2. Searching and Sorting

⇒ Searching
Searching is the finding location of the data with in the given data structure.

- Types of Searching

1] Linear Search
Linear Search is also called as Sequential Search. In Sequential search all elements are examine sequentially starting from 1st element

The process of searching terminates when list is ended or comparison results in success

Algo Search Algorithm starts comparison between a[i] and key value.

- Advantages of Linear Search

- Simple
- easy to understand
- not complex

- Disadvantages of Linear Search
- Should not be used for long list.
- Time for searching will increases if list is long.

Ex: List = 10, 20, 30, 40, 50 & Search element 30

Step 1:

10	20	30	40	50
0	1	2	3	4

10 != 30 Index = Index + 1

Step 2: 10 20 30 40 50

20 != 30 Inder = Index + 1

Step 3: 10 20 30 40 50

30 = 30 → Numie is found

Search is successful

2] Binary Search:

Binary Search is applicable only when the given array is Sorted. This method makes a comparison between key value that is element to be Search and the middle element or Centre element of the array

Since elements are Sorted, Comparison could be continue with either left half of elements or right half of the elements.

Left half of the elements could be selected by simply making J = C - 1

Right half of the elements could be selected by simply making I = C + 1

Process of selecting either the left half or right half continues until the element is found or element is not there.

Ex:

Array List:

0	1	2	3	4	5
2	5	11	19	27	35
↑		↑			↑
i=0		c=2			J=5

i = index of first element

J = index of last element

$$c = \frac{\text{index of first element} + \text{index of last element}}{2}$$

$$c = \frac{i+J}{2}$$

$$c = \frac{0+5}{2} = 2$$

Case 1 : if key == a[c] then element is found

Case 2 : if key > a[c] then right half 19, 27, 35 is selected.

Case 3 : if key < a[c] then left half selected 2, 5

Q Search the Element 29 in the given array
{ 5, 9, 11, 15, 25, 29, 30, 35, 40 }

→

0	1	2	3	4	5	6	7	8	→ Index
5	9	11	15	25	29	30	35	40	→ Element

i = Index of first element = 0
J = Index of last element = 8

$$\text{Centre} = \frac{i+J}{2}$$

$$= \frac{0+8}{2}$$

$$= 4$$

Centre 4 element 25

key value 29 The element to be search that is key value equal to 29

Step 1: key > a[c]
29 > 25

The eight half part is selected

29	30	35	40	Element
5	6	7	8	Index
↑	↑		↑	
i=5	C=6		J=8	

$$c = \frac{i+j}{2} = \frac{5+8}{2} = \frac{13}{2} = 6$$

Step 2: key is less than a[c] that is
29 < 30.
Left half part will be selected.
29 → Element, 5 = Index
∴ left half part contain only one element that is key value
∴ Search is successful.

∴ key value is get on location 5 that is 29

Q. Find the position of element 29 using Binary search in Array given below

A = { 11, 5, 21, 3, 29, 17, 2, 43 }

→ Sorted list of given arry:
Data should be sorted for Binary search

2	3	5	11	17	21	29	43	→ Element
0	1	2	3	4	5	6	7	→ Index

i = Index of first element = 0
j = Index of last element = 7

$$c = \frac{0+7}{2} = 3$$

centre = 3
element = 11

Key value is 29 The element to be search in right part.

Step 1 : Key > a[c]

29 > 11

The right half part is selected

	4	5	6	7	→ Index
#	17	21	29	43	→ Element
	↓ i	↓ c		↓ j	

$$c = \frac{4+7}{2} = 5$$

Step 2 :

Key > 21

Key > a[c]

29 > 21

Left part is selected.

Elements of left part

29 43

6 7

$$\therefore \text{The } c = \frac{6+7}{2} = \frac{13}{2} = 6$$

∴ The (element) position of element 29 is found at position 6

⇒ Sorting

Sorting is the Arranging data in a logical order.

