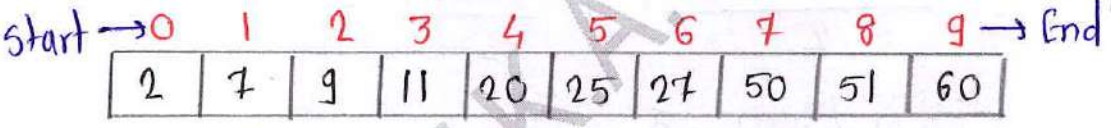
start = mid + 1 & end = end
= 6 + 1 end = 7
= 7



A [mid] = element
88 = 88
∴ Search is found

2) Using Binary search, find element 25 in a given sequence below.
11, 20, 2, 7, 9, 50, 27, 25, 51, 60

⇒ Sorted array is :

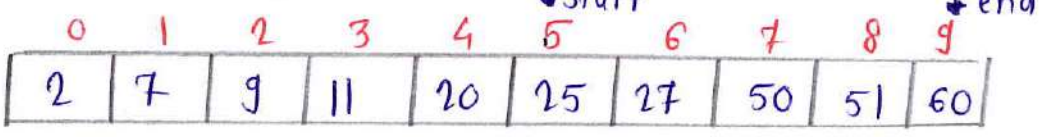


$mid = \frac{start + end}{2} = \frac{0+9}{2} = 4.5$

Floor/base = 4

∴ mid = 4

A [mid] < element
20 < 25
Move towards right side of array
start = mid + 1 End = end
= 5 = 9



$mid = \frac{start + end}{2} = \frac{5+9}{2} = 7$

$\therefore mid = 7$

A mid < element

50 < 25

Move towards left side of array from mid

start = start end = mid - 1

start = 5 = 7 - 1 = 6

0	1	2	3	4	5	6	7	8	9
2	7	9	11	20	25	27	50	51	60

$mid = \frac{start + end}{2} = \frac{5+6}{2} = 5.5$

Floor / base := 5

$\therefore mid = 5$

A [mid] Search

25 = 25

Element is found at index location 5, so search is successful.

2.2 Sorting :

- Sorting is a technique which is used to rearrange the elements of a list in ascending or descending order.
- A sorting Algorithm rearranges elements of an array or list in a specific order.
- The order can be :
  - Ascending order → Smallest to largest
    Example [10, 20, 5, 2] → [2, 5, 10, 20]
  - Descending Order → largest to smallest
    Example [10, 20, 5, 2] → [20, 10, 5, 2]

Sorting is of two types
1) Internal sort
2) External sort

Internal sort :

- Internal sorting is when all the data is placed in the main memory or internal memory. In internal sorting, the problem cannot take input beyond allocated memory size.
- In this method only the primary memory, main memory, RAM is used for processing.

External sort:

External sorting is when all the data that needs to be sorted not to be placed in memory at a time,

the sorting is called external sorting. External sorting is used for the massive amount of data.
For Example Marge sort can be used in external sorting as the whole array does not have to be present all the time in memory.
- In this method there is need of external secondary storage for performing sorting operation.

1) Bubble sort :

Bubble sort is very simple method that sorts the array elements by repeatedly moving the largest element to the highest index position of the array segment.

If the element at the lower index is greater than the element at the higher index, then two elements are interchanged, swapped.
So that the element is placed before the bigger one.

Algorithm for bubble sort :

1) Compare two adjacent element
2) If necessary, swap (exchange) them
   i) Sorting begins with 0^{th} element & it compares with 0^{th} element and it compares with 1^{st} element in list.

ii) If 1st element is less than 0th elements, then interchange take place.

iii) Like this all elements are compared with next element and swap it.

iv) At the end of 1st pass largest element is placed at last position.

v) In 2nd pass again comparisons starts with 0th element and 2nd largest element is placed at second last position.

vi) This process continues till list is in sorted order.

Sort given array in ascending order using bubble sort.

A = { 19, 2, 27, 3, 7, 5, 31 }

⇒ Here given array is

0	1	2	3	4	5	6
19	2	27	3	7	5	31

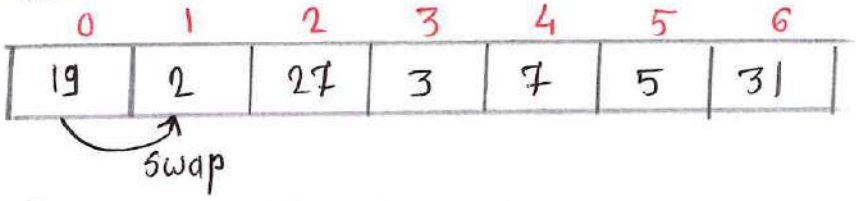
N = Total no. of element in array

N = 7

passes = N-1 = 7-1 = 6

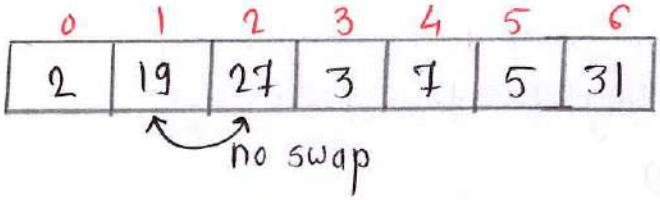
∴ passes = 6

pass - I



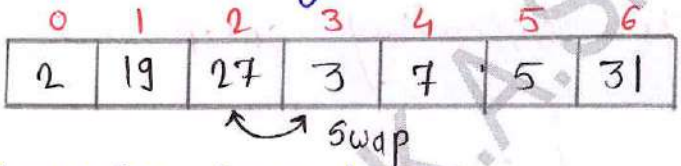
Comparing A[0] & A[1]

Swapping 19 & 2



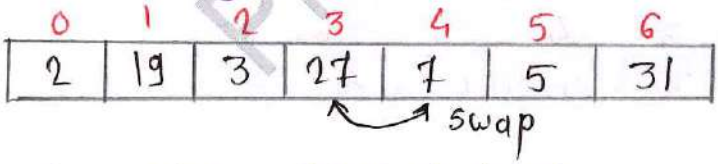
Comparing A[1] & A[2]

No swapping



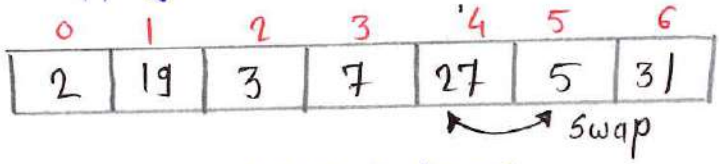
Comparing A[2] & A[3]

swapping



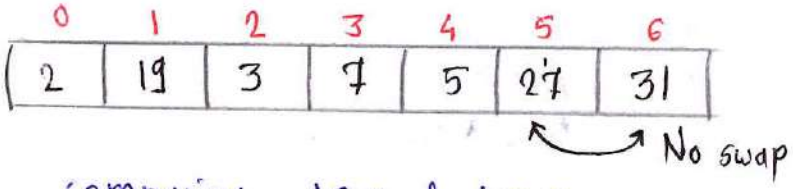
Comparing A[3] & A[4]

Swapping element



Comparing A[4] & A[5]

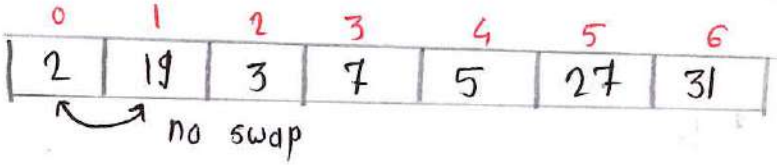
Swapping element



comparing A[5] & A[6]

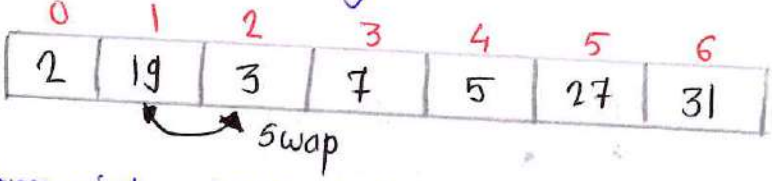
so no swap

pass - II :



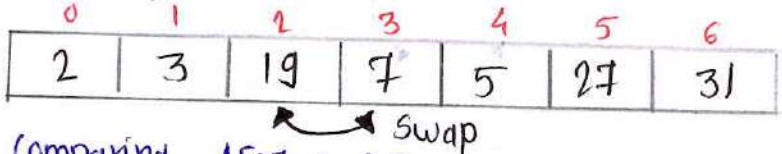
Comparing A[0] & A[1]

