DATA STRUCTURE USING ‘C’

UNIT II – SEARCHING AND SORTING.

(Weightage-12marks)

Pranjal Save (SY-COMPS)

Unit– II Searching and Sorting	2a. Explain working of the given search method with an example. 2b. Write an algorithm to search the given key using binary Search method.	2.1 Searching: searching an item in a data set using following methods: (i) Linear Search (ii) Binary Search 2.2 Sorting: sorting of data set in an order using following methods:

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
	2c. Write an Algorithm to sort data using a specified sorting method. 2d. Explain the working of given sorting method step-by-step with an example and small data set.	(i) Bubble Sort (ii) Selection Sort (iii) Insertion Sort (iv) Quick Sort (v) Radix Sort.

SEARCHING

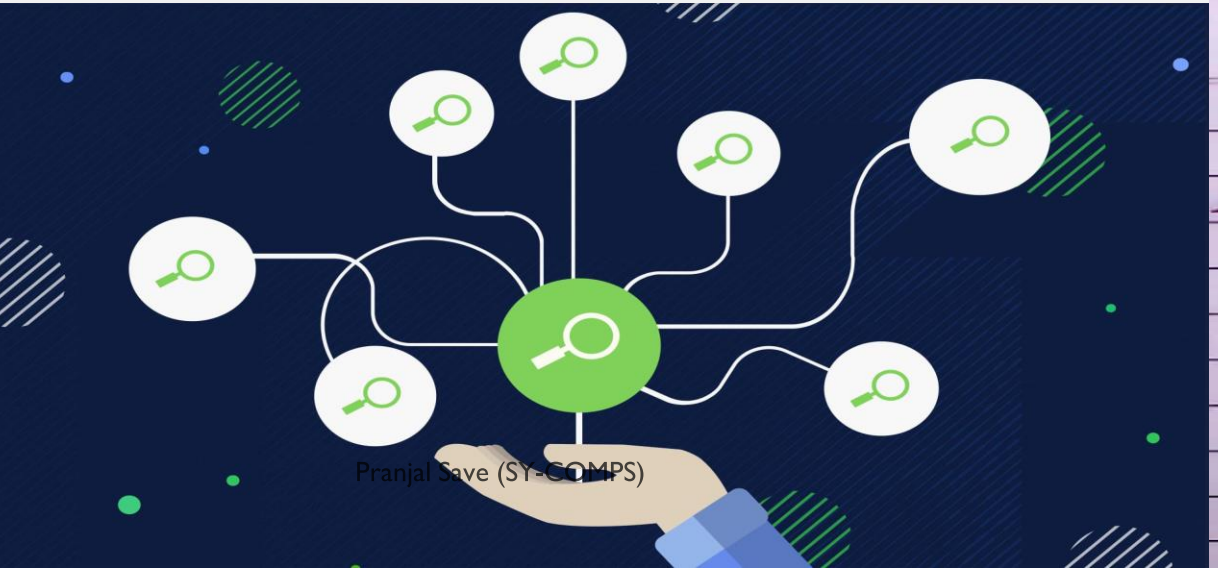
-Linear Search

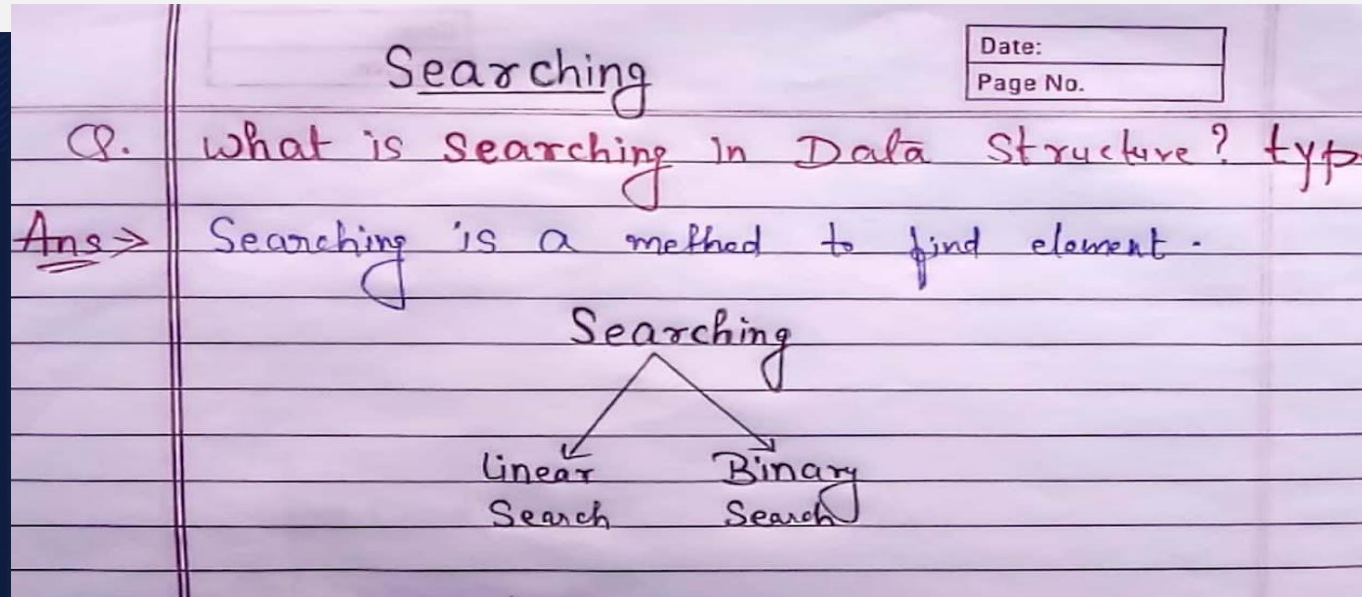
-Binary Search

Pranjal Save (SY-COMPS)

WHAT IS SEARCHING?

- Searching is the process of finding some particular element in the list.
- If the element is present in the list, then the process is called successful, and the process returns the location of that element; otherwise, the search is called unsuccessful.
- Two popular search methods are
  * Linear Search * Binary Search.