- Types of Sorting.

1] Bubble Sort

Bubble Sort is one of the simplest and most popular sorting method.

The basic idea behind bubble sort is as a bubble rises up in the water, the smaller element goes to the bignning

This method is based on successive selecting element through exchange of adjacent element

Let n be the number of elements in array the 1st pass begins with the comparison of a of n-1, and a of n-2 is larger than a of n-1 the two elements are exchange.

ex: P-1: 5 2 1

2 5 1

P-2: 2 1 5

1 2 5

1 2 5

Q. original array with n = 6, 5, 9, 6, 2, 8, 1

→ Pass 1:

6 5 9 6 2 8 1

5 6 9 6 2 8 1

5 6 9 6 2 8 1

5 6 6 9 2 8 1

5 6 6 2 9 8 1

5 6 6 2 8 9 1

5 6 6 2 8 1 9 output of pass 1

Pass 2 :

5 6 6 2 8 1 9

5 6 6 2 8 1 9

5 6 6 2 8 1 9

5 6 6 2 8 1 9

5 6 6 2 1 8 9

5 6 2 6 1 8 9 output of pass 2

Pass 3:

5 6 2 6 1 8 9

5 6 2 6 1 8 9

5 2 6 6 1 8 9

5 2 6 6 1 8 9

5 2 6 1 6 8 9

5 2 6 1 6 8 9

5 2 6 1 6 8 9 → output of pass 3

Pass 4:

5 2 6 1 6 8 9

2 5 6 1 6 8 9

2 5 6 1 6 8 9

2 5 1 6 6 8 9

2 5 1 6 6 8 9

2 5 1 6 6 8 9 → output pass 4

Pass 5:

2 5 1 6 6 8 9

2 5 1 6 6 8 9

2 1 5 6 6 8 9

2 1 5 6 6 8 9

2 1 5 6 6 8 9

Pass 6:

2 1 5 6 6 8 9

1 2 5 6 6 8 9 Sorted List

Swapping logic:

```
temp = a[J]
a[J] = a[J+1]
a[J+1] = temp
```

2] Selection Sorting

Selection Sorting is the very simple Sorting method.

In the i^{th} pass we select the element with lowest value among a[i], a[i]+1, upto a[i] n-1 and be swap it with a[i]
As a result after i passes elements will be Sorted order.

Ex:

original Array 5 9 1 11 2 4

5 9 1 11 2 4

After pass1 → 1 9 5 11 2 4

After pass 2 → 1 2 5 11 9 4

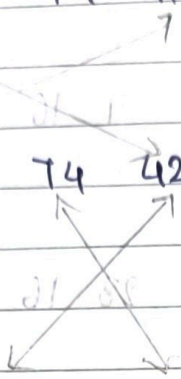
After pass 3 → 1 2 4 11 9 5

After pass 4 → 1 2 4 5 9 11

After pass 5 → 1 2 4 5 9 11

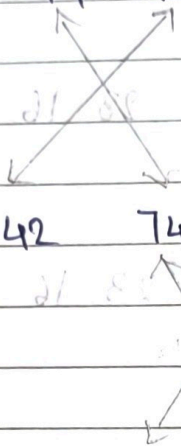
2. Sort the no in ascending order
A = {42, 23, 74, 11, 65}

→ original array 42 23 74 11 65



The Small no is 11 Comparing with a[0] and exchange it.

After Pass 1 11 23 74 42 65



The Small no is 42 Comparing with a[2] & exchange it

After Pass 2 11 23 42 74 65

The Small no is 65 comparing with a[3] & exchange it.

After Pass 3 11 23 42 65 74

11 23 42 65 74 ⟶ Sorted array

PAGE NO.:

3] Insertion Sort

An Element can always be placed at eight place in the sorted list

Ex: 5 9 10 15 20

Element to be place : 6

if the element 6 is to be inserted in sorted list of elements its eight place between 5 and 9

Elements with value greater than 6 should be moved eight by 1 place.