No swapping



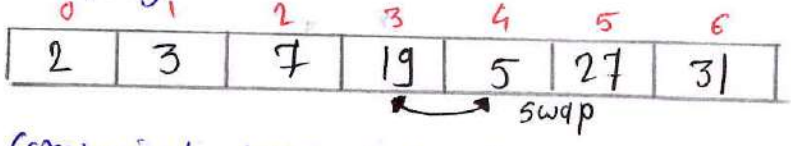
Comparing A[1] & A[2]

So swap elements



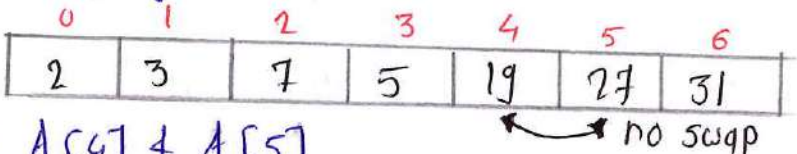
comparing A[2] & A[3]

swapping



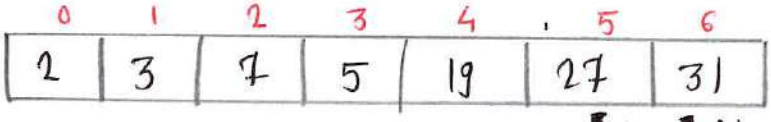
comparing A[3] & A[4]

swapping 19 & 5



Comparing A[4] & A[5]

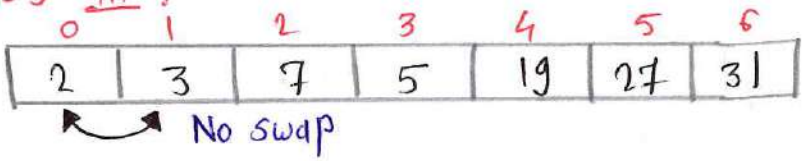
No swapping.

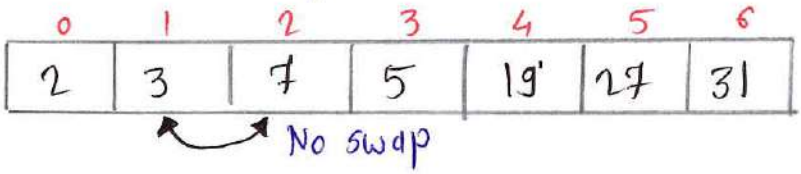


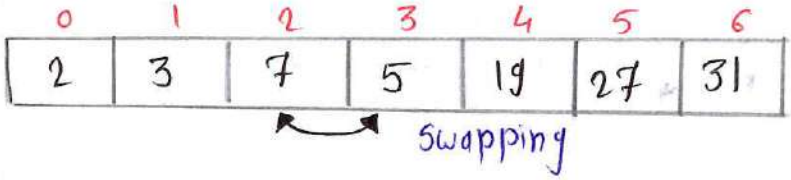
Comparing A[5] & A[6]

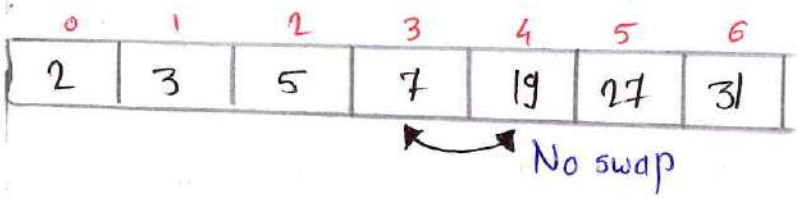
No swapping.

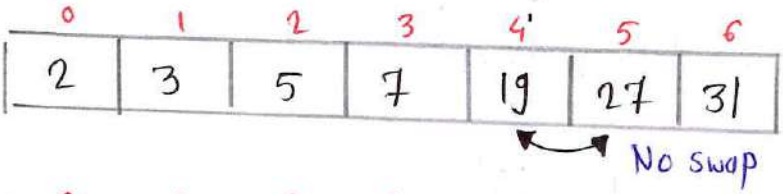
Pass - III :

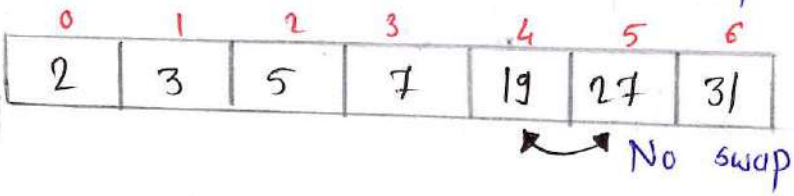


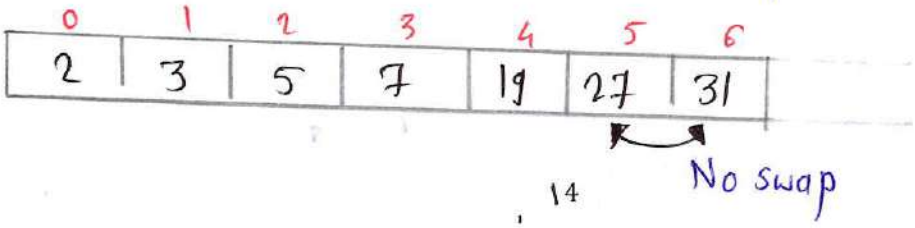




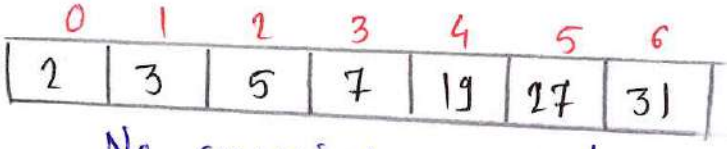






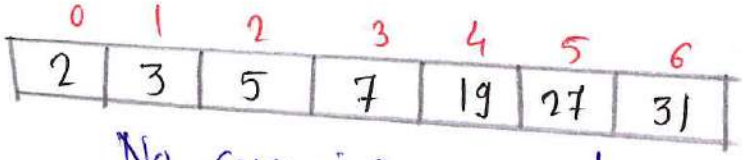


pass - IV



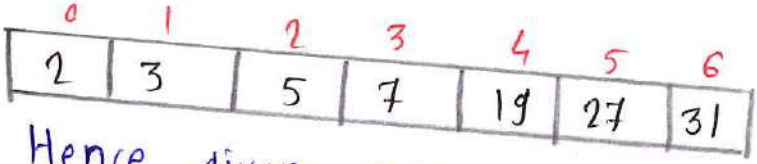
No swapping required

pass - V



No swapping required

pass - VI



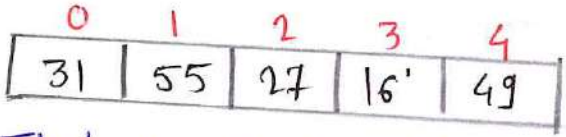
Hence, given sequence or array is sorted in ascending order.

---

2) Sort the given array / sequence using bubble sort.

{ 31, 55, 27, 16, 49 }

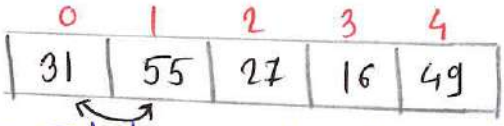
⇒ Here given array is



N = Total no. of element in array = 5

passes = N-1 = 5-1 = 4

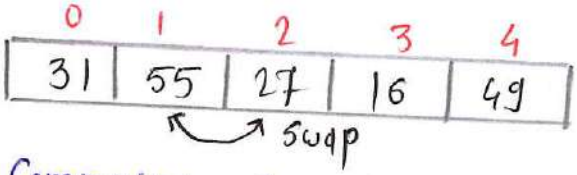
pass - I



N = Total no. of element. in

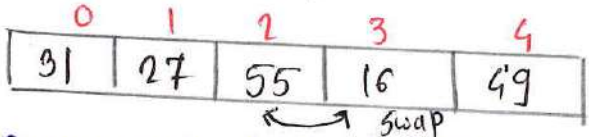
Comparing A[0] & A[1]