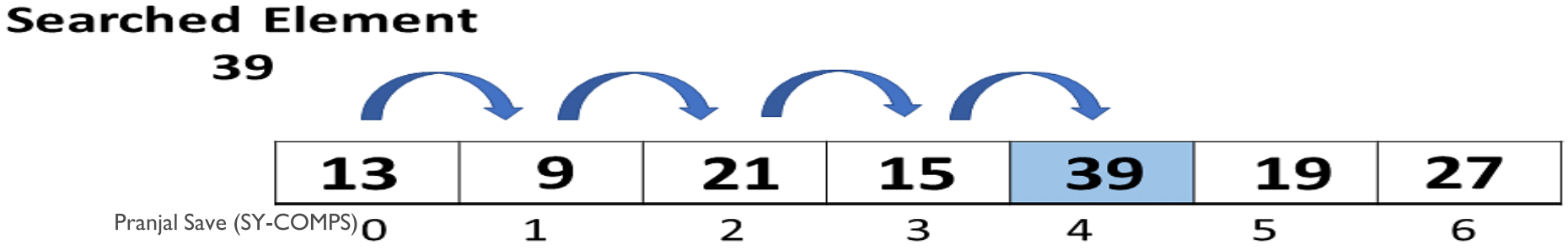




LINEAR SEARCH

- Linear search is also called as sequential search algorithm.
- It is the simplest searching algorithm.
- In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found.
- If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

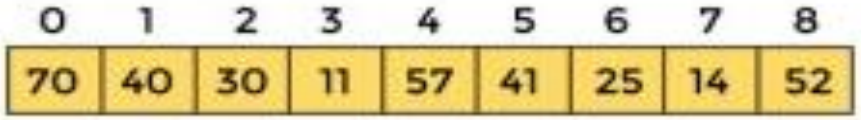
The method in which the elements to be searched is checked in entire data structure in a sequential way from starting to end is known as Linear search.
Linear search is an easiest and straightmost searching technique.



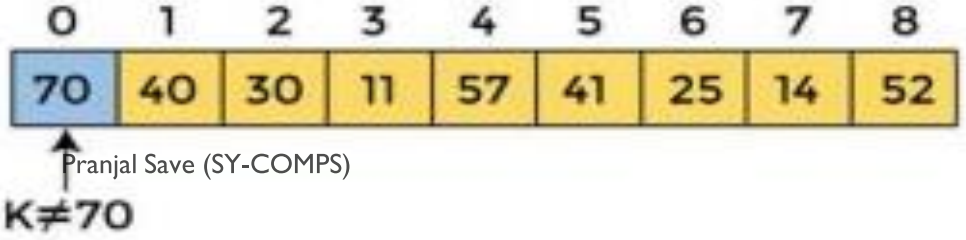
WORKING OF LINEAR SEARCH

Now, let's see the working of the linear search Algorithm. To understand the working of linear search algorithm, let's take an unsorted array. It will be easy to understand the working of linear search with an example

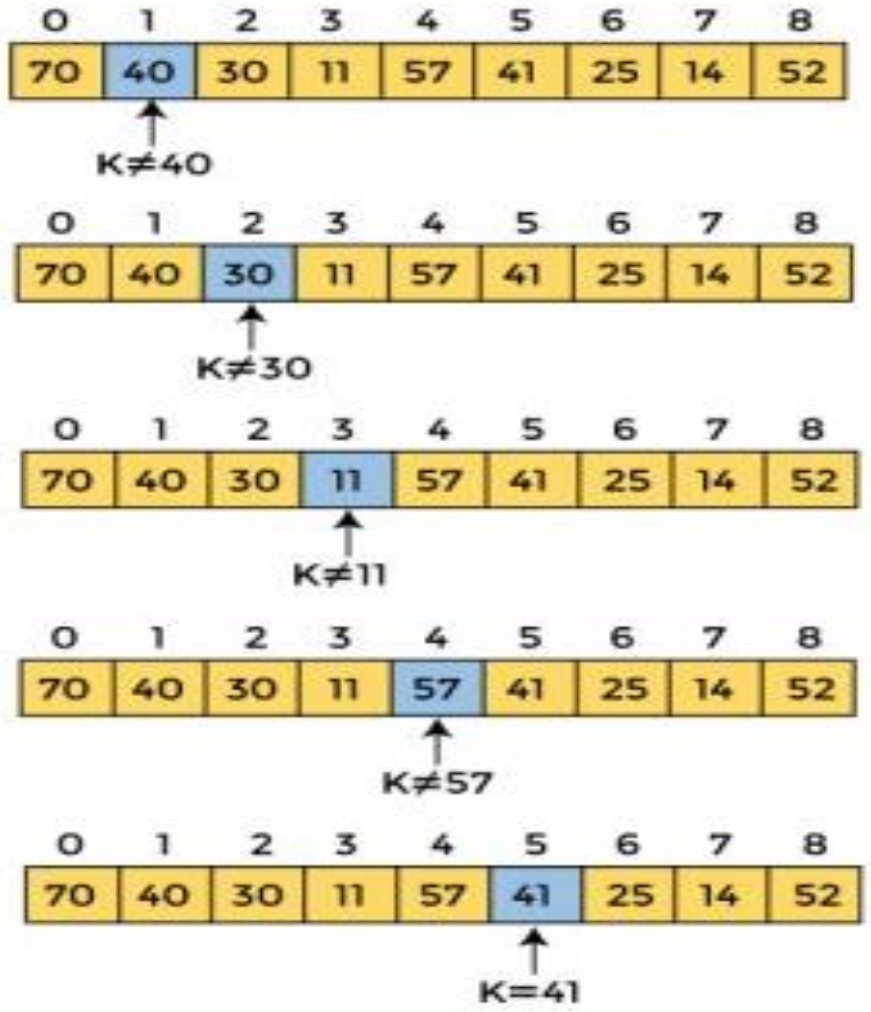
Let the elements of array are -



Let the element to be searched is **K = 41**
Now, start from the first element and compare **K** with each element of the array.



The value of **K**, i.e., **41**, is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found.



