Ques 1a What do you mean by searching and sorting, and what are its types.

Ans Searching: The process of finding the desired information from the set of items stored in the form of elements in various forms, such as an array, tree, graph or linked list.

Types of Searching:

1. Linear Search: In this algorithm, each element in the collection is sequentially check until the desired item is found, or the entire list is traversed. It is suitable for small-sized or unsorted lists, but its time complexity is O(n) in the worst case.

2. Binary Search: This algorithm is applicable only to sorted lists. It repeatedly compares the middle element of the list with the target element and narrows down the search range by half based on the comparison result. Binary search has a time complexity of O(log n), making it highly efficient for large sorted list.

→ Sorting: Sorting in data structure is arranging the elements of the data structure in a specific. Now, this order can be increasing

or decreasing. It makes searching for elements in them easy and quick.

Types of Sorting:

1. Bubble Sort: It is one of the simplest algorithm which repeatedly iterates through the list, compares the elements next to each other, and swaps according to the condition passed to them.

2. Insertion Sort: It is a simple sorting algorithm. It works by building a sorted list one element at a time. It takes an element from the unsorted part of the list and places in its correct position in the sorted portion. This process is repeated until all the elements of the list are sorted.

3. Selection Sort: Selection sort works by repeatedly finding the smallest element from the array and then putting it in the beginning and then again finding the smallest from the rest of the array other than the previous one and arranging it in the beginning.

4. Quick Sort: It is based on partition. It is also known as partition exchange sorting. The basic concept of this algorithm is to pick one element from an array and arrange the rest of the elements around it, that element is known as the pivot element.

5. Merge Sort: It is a popular sorting algorithm that uses a divide and conquer approach. It works by recursively dividing an array into two

values, and then merging the two sorted values back together to produce a fully sorted array.

6. Heap Sort: It is one of the best technique possible for sorting, where we first build the heap using the given elements. The heap is always a complete binary tree. We use the max heap/min heap property to sort the elements. Onces the heap has been created satisfying all the conditions, we swap the last element with the root element and delete the last node.

1b Define bubble sort, its implementation, and algorithm:

Ans Bubble Sort: It is one of the simplest algorithm which repeatedly iterates through the list, compares the elements next to each other, and swaps according to the condition passed to them.

Algorithm:

```
Bubble Sort (A, n)
{
  for k ← 1 to n-1
  {
    ptr = 1
    while (ptr ≤ n-k)
    {
      If A[ptr] > A[ptr+1]
      {
        exchange (A[ptr], A[ptr+1])
      }
      ptr = ptr + 1
    }
  }
}
```

Page No.:
Date:

Implementation:

```
def bubble_sort (arr):
    n = len (arr)
    for k in range (0,n-1):
        for ptr in range (n-1-k):
            if arr[ptr] > arr[ptr+1]:
                temp = arr[ptr]
                arr[ptr] = arr[ptr+1]
                arr[ptr+1] = temp
    return arr
x = [5,2,9,16,8,7,12]
bubble_sort (x)
```

Output: [2,5,7,8,9,12,16]

1c What is Quick Sort, its algorithm, and implementation?

Ans Quick Sort: The basic concept of this algorithm is to pick one element from an array and arrange the rest of the elements around it, that element is known as the pivot element and that element divides the main array into two sub arrays. Once the pivot is fixed then it moves all the elements lesser than it to left and elements greater than it to right. This procedure is repeated recursively until there is one element left in the sub array.

Time complexity for best and average case is O(n log n) and worst case is $O(n^2)$.

Space complexity is O(log(n)) due to recursion.