No swapping



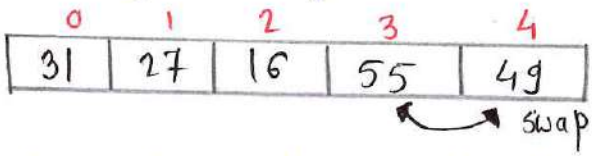
Comparing A[1] & A[2]

Swapping 55 & 27



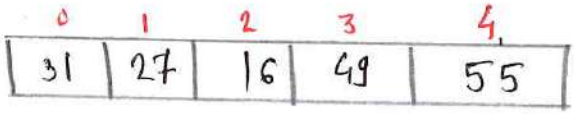
Comparing A[2] & A[3]

Swapping 55 & 16

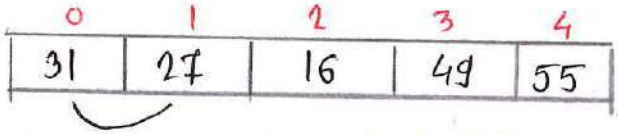


Comparing A[3] & A[4]

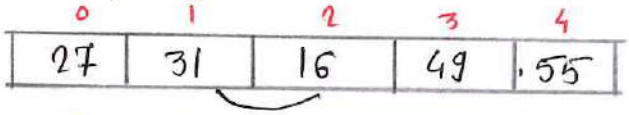
Swapping 55 & 49



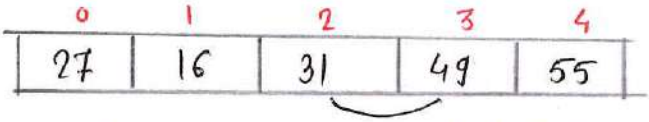
pass II :



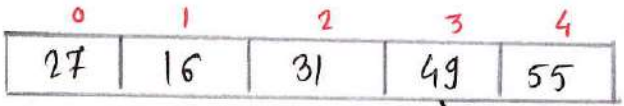
Comparing A[0] & A[1]
Swapping 31 & 27



Comparing A[1] & A[2]
Swapping 31 & 16

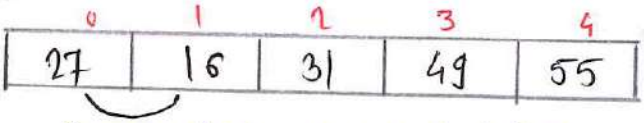


Comparing A[2] & A[3]
No swap

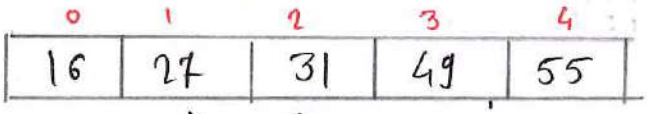


Comparing A[3] & A[4]
No swap

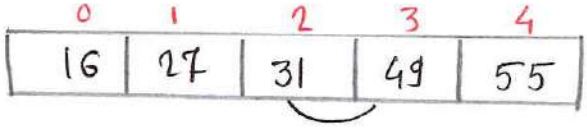
pass III :



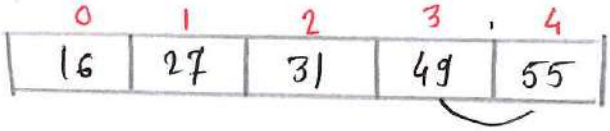
Comparing A[0] & A[1]
Swapping 27 & 16



Comparing A[1] & A[2]
No swapping

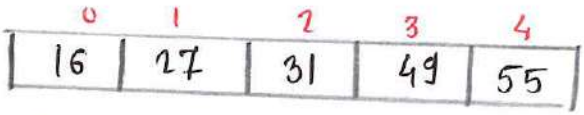


Comparing A[2] & A[3]
No swapping



Comparing A[3] & A[4]
No swapping.

Pass - IV



No swapping required.

Hence, given sequence or array is sorted in ascending order.

Advantages of bubble sort

1) Bubble sort is easy to understand and implement.
2) It does not require any additional memory space.

Disadvantages of bubble sort

1) It require more time for sorting.

2) Selection Sort :

Selection sort is a comparison-based sorting algorithm. It sorts by repeatedly selecting the smallest (or largest) element from the unsorted portion & swapping it with the first unsorted element.

1. Find the smallest element & swap it with the first element. This way get the smallest element at its correct position.
2. Then find the smallest among remaining elements (or second smallest) & swap it with the second element.
3. We keep doing this until we get all elements moved to correct position.

Algorithm :

Step 1 : Set MIN to location 0.

Step 2 : Find the minimum value in the list or search.

Step 3 : Swap with value at location MIN or first position.

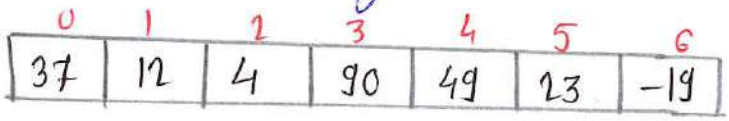
Step 4 : Increment MIN to point to next location.

Step 5 : Repeat step. till array is sorted.

Perform selection sort over given list of an array.