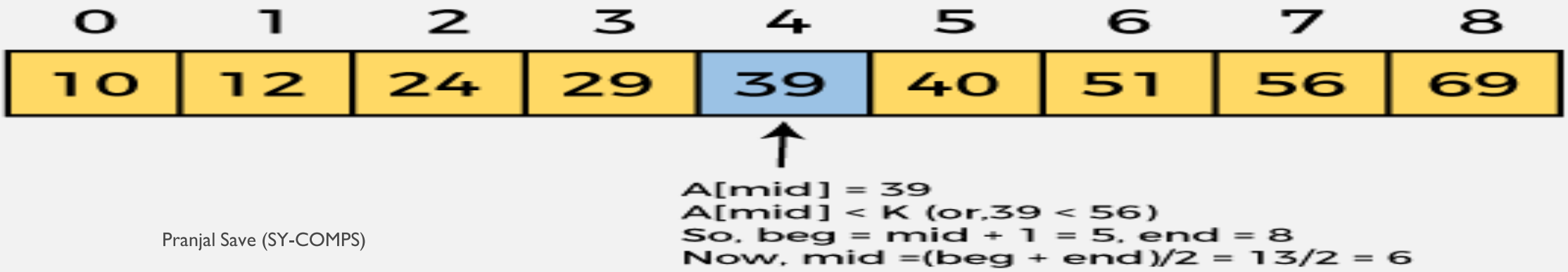
Now, the element to be searched is found. So algorithm will return the index of the element matched.

ALGORITHM OF LINEAR SEARCH:

```
Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value
to search
Step 1: set pos = -1
Step 2: set i = 1
Step 3: repeat step 4 while i <= n
Step 4: if a[i] == val
set pos = i
print pos
go to step 6
[end of if]
set ii = i + 1
[end of loop]
Step 5: if pos = -1
print "value is not present in the array "
[end of if]
Step 6: exit
```

BINARY SEARCH

Binary search is the search technique that works efficiently on sorted lists.
Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list.
If the match is found then, the location of the middle element is returned.
Otherwise, we search into either of the halves depending upon the result produced through the match.
There are two methods to implement the binary search algorithm –
o Iterative method '
o Recursive method
The recursive method of binary search follows the divide and conquer approach

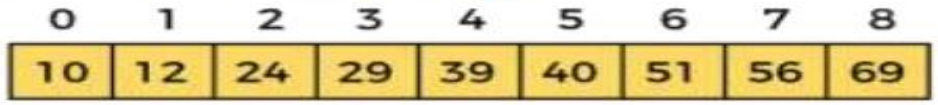


Binary Search *is defined as a searching algorithm used in a sorted array by* ***repeatedly dividing the search interval in half****. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).*

Conditions for when to apply Binary Search in a Data Structure:
To apply Binary Search algorithm:
•The data structure must be sorted.
•Access to any element of the data structure takes constant time.

WORKING OF BINARY SEARCH

Let the elements of array are -



Let the element to search is, **K = 56**

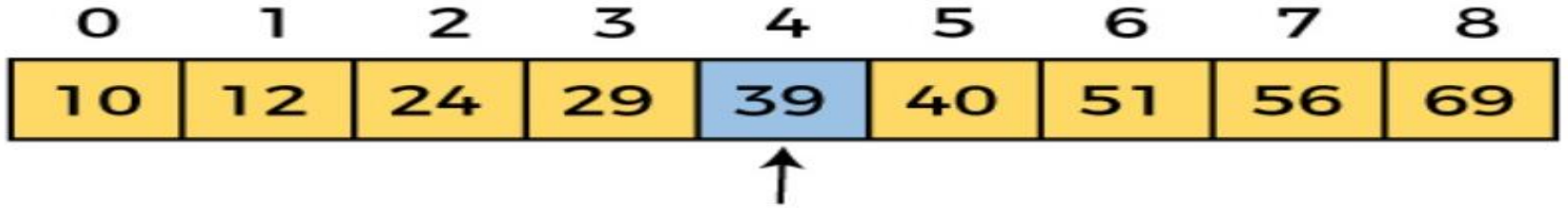
We have to use the below formula to calculate the **mid** of the array –