Thus creating a space for incoming element.

Insertion Sort

Q1. Arrange the given no in Ascending order using insertion sort.

77 33 44 11 88 22 66 55

→ | 77 | 33 44 11 88 22 66 55 | → Initial unsorted array
Sorted unsorted

Next element is 33 insert before 77 in Sorted array

| 33 77 | 44 11 88 22 66 55 | → After Pass 1
Sorted unsorted

Next element is 44 insert between 33 & 77

| 33 44 77 | 11 88 22 66 55 | → After pass 2
Sorted unsorted

Next element is 11 insert before 33 in sorted array

| 11 33 44 77 | 88 22 66 55 | → After Pass 3
Sorted unsorted

Next element is 88 insert before 77 in Sorted array

| 11 33 44 77 88 | 22 66 55 | → After Pass 4
Sorted unsorted

Nexte element is 22 insert between 11 & 33 in Sorted array

PAGE NO:

11	22	33	44	77	88		66	55
Sorted							unsorted	

→ After Pass 5

Next element is 66 insert in between 44 & 77 in Sorted array.

11	22	33	44	66	77	88		55
Sorted								unsorted

→ After Pass 6

Next element is 55 insert in between 44 & 66 in Sorted array

11	22	33	44	55	66	77	88

After pass 7

Sorted list

4] Radix Sort :-

Radix Sort is generalization of bucket Sort to Sort decimal numbers where the radix or base is 10. We need 10 bucket

In the first pass numbers are sorted on ~~list~~ Least Significat digit. numbers with the same Least significat digit are stored in same bucket in the.

In the 2nd pass numbers are sorted on 2nd least significand digit.

at the end of every pass numbers in buckes are merge to produce a Common List

Number of passes required to sort is equal to number of digit in the largest number in the list.

PAGE NO.

① Sort the given no in ascending order using radix sort.
Sort 348, 14, 614, 5381, 47

→ Pass 1:

0	1	2	3	4	5	6	7	8	9
	5381			614 14			47	348	

Merge list : 5381, 14, 614, 47, 348

Pass 2:

0	1	2	3	4	5	6	7	8	9
	614 14			348 47				5381	

Merge list : 14, 614, 47, 348, 5381

Pass 3:

0	1	2	3	4	5	6	7	8	9
47 14			5381 348			614			

14, 47, 348, 5381, 614

merge list :
14, 47, 348, 5381, 614

Pass 4:

614									
848									
47									
14						5381			
0	1	2	3	4	5	6	7	8	9

mergelist : 14, 47, 348, 614, 5381.

This is the sorted list

5] Quick Sort:

Quick Sort is the internal Sorting Algorithm with the time complexity equal to O(nlogn)

The basic algorithm to Sort an array of n elements can be describe recursively as follow

Step1: if n<=1 return

Step2: Pick any element V in array a[] this is called the pivote

Step3: rearrange elements of the array by moving all elements $x_i > V$

Right of V & all the elements $x_i < V$ left of V

Q. 5, 1, 2, 90, 8, 13

→

5 1 2 9 0 8 13 ← original list /array

Select Pivot element (5)

1 2 0 5 9 8 13

(1) 2 0 (5) (9) 8 13

Selecting 1 as pivot elem

Selecting 9 as pivot element

0 (1) 2 5 8 (9) 13

This is sorted list by using quick sort algorith = 0 1 2 5 8 9 13

Time Complexities of different Algorithm

1] Binary Search.
Time Complexity: $\log_2 n$

2] Sequential Search
Time Complexity: $O(n)$

3] Bubble Sort
Time Complexity: $O(n^2)$

4] Insertion Sort
Time Complexity $O(n^2)$

5] Selection Sort
Time Complexity $O(n \log n)$

6] Quick Sort
Time Complexity $O(n^2)$