A = { 37, 12, 4, 90, 49, 23, -19 }

⟹ Here given array is

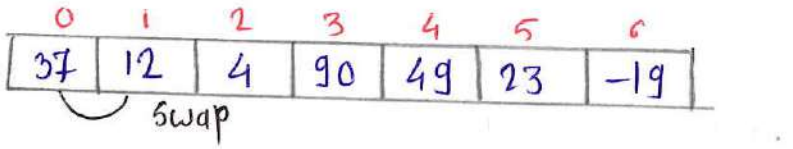


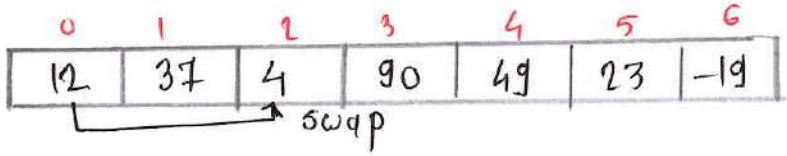
N = 7

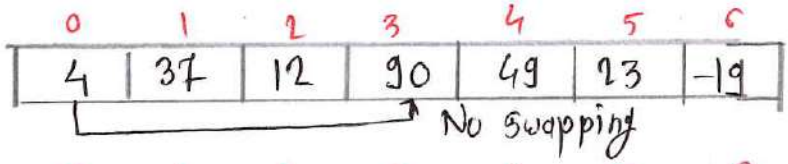
passes = N-1 = 7-1 = 6

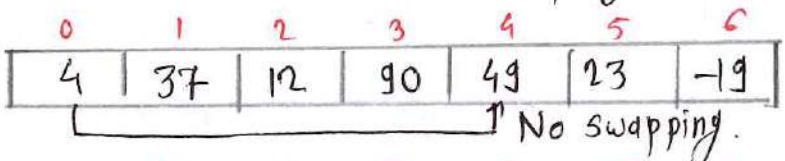
∴ pass = 6

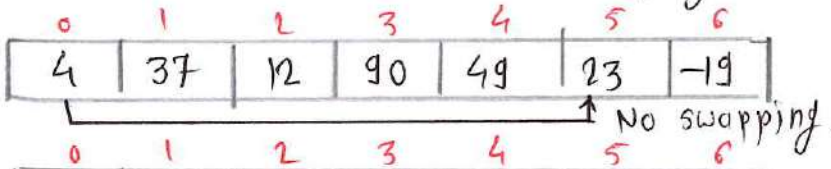
pass I : MIN = 0

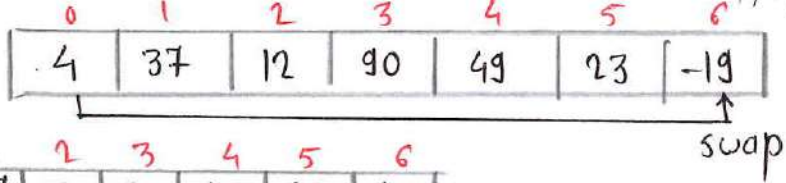


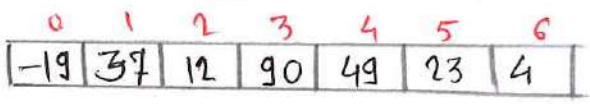




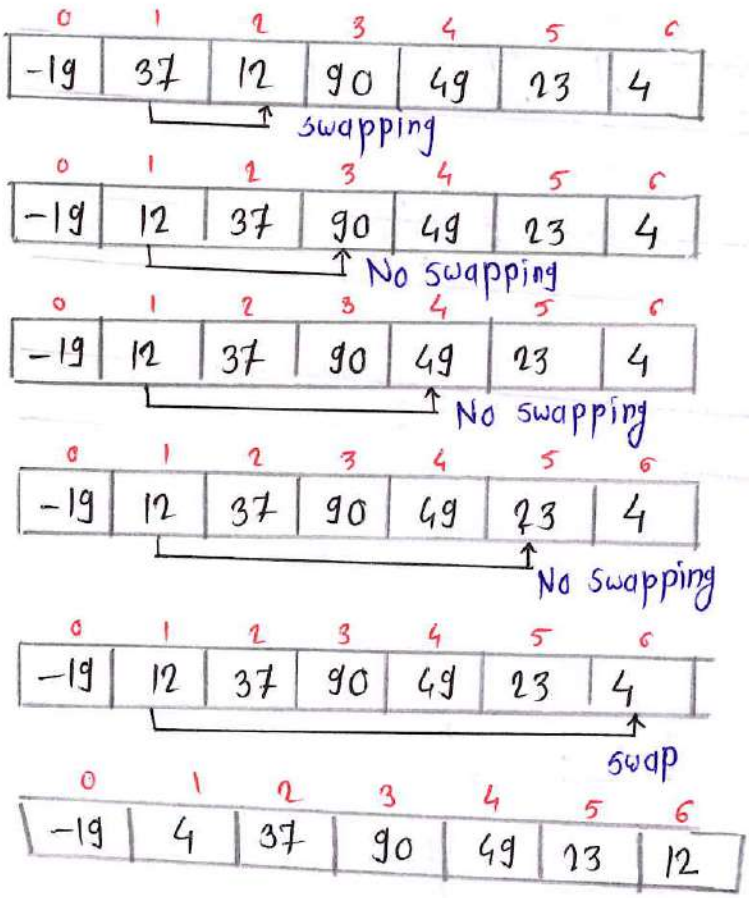




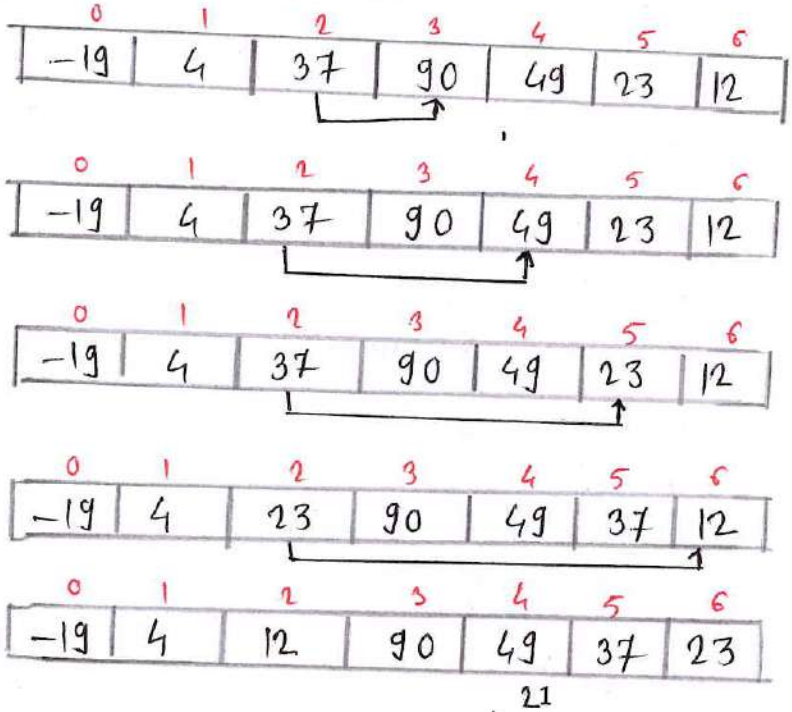




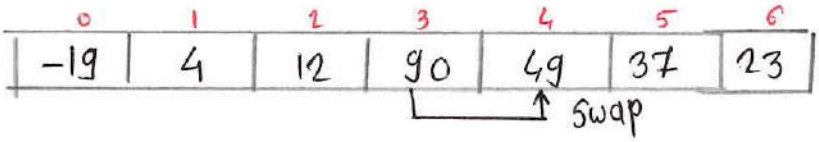
Pass - II : MIN = 1

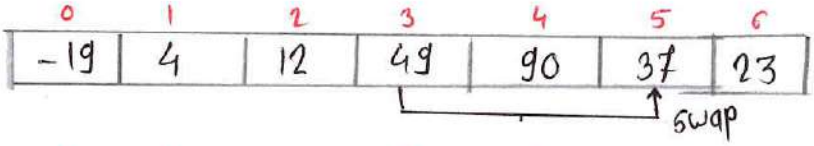


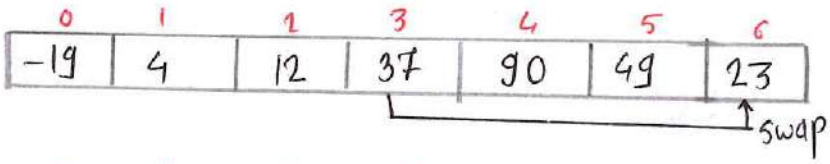
Pass - III : MIN = 2



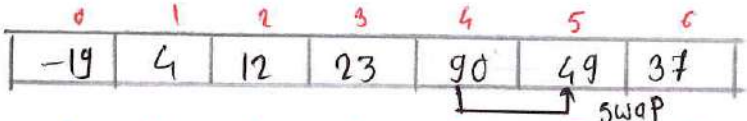
pass - IV MIN = 3

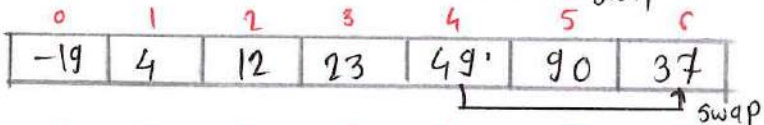


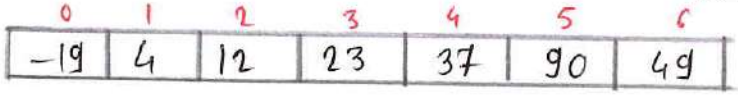




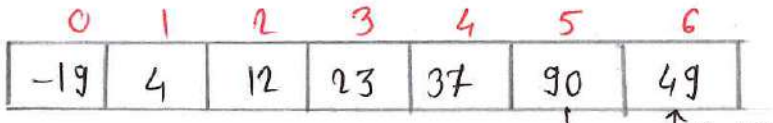
pass - V MIN = 4

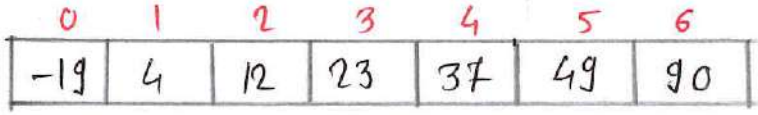






pass - VI MIN = 5



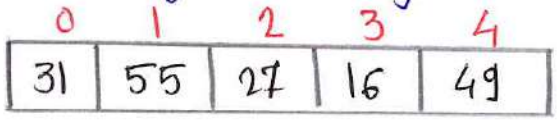


Hence, given sequence of array is sorted in ascending order.

* Sort the given sequence in ascending order using selection sort.