mid = (beg + end)/2

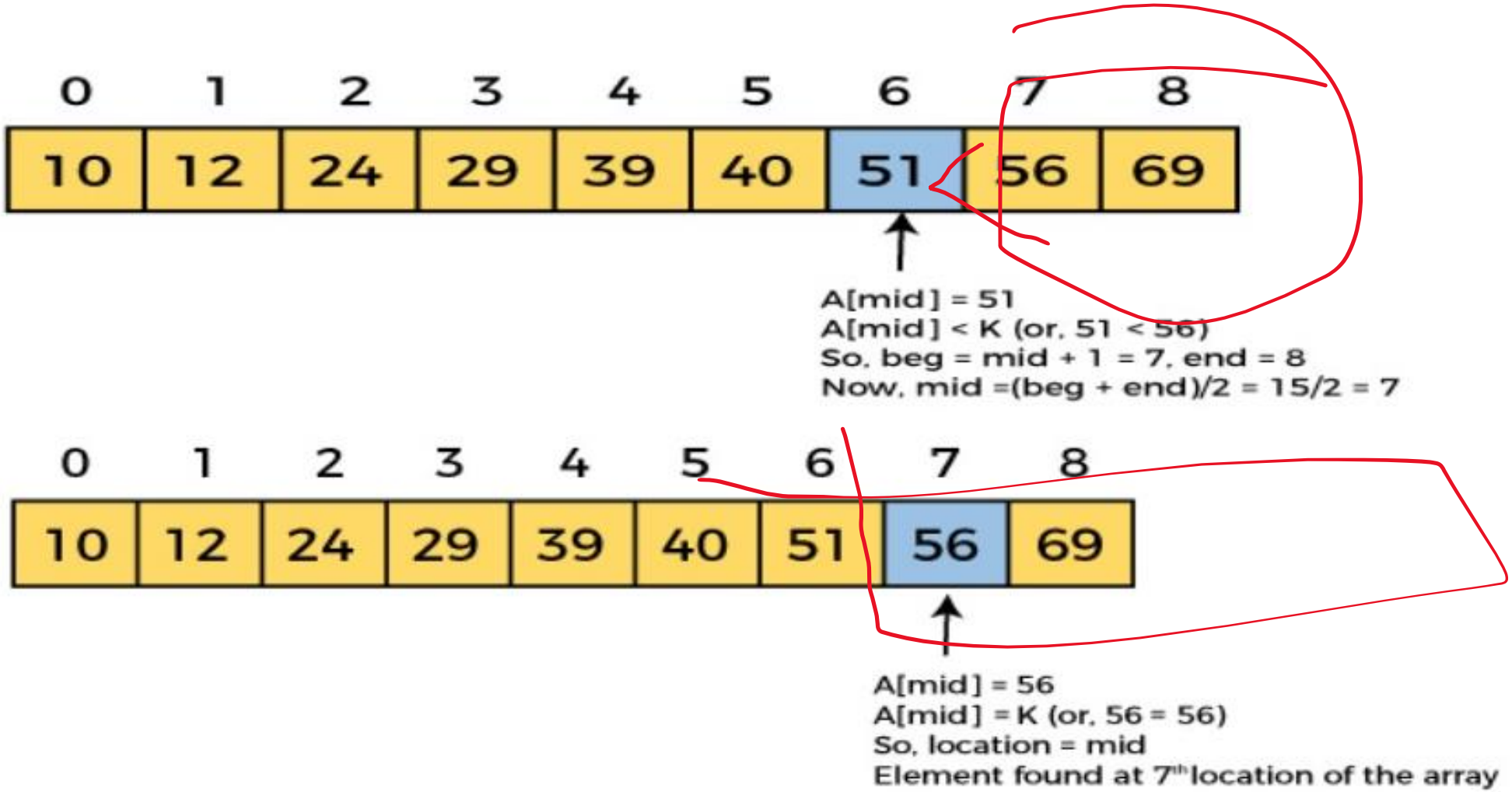
So, in the given array –

beg = 0 **end** = 8 **mid** = (0 + 8)/2 = 4.

So, 4 is the mid of the array.

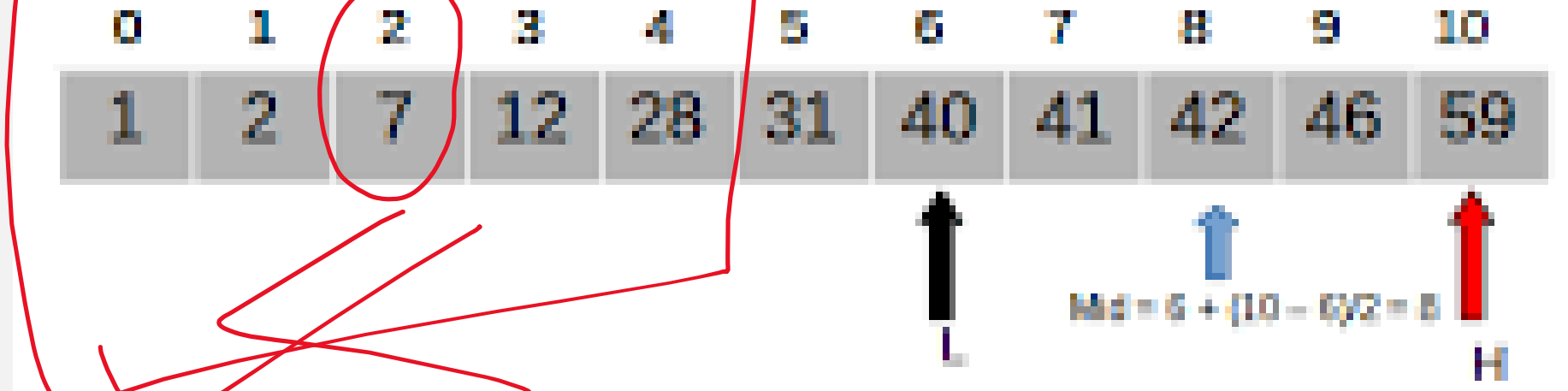


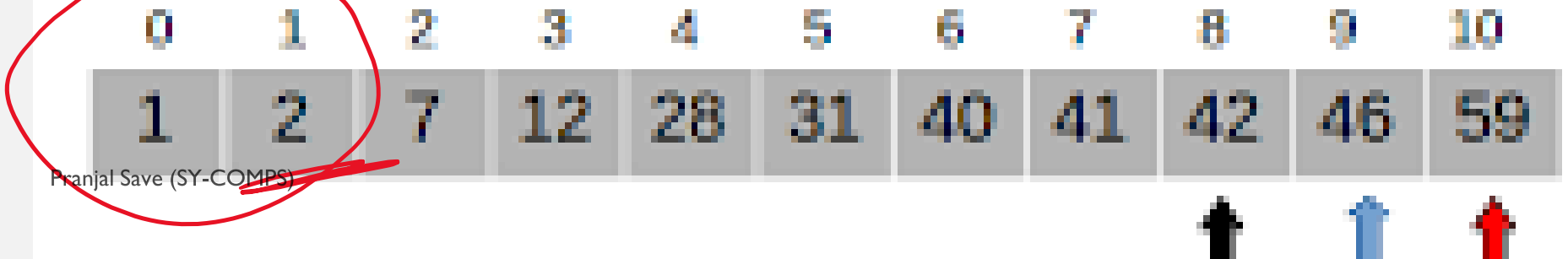
A[mid] = 39
A[mid] < K (or,39 < 56)
So, beg = mid + 1 = 5, end = 8
Now, mid =(beg + end)/2 = 13/2 = 6



Now, the element to search is found. So algorithm will return the index of the element matched.