Algorithm:

Partition Algorithm:

Partion (arr, l, h)
pivot = arr[h]
j = h-1
while (i ≤ j):
    while (i ≤ j) arr[i] ≤ pivot
    increment i
    while (i ≤ j) and arr[j] ≥ pivot
    decrement j
    If i < j:
        Swap arr[i] and arr[j]
Swap arr[i] and arr[h]
Return i
End

Quick Sort Algo:

Quick Sort (arr, l, h)
    If h is Null:
        h = length (arr) - 1
    If i < h:
        pivot-index = partition (arr, l, h)
        Quick Sort (arr, l, pivot - index - 1)
        Quick Sort (arr, pivot - index + 1, h)
End

Implementation:

def part (arr, l, h):
    pivot = arr[h]
    i = l
    j = h-1

```
    i = l
    j = h-1
    while (i <= j):
        while (i <= j) and arr[i] <= pivot:
            i = i+1
        while (i <= j) and arr[j] >= pivot:
            j = j-1
        if i < j:
            arr[i], arr[j] = arr[j], arr[i]
    arr[i], arr[h] = arr[h], arr[i]
    return i
def quicksort (arr, l=0, h=None):
    if h is None:
        h = len (arr) - 1
    if l < h:
        p_e = part (arr, l, h)
        quicksort (arr, l, p_e - 1)
        quicksort (arr, p_e + 1, h)
array = [46, 87, 97, 45, 34, 89, 23, 79]
quicksort (array)
print ("Sorted array", array)
```

Output = Sorted array [23, 34, 45, 46, 79, 87, 89, 97]

Ques: What do you mean by stack data structure. Write all its operations, algorithm and implementation.

Ans: A stack is a type of data structure that works like a pile of plates. You can only add and remove plates from the top of pile, not from middle and bottom. The order may be LIFO (Last In First Out) or FILO (First in Last Out).

⇨ Binary Search Algo:

Step 1. Initialize two pointers
Set lb = 0
Set ub = (length of list) - 1

Step 2. Repeat the following steps while lb <= ub:
(1) Calculate middle index
mid = (lb + ub) // 2
(2) Compare the mid value with the target:
If list[mid] == target:
return element found
elif list[mid] < target:
return lb = mid + 1
else:
return ub = mid - 1
else:
return element not found

⇨ Linear Search	Binary Search
→ Also known as Sequential Search	Also known as half interval Search
→ Time Comp O(n)	Time Comp O(log n)
→ Work on Multi dimensional array	Work on single dimensional array
→ Not time efficient	Save Time
→ Slow Process	Fast Process

Date / /
Page No.

Linear Search: Search Particular element

Best Case → O(1)
Worst Case → O(n)

```
def linear(array, target):
    for i in range(len(array)):
        if array[i] == target:
            return i
    else:
        element not found.
```

Binary search: array must be in sorted form then the target value get searched.

```
def binary(array, target)
    left, right = 0, len(array) - 1
    while left <= right:
        mid = (left + right) // 2
        if array[mid] == target
            return mid
        elif array[mid] < target
            left = mid + 1
        else
            right = mid - 1
    else
```

Best case ← O(1)

array divided into 2 parts. Then checked whether the target is in right or left array. Then pick that part and again divide it into two parts and repeat the process

Time Complexity → log(n)

0	1	2	3	4	5
10	20	30	40	50	

target = 40

→ divide it $\frac{4}{2}$ = 2

mid = 2

0	1	2	3	4
10	20	30	40	50

Ignore it

If mid == target then best case is O(1) If not then compare it with target value eg: 30 < 40 True left = mid + 1

Now left = 3

40	50

→ divide it ⇒ $\frac{7}{2}$ ⇒ 3.5

mid = 3 → 40

↓

Solved

eg: l = 5 target = 10 a = [10, 20, 30, 40, 50]

left = 0 right = 4 mid = $\frac{4+0}{2} = \frac{4}{2} = 2$

[10, 20] [30] [40, 50]

a[mid] ≠ target Now a[mid] > target

30 > 10

∴ right = mid − 1

2 − 1 = 1

0	1
10	20

→ divide → $\frac{0+1}{2} = \frac{1}{2} = 0.5$

a[mid] = target mid = 0

yes