A = {31, 55, 27, 16, 49}

⟹ Here given array is

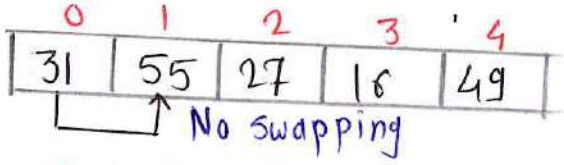


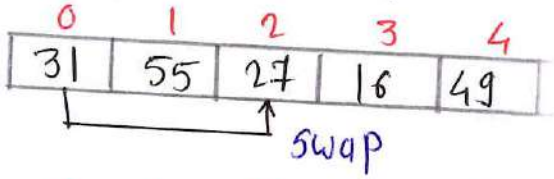
N = 5

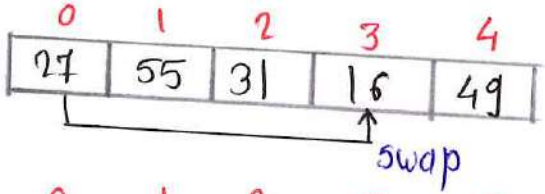
pass = N-1 = 5-1 = 4

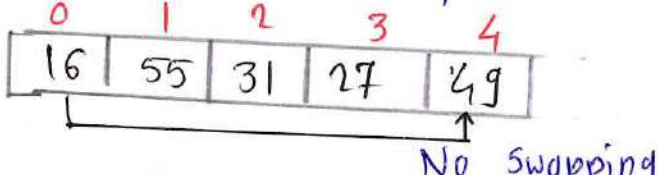
∴ pass = 4

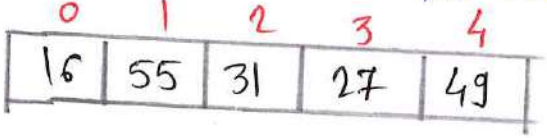
pass-I : MIN = 0



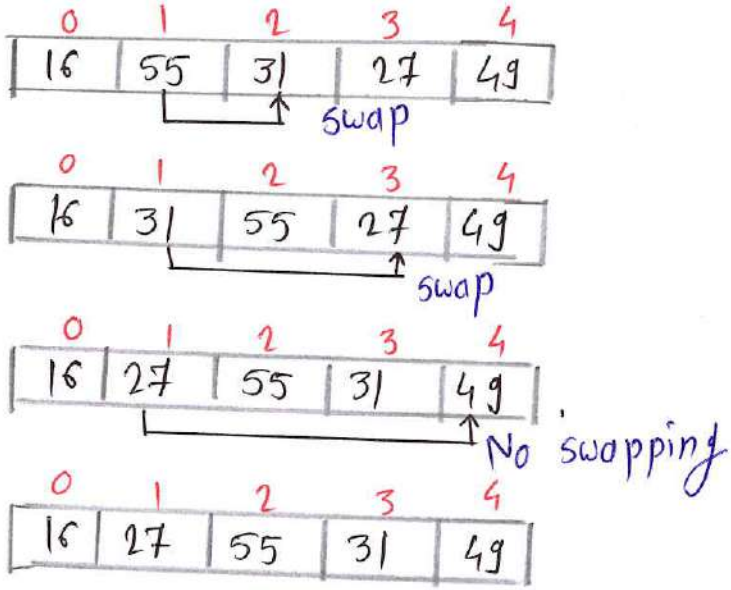




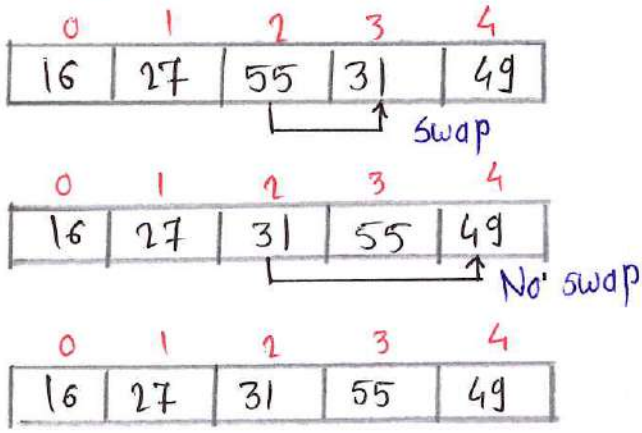




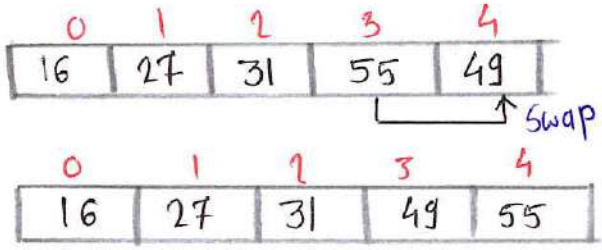
pass - II : MIN = 1



pass - III : MIN = 2



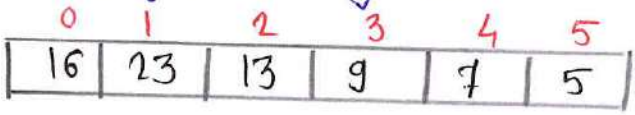
pass - IV : MIN = 3



Hence, given sequence of array is sorted in ascending order.