0 1 2 3 4 5 6 7 8 9 10
1 2 7 12 28 31 40 41 42 46 59
L
Mid = 0 + (10 – 0)/2 = 5
H


0 1 2 3 4 5 6 7 8 9 10
1 2 7 12 28 31 40 41 42 46 59
L
Mid = 6 + (10 – 6)/2 = 8
H


0 1 2 3 4 5 6 7 8 9 10
1 2 7 12 28 31 40 41 42 46 59

BINARY SEARCH ALGORITHM

- *Divide the search space into two halves by **finding the middle index "mid"**.*
- *Compare the middle element of the search space with the key.*
- *If the key is found at middle element, the process is terminated.*
- *If the key is not found at middle element, choose which half will be used as the next search space.*
- *If the key is smaller than the middle element, then the left side is used for next search.*
- *If the key is larger than the middle element, then the right side is used for next search.*
- *This process is continued until the key is found or the total search space is exhausted*

isolated node or not connected node in node.

4. (a) Attempt any THREE of the following: — **12**

Differentiate between binary search and sequential search (linear search). — **4M**

Ans.

Sr. No.	Binary Search	Sequential search (linear search)
1	Input data needs to be sorted in Binary Search	Input data need not to be sorted in Linear Search.
2	In contrast, binary search compares key value with the middle element of an array and if comparison is unsuccessful then cuts down search to half.	A linear search scans one item at a time, without jumping to any item.
3	Binary search implements divide and conquer approach.	Linear search uses sequential approach.
4	In binary search the worst case complexity is O(log n) comparisons.	In linear search, the worst case complexity is O(n), comparisons.
5	Binary search is efficient for the larger array.	Linear search is efficient for the smaller array.

Any four points 1M each

MSBTE QUESTIONS ON SEARCHING

Pranjal Save (SY-COMPS)

- Find location of element 20 by using binary search algorithm in list given below:
  - 10,20,30,40,50,60,70,80
  - Describe working of linear search with example
  - Describe working of Binary search with example
  - Find the position of element 30 using binary search method in array;
  - A={10,5,20,25,8,30,40}
  - Implement a program to search particular data from the given array using Linear Search.
  - Find the position of element 21 using binary search method in array;
  - A={11,5,21,3,29,17,2,45}
  - Difference between linear and binary search
  - Find the position of element 29 using binary search method in array;
  - A={11,5,21,3,29,17,2,45}

SEARCHING

i. Selection sort
ii. Bubble sort
iii. Insertion sort
iv. Quick sort
v. Radix sort

SEARCHING

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements.
The comparison operator is used to decide the new order of elements in the respective data structure.
Following are sorting techniques:

Selection sort
Bubble sort
Insertion sort
Quick sort
Radix sort

SELECTION SORT

☐ In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.
☐ It is also the simplest algorithm. It is an in-place comparison sorting algorithm.
☐ In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part.
☐ Initially, the sorted part of the array is empty, and unsorted part is the given array.
☐ Sorted part is placed at the left, while the unsorted part is placed at the right.
☐ In selection sort, the first smallest element is selected from the unsorted array and placed at the first position.
☐ After that second smallest element is selected and placed in the second position.
☐ The process continues until the array is entirely sorted.
☐ The average and worst-case complexity of selection sort is O(n2), where n is the number of items. Due to this, it is not suitable for large data sets.
☐ Selection sort is generally used when –
o A small array is to be sorted
o Swapping cost doesn't matter
o It is compulsory to check all elements

Algorithm:

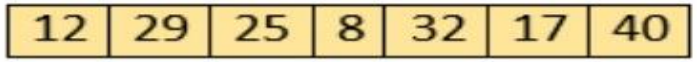
SELECTION SORT(arr, n)

Step 1: Repeat Steps 2 **and** 3 **for** i = 0 to n-1
Step 2: CALL SMALLEST(arr, i, n, pos)
Step 3: SWAP arr[i] with arr[pos]
[END OF LOOP]
Step 4: EXIT

SMALLEST (arr, i, n, pos)
Step 1: [INITIALIZE] SET SMALL = arr[i]
Step 2: [INITIALIZE] SET pos = i
Step 3: Repeat **for** j = i+1 to n
if (SMALL > arr[j])
SET SMALL = arr[j]
SET pos = j
[END OF **if**]
[END OF LOOP]
Step 4: RETURN pos

Working of Selection sort Algorithm

Now, let's see the working of the Selection sort Algorithm.
Let the elements of array are -



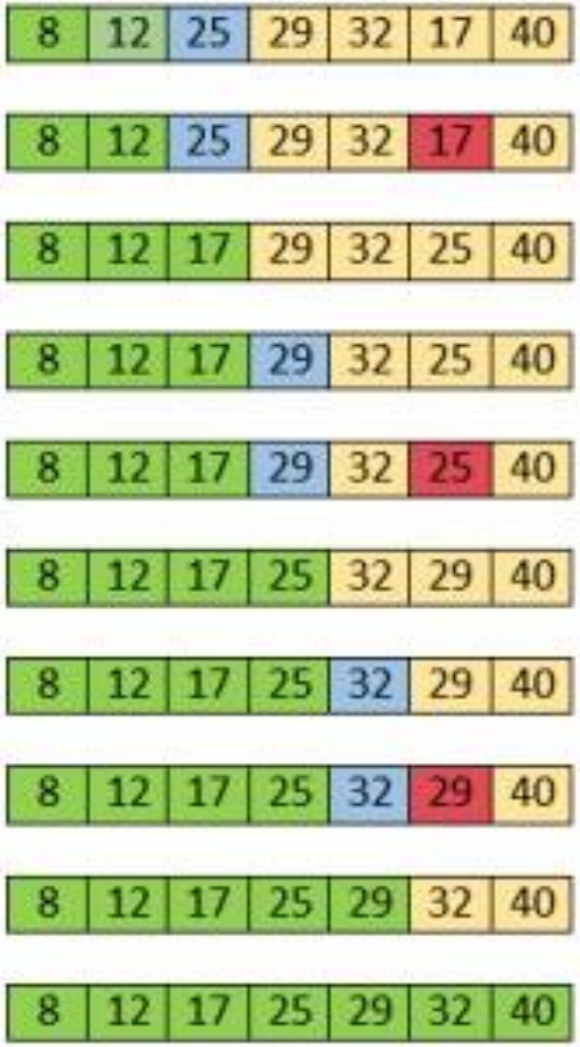
Now, for the first position in the sorted array, the entire array is to be scanned sequentially.
At present, **12** is stored at the first position, after searching the entire array, it is found that **8** is the smallest value.

So, swap 12 with 8. After the first iteration, 8 will appear at the first position in the sorted array.

For the second position, where 29 is stored presently, we again sequentially scan the rest of the items of unsorted array. After scanning, we find that 12 is the second lowest element in the array that should be appeared at second position.

Now, swap 29 with 12. After the second iteration, 12 will appear at the second position in the sorted array. So, after two iterations, the two smallest values are placed at the beginning in a sorted way.

The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process.



Now, the array is completely sorted.

Time Complexity

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

Space Complexity O(1)

Advantages:

1. It performs very well on small lists
2. It is an in-place algorithm. It does not require a lot of space for sorting. Only one extra space is required for holding the temporal variable.
3. It performs well on items that have already been sorted.

Disadvantages:

1. It performs poorly when working on huge lists.
2. The number of iterations made during the sorting is n-squared, where n is the total number of elements in the list.
3. Other algorithms, such as quicksort, have better performance compared to the selection sort.

BUBBLE SORT

☐ Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order.
☐ It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water.
☐ Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.
☐ Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world.
☐ It is not suitable for large data sets.
☐ The average and worst-case complexity of Bubble sort is O(n2), where n is a number of items.
☐ Bubble short is majorly used where –
o complexity does not matter
o simple and shortcode is preferred

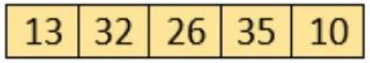
Algorithm

In the algorithm given below, suppose **arr** is an array of **n** elements. The assumed **swap** function in the algorithm will swap the values of given array elements.

```
1. begin BubbleSort(arr)
2.    for all array elements
3.       if arr[i] > arr[i+1]
4.          swap(arr[i], arr[i+1])
5.       end if
6.    end for
7.    return arr
8. end BubbleSort
```

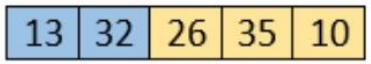
Working of Bubble sort Algorithm

Let the elements of array are -



First Pass

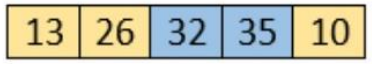
Sorting will start from the initial two elements. Let compare them to check which is greater.



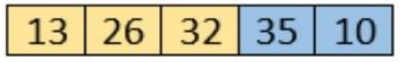
Here, 32 is greater than 13 (32 > 13), so it is already sorted. Now, compare 32 with 26.

Here, 26 is smaller than 36. So, swapping is required. After swapping new array will look like -

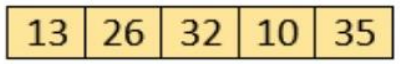
Now, compare 32 and 35.



Here, 35 is greater than 32. So, there is no swapping required as they are already sorted.
Now, the comparison will be in between 35 and 10.



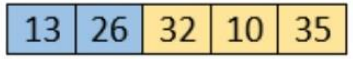
Here, 10 is smaller than 35 that are not sorted. So, swapping is required. Now, we reach at the end of the array. After first pass, the array will be -

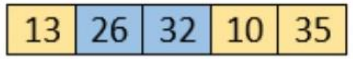


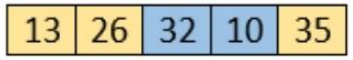
Now, move to the second iteration.

Second Pass

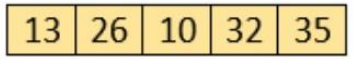
The same process will be followed for second iteration.

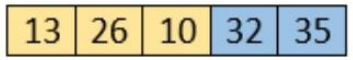






Here, 10 is smaller than 32. So, swapping is required. After swapping, the array will be

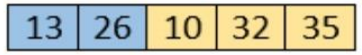


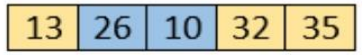


Now, move to the third iteration.

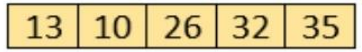
Third Pass

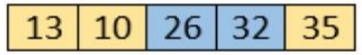
The same process will be followed for third iteration.

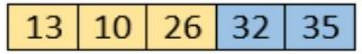




Here, 10 is smaller than 26. So, swapping is required. After swapping, the array will be -



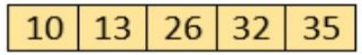




Now, move to the fourth iteration.

Fourth pass

Similarly, after the fourth iteration, the array will be -



Hence, there is no swapping required, so the array is completely sorted.

Bubble sort complexity

Now, let's see the time complexity of bubble sort in the best case, average case, and worst case. We will also see the space complexity of bubble sort.

1. Time Complexity

Case	Time Complexity
Best Case	O(n)
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

2. Space Complexity

Space Complexity	O(1)

INSERTION SORT

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part

---

Characteristics of Insertion Sort:

☐ This algorithm is one of the simplest algorithm with simple implementation

☐ Basically, Insertion sort is efficient for small data values

☐ Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

Algorithm:

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a key.

Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Working of Insertion Sort algorithm

Consider an example: arr[]: **{12, 11, 13, 5, 6}**

12	**11**	**13**	**5**	**6**

First Pass:

- Initially, the first two elements of the array are compared in insertion sort.

12	**11**	13	5	6

- Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.
- So, for now 11 is stored in a sorted sub-array.