* {16, 23, 13, 9, 7, 5}

⇒ Here given array is

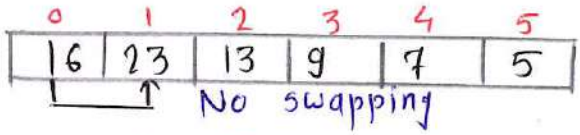


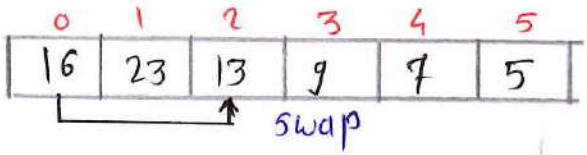
N = 6

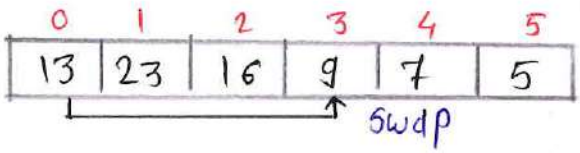
pass = N-1 = 6-1 = 5

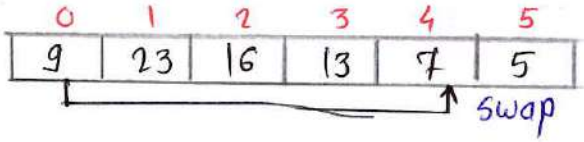
∴ pass = 6

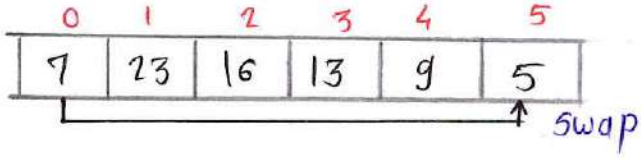
Pass-I : MIN = 0

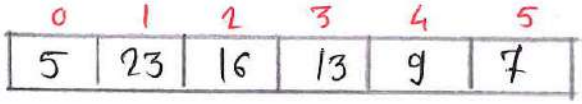




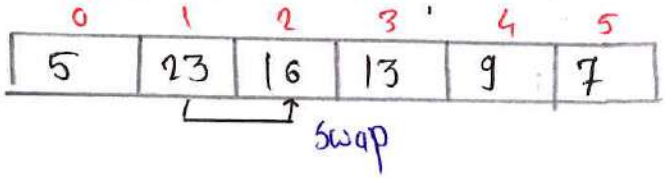


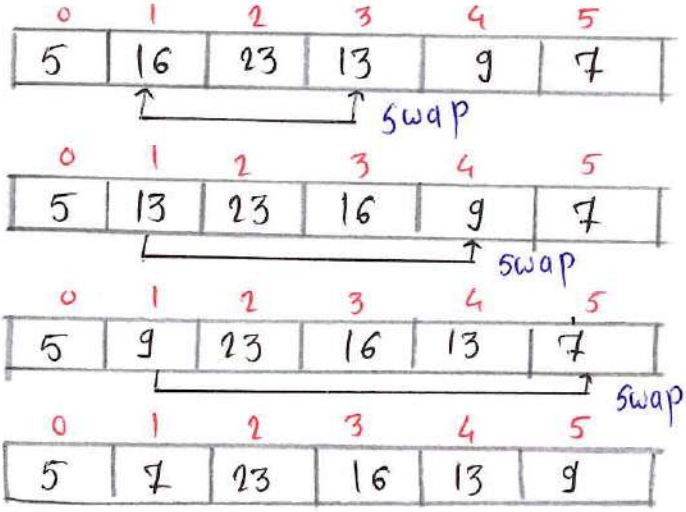




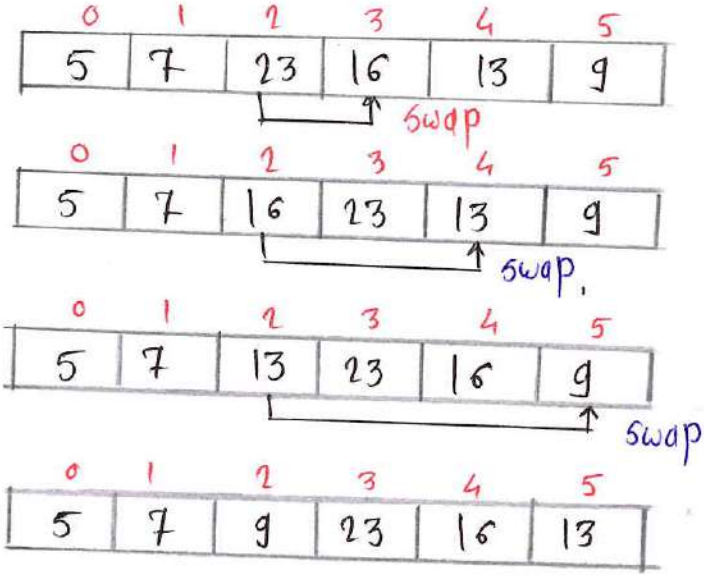


Pass-II : MIN = 1

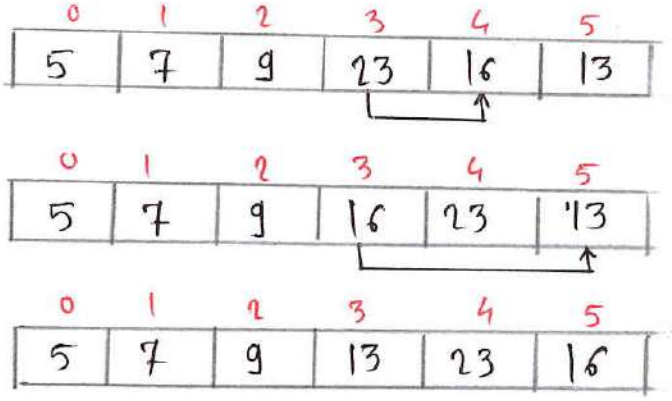




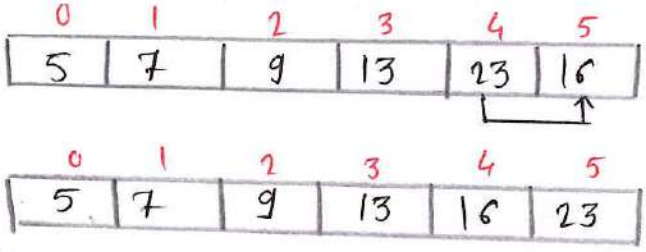
pass - III MIN = 2



pass - IV MIN = 3



pass-V: MIN = 4



Hence, given sequence of array is sorted in ascending order.

3) Insertion Sort

- Insertion sort is a simple algorithm that works by interatively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
- It is like sorting playing cards in your hands.
- You split the cands into two groups: the sorted cards & the unsorted cards.
- Then, you pick a card from the unsorted group & put it in the right place in the sorted group.
  - start with the 2nd element as the first element is assumed to be sorted.
  - Compare the second element with the first if the second is smaller then swap them.
  - Move to the third element, compare it with the first two, & put it in its correct position.
  - Repeat until the entire array is sorted.

Algorithm :

Step 1 : IF it is the first element, it is already sorted.

Step 2 : pick next element

Step 3 : Compare with all elements in the sorted sub list

Step 4 : shift all elements in the sorted sub-list that is greater value to be sorted.

Step 5 : Insert the value

Step 6 : Repeat until list is sorted.

Arrange the given number in ascending order using insertion sort.

A = {77, 33, 44, 11, 88, 22, 66, 55}

⇒

0	1	2	3	4	5	6	7
77	33	44	11	88	22	66	55

N = 8

pass = N - 1 = 8 - 1 = 7

∴ pass = 7

Pass - I :

77 | 33 44 11 88 22 66 55

sorted list | unsorted list

pass - II

77 33 | 44 11 88 22 66 55

Comparing.

33 77 | 44 11 88 22 66 55

Sorted list | unsorted list.

pass - III

33 77 | 44 11 88 22 66 55

Sorted List | unsorted list

33 77 44 | 11 88 22 66 55

Sorted list | Compare No swap | unsorted List

33 44 77 | 11 88 22 66 55

Sorted List | Compare No swap | unsorted list

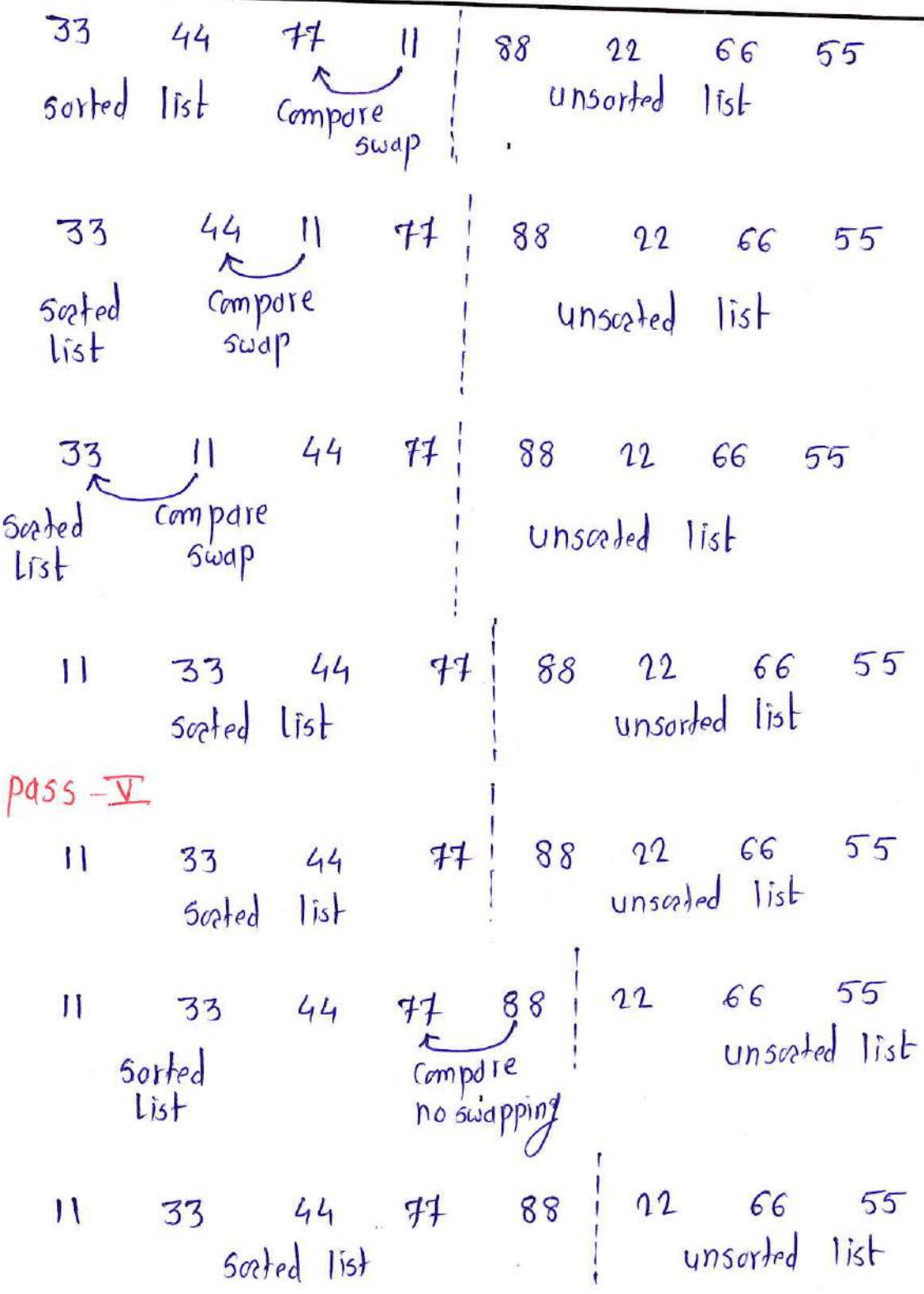
33 44 77 | 11 88 22 66 55

sorted list | unsorted list

pass - IV

33 44 77 | 11 88 22 66 55

Sorted list | unsorted List


33 44 77 11 | 88 22 66 55
sorted list
Compare
swap
unsorted list
33 44 11 77 | 88 22 66 55
sorted list
Compare
swap
unsorted list
33 11 44 77 | 88 22 66 55
sorted list
compare
swap
unsorted list
11 33 44 77 | 88 22 66 55
sorted list
unsorted list
pass - V
11 33 44 77 | 88 22 66 55
sorted list
unsorted list
11 33 44 77 88 | 22 66 55
sorted list
compare
no swapping
unsorted list
11 33 44 77 88 | 22 66 55
sorted list
unsorted list

pass - VI

11 33 44 77 88 | 22 66 55
Sorted list | unsorted list

11 33 44 77 88 22 | 66 55
Sorted list | Compare swap | unsorted list

11 33 44 77 22 88 | 66 55
Sorted list | Compare swap | unsorted list

11 33 44 22 77 88 | 66 55
Sorted list | compare swap | unsorted list

11 33 22 44 77 88 | 66 55
Sorted list | compare swap | unsorted list

11 22 33 44 77 88 | 66 55
Sorted list | unsorted list

pass - VII

11 22 33 44 77 88 | 66 55
Sorted | unsorted

11 22 33 44 77 88 66 | 55
Sorted list | Compare swap | unsorted

Sorted list 11 22 33 44 77 66 88 | 55
Compare swap | unsorted

11 22 33 44 66 77 88 | 55

11 22 33 44 55 66 77 88

Arrang the given no. in ascending order using insertion sort. A = { 12 , 11 , 13 , 5 6 }

pass I : key = (11)

12 | 11 13 5 6

Sorted unsorted.

No element left to campare so insert the key at the beginning.

11 | 12 13 5 6

Sorted unsorted.

pass II : key = (13)

Compare key with 12,
12 is in the correct order, so no changes take place.

11 12 | (13) 5 6

Sorted Unsorted

pass III key : (5)

Compare key with all the elements in the sorted subarray starting with 13, if the element is in the wrong order, shift that element to the right

11 12 13 | (5) 6

Sorted unsorted

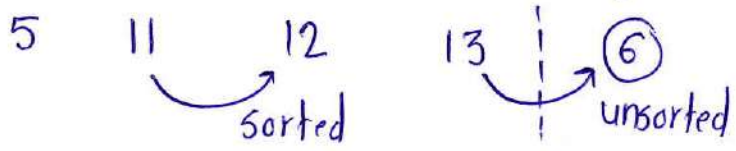
No element left to compare, so insert key at the beginning.

(5) 11 12 | 13 6

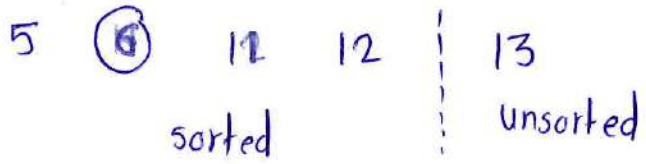
Sorted unsorted

pass IV pass V: key = 6

Compare key with all the elements in the sorted subarray starting with 13, if the elements is in the wrong order, shift that element to the right.



5 is in the correct order, so shifting stops.
Insert key after 5.



The sorted part contains the whole array.
Means that the whole array is sorted.

4) Quick sort :

1) Quick sort is a faster and highly efficient sorting algorithm
2) This algorithm follows the divide and conquer approach
3) Divide & conquer is a technique of breaking down the algorithms into sub problems, then solving sub problems & combining the result back together to solve the orjginal problem.

Divide :
(In divide, first pick a pivot element)
(After that partition or rearrange the array into two sub-arrays).
such that each element in the left sub-array is less than or equal to pivot.

Left sub array → [left] <= [Pivot]
Right sub array → [Right] > [Pivot]

Conquer : Recursively, sort two subarrays with quick sort.

Combine : Combine the already sorted array.

Advantages:

1) It is a divide and conquer algorithm that makes it easier to solve problem.
2) It is efficient on large data sets.
3) It has a low overhead, as it only requires a small amount of memory to function.

Disadvantages:

1) The difficulty of implementing the partitioning algorithm & the aveage efficiency for the worst case scenario, which is not offset by the difficult implementation.
2) Quick sort works by "partitioning" the array to be sorted, then recursively sorting each partition.

Algorithm:

Step 1: If $n<=1$, then return

Step 2: Pick any element v in array.
This v is called pivot.
Rearrange element of array by moving
all elements $x_i > v \rightarrow$ right of v
& all elements $x_i \leq v \rightarrow$ left of v

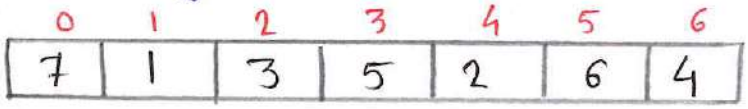
Step 3: Apply quick sort recursively, to
a[0] ..... a[n-1] & to a[j+1] ..... a(n-1)

Entire array will be sorted by selecting an element V.

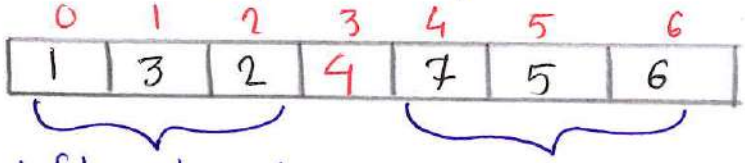
- partitioning the array around V
- Recursively, the array around V
- Recursively, sorting left partition
- Recursively, sorting right partition

* A = {7, 1, 3, 5, 2, 6, 4} sort the given sequence using quick sort

⇒ Given array

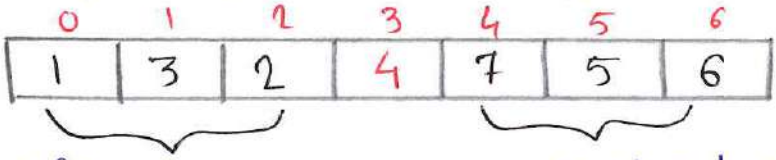


Select a pivot = 4



left subarray of pivot = 4

Right sub-array of pivot = 4



left sub-array of pivot = 4

Right sub-array of pivot = 4

for left sub-array select pivot = 2

For right sub-array select pivot = 6

0	1	2	3	4	5	6
1	2	3	4	5	6	7

Left sub-array of pivot = 2 | Right sub-array of pivot = 2 | LSA of pivot = 6 | RSA of pivot = 6

Combining two sub-array sorted array is

0	1	2	3	4	5	6
1	2	3	4	5	6	7

---

* A = {44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88}

⟹ Given array

0	1	2	3	4	5	6	7	8	9	10
44	33	11	55	77	90	40	60	99	22	88

Select a pivot = 88

0	1	2	3	4	5	6	7	8	9	10
44	33	11	55	77	40	60	22	88	90	99

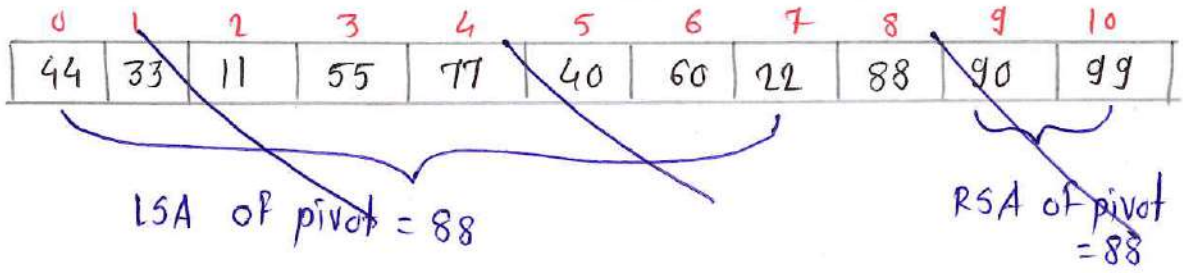
↓ pivot

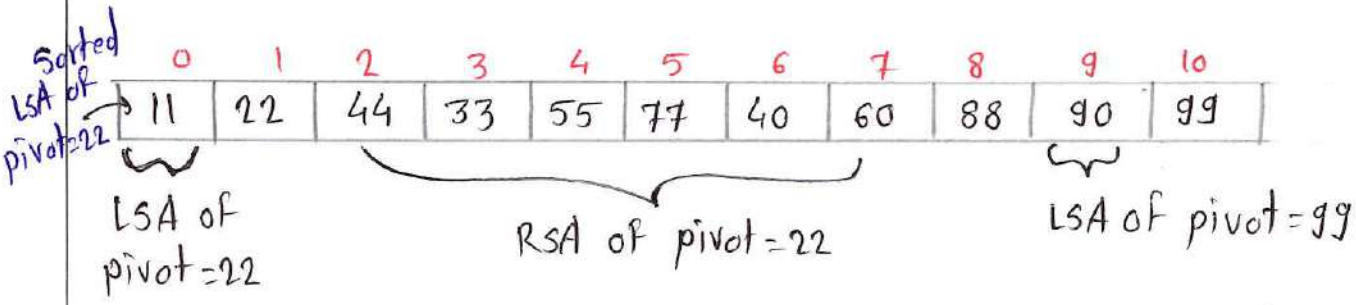
0	1	2	3	4	5	6	7	8	9	10
44	33	11	55	77	40	60	22	88	90	99