11	**12**	13	5	6

Second Pass:

- Now, move to the next two elements and compare them

11	**12**	**13**	5	6

- Here, 13 is greater than 12, thus the elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11

Third Pass:

- Now, two elements are present in the sorted sub-array which are **11** and **12**
- Moving forward to the next two elements which are 13 and 5

11	12	**13**	**5**	6

- Both 5 and 13 are not present at their correct place so swap them

11	12	**5**	**13**	6

- After swapping, elements 12 and 5 are not sorted, thus swap again

11	**5**	**12**	13	6

- Here, again 11 and 5 are not sorted, hence swap again

5	**11**	12	13	6

- Here, 5 is at its correct position

Fourth Pass:

- *Now, the elements which are present in the sorted sub-array are **5, 11** and **12***
- *Moving to the next two elements 13 and 6*

5	*11*	*12*	***13***	***6***

- *Clearly, they are not sorted, thus perform swap between both*

5	*11*	*12*	***6***	***13***

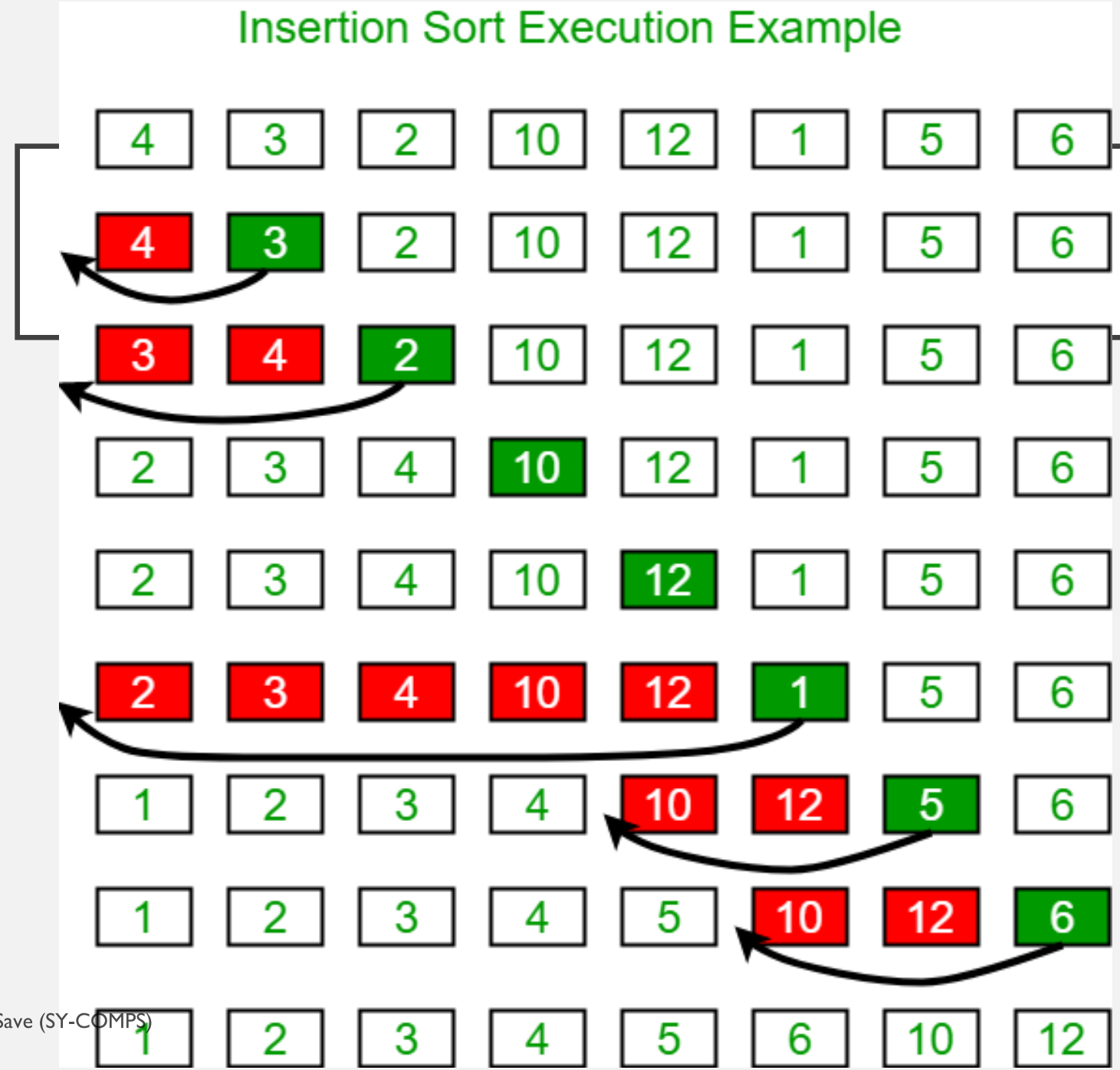
- *Now, 6 is smaller than 12, hence, swap again*

5	*11*	***6***	***12***	*13*

- *Here, also swapping makes 11 and 6 unsorted hence, swap again*

5	***6***	***11***	*12*	*13*

- *Finally, the array is completely sorted.*


Insertion Sort Execution Example
4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Insertion sort complexity

Time & space Complexity:

Case	Time Complexity
Best Case	O(n)
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

Space Complexity O(1)

RADIX SORT

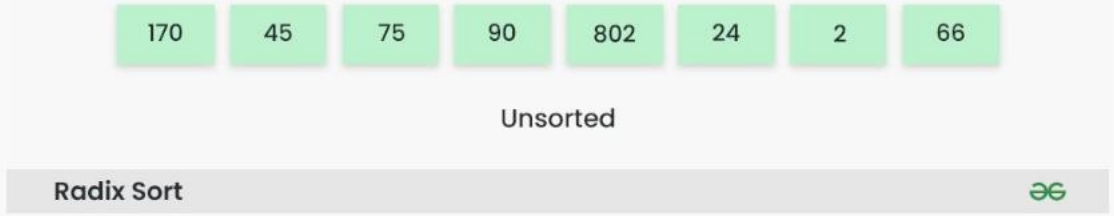
- In this article, we will discuss the Radix sort Algorithm. Radix sort is the linear sorting algorithm that is used for integers.
- In Radix sort, there is digit by digit sorting is performed that is started from the least significant digit to the most significant digit.
- The process of radix sort works similar to the sorting of students names, according to the alphabetical order. In this case, there are 26 radix formed due to the 26 alphabets in English.
- In the first pass, the names of students are grouped according to the ascending order of the first letter of their names
- After that, in the second pass, their names are grouped according to the ascending order of the second letter of their name. And the process continues until we find the sorted list. Now, let's see the algorithm of Radix sort.

ALGORITHM

Step 1: *Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.*
Step 2: *Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place.*
Step 3: *Sort the elements based on the tens place digits.*
Step 4: *Sort the elements based on the hundreds place digits.*
Step 5: *The array is now sorted in ascending order.*

WORKING OF RADIX SORT

To perform radix sort on the array [170, 45, 75, 90, 802, 24, 2, 66], we follow these steps:



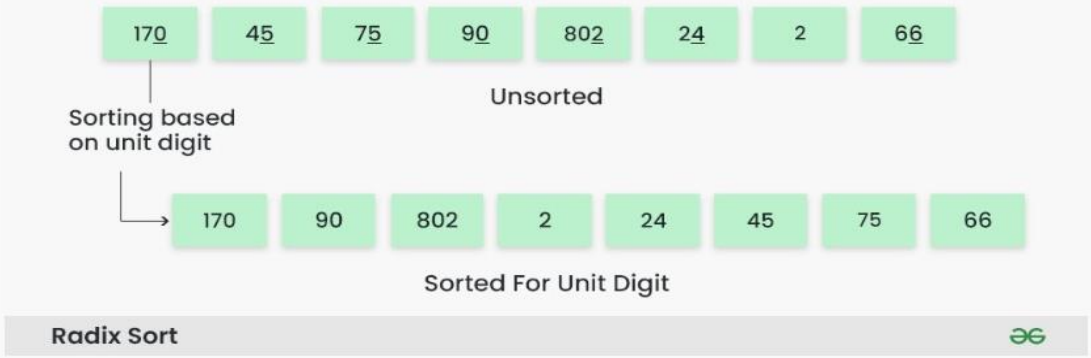
How does Radix Sort Algorithm work | Step 1

Step 1: *Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.*

Step 2: *Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place.*

Sorting based on the unit place:

- *Perform counting sort on the array based on the unit place digits.*
- *The sorted array based on the unit place is [170, 90, 802, 2, 24, 45, 75, 66].*

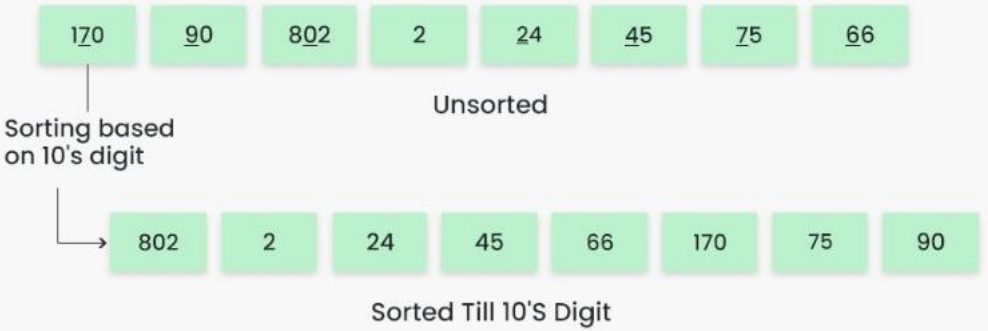


How does Radix Sort Algorithm work | Step 2

Step 3: *Sort the elements based on the tens place digits.*

Sorting based on the tens place:

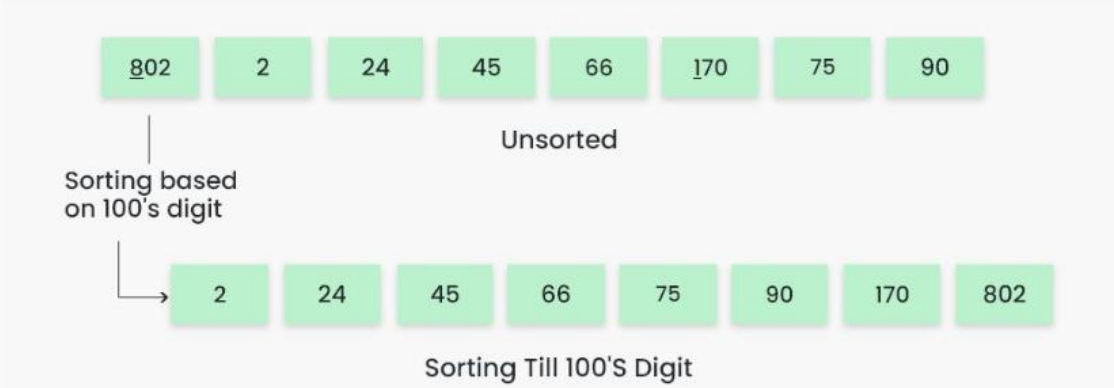
- *Perform counting sort on the array based on the tens place digits.*
- *The sorted array based on the tens place is [802, 2, 24, 45, 66, 170, 75, 90].*



Step 4: *Sort the elements based on the hundreds place digits.*

Sorting based on the hundreds place:

- *Perform counting sort on the array based on the hundreds place digits.*
- *The sorted array based on the hundreds place is [2, 24, 45, 66, 75, 90, 170, 802].*



> ***Step 5:*** *The array is now sorted in ascending order.*
>
> *The final sorted array using radix sort is [2, 24, 45, 66, 75, 90, 170, 802].*

SORTING QUESTIONS

- Describe working of bubble sort with example. (18-W) 4MARKS (22-S)
- Describe working of selection sort method. Also sort given input list in ascending order using selection sort input list – 55, 25, 5, 15, 35. (18-W) 6MARKS
- Sort the given numbers in ascending order using Radix sort : 348, 14, 641, 3851, 74 (19-W) 4MARKS
- Elaborate the steps for performing selection sort for given elements of array. A = {37, 12, 4, 90, 49, 23, –19} (19-w) 6marks
- Sort the following numbers in ascending order using Bubble sort. Given numbers : 29, 35, 3, 8, 11, 15, 56, 12, 1, 4, 85, 5 & write the output after each interaction. (19-s) 6MARKS
- Sort the following numbers in ascending order using quick sort. Given numbers 50, 2, 6, 22, 3, 39, 49, 25, 18, 5. 4MARKS
- Describe the working of radix sort with example. (22-S) 6MARKS
- a) Describe the working of Selection Sort Method. Also sort given input list in ascending order using selection sort. Input list : 50, 24, 5, 12, 30 (6MARKS)


THANK YOU