LSA of pivot = 88 | RSA of pivot = 88

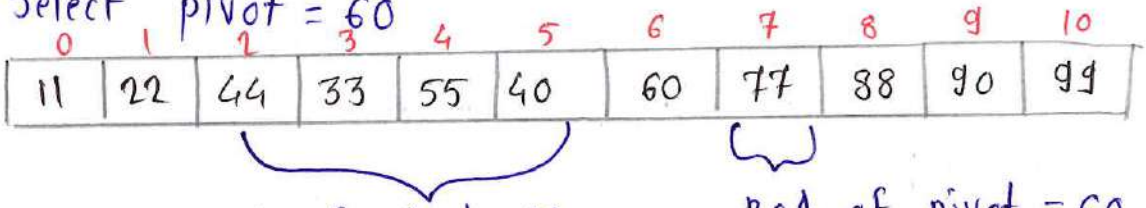
For LSA select pivot = 22

For RSA select pivot = 99

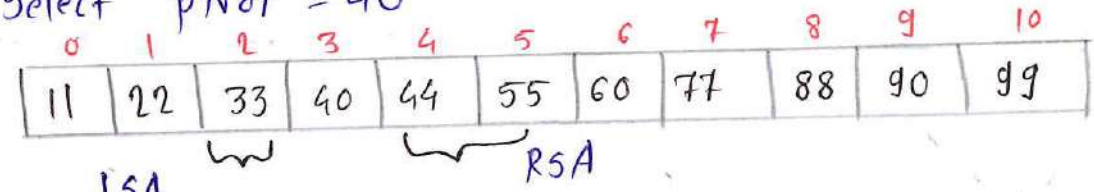




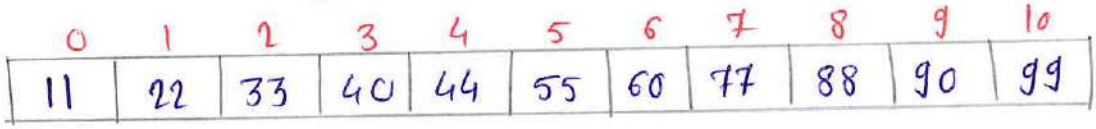
For RSA of pivot = 22
Select pivot = 60



For LSA of pivot = 60
Select pivot = 40



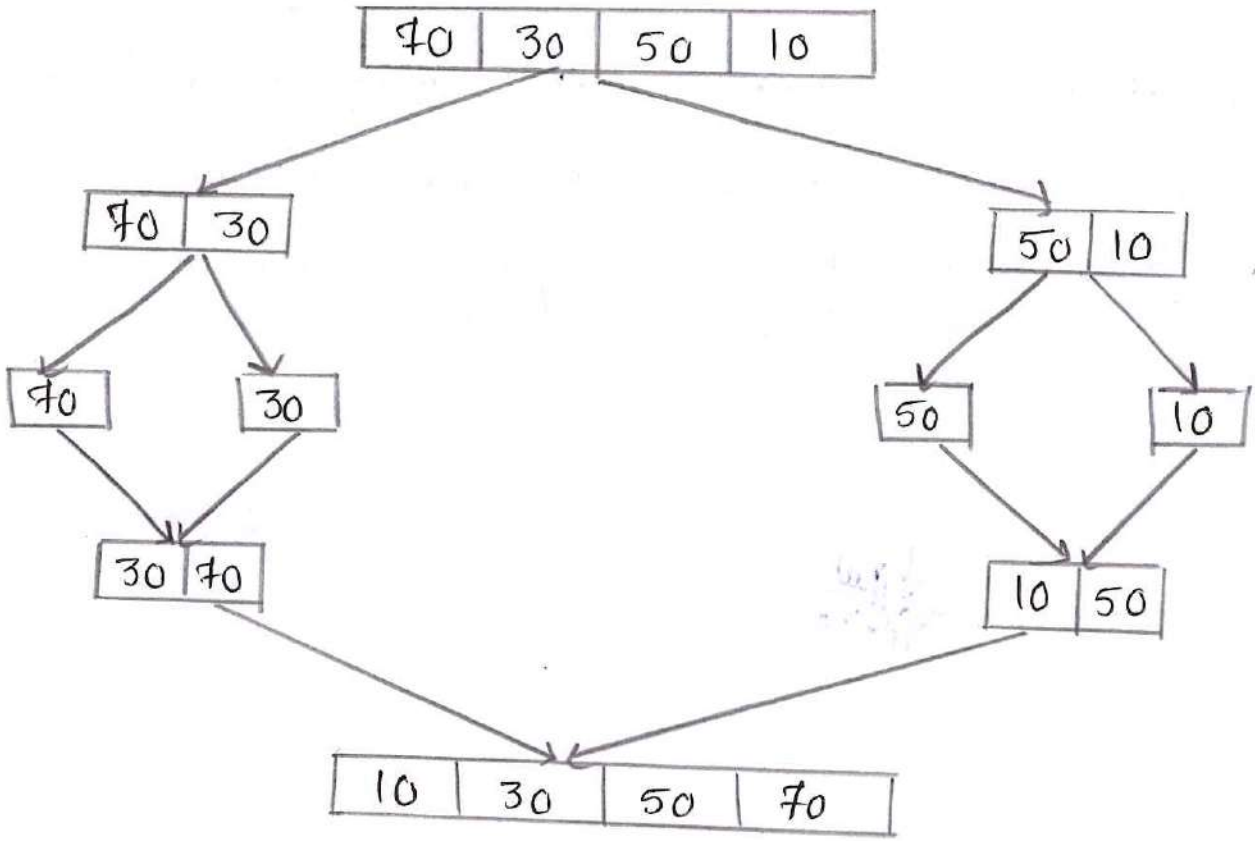
Combining sub-arrays
Sorted array is



5) Merge Sort :

- Merge sort is a popular sorting algorithm known for its efficiency & stability.
- It follows the Divide & Coquer approach.
- It works by recursively dividing the input array into two halves, recursively sorting the two halves & finally merging them back together to obtain the sorted array.

$A = \{70, 30, 50, 10\}$



Differences between Linear Search and Binary Search

	Linear Search	Binary Search
1)	In linear search input data need not to be insorted.	In binary search inputdata need to be in sorted order.
2)	It is also called sequential search.	It is also called half-interval search.
3)	The time complexity of linear search O(n).	The time complexity of binary search O(log n).
4)	Multidimensional array can be used.	Only single dimensional array is used.
5)	It is less complex.	It is more complex.
6)	It is very slow process.	It is very fast process
7)	Linear search performs equality comparisons.	Binary search performs ordering comparisons.

25/5/20

25/5/20

IMP Questions :— 2M & 4M

1) List any four sorting techniques. (2M)
2) State any two differences between linear search and binary search. (2M) (W-24, W-23)
3) Define searching. what are it's types. (2M) (S-23)
4) Discribe bubble sort. state it's advantages & disadvantages (4M)
5) Find the position of elements 65 using binary search method in a array A = {23, 12, 5, 29, 10, 65, 55, 70}
6) Sort the following no in ascending order using selection sort.
7) Explain the working of binary search with an example. (S-19)
8) Sort the following no. in ascending order using Quick sort.
Given no = 50, 2, 6, 22, 3, 39, 49, 25, 18, 5. (S-19)
9) Sort the following no. in ascending order using bubble sort.
Given no = 29, 35, 3, 8, 11, 15, 56, 12, 1, 4, 85, 5 and write the o/p after each interaction.
10) Elaborate the steps for performing selection sort for Given element of array A = {37, 12, 4, 90, 49, 23, -19}
11) state the following terms : i) Searching
ii) Sorting (2M) (S-25)
12) Describe the working of Bubble sort method with example. (S-25)
13) Describe the working of Merge sort method with an example. (S-25)
14) Sort the following in descending order using Quick sort A = {3, 12, 5, 19, 1, 17}. (S-25)
15) Write down an algorithm for selection sort. (2M)
16) Give algorithm for bubble sort. (2M)

28/5/26
28/5/26