* Data Structure.

Section B subject / module.

In exam minimum 7 Question and maximum 8/9 questions asked.

mainly question based on pseudo code oriented. and theory or concept based.

* Why to Study ?

- To build a foundation / base which is important to learn and implement data structure and algorithm in CDAC courses.

- The need of Data structure in programming to achieve efficiency in operation.

* What is Data Structure. ?

- Data structure is programming concept. It is way to store data element in memory [RAM] in organised manner.

- We can do addition, deletion, sorting, searching, traversal can be perform on it efficiently.

* Data Types.

There are two types of data types.
1] Premitive / predefined / Builtin
2] Non-premitive / user-defined / derived.

1] Premitive Data types. [Already known to compiler
a) char
b) int
c) Float
d) double.
e) void.

2] Non-premitive Data types.
a) pointer.
b) array
c) structure
d) union.
e) enum.

* Types of Data Structure.

There are two types of data structure.
1] Linear / basic data structure.
2] Non-linear / advanced data structure.

1] Linear / basic data structure.
Data structure in which data elements gets stored in memory in a linear manner / linearly and hence can be accessed linearly.

Following are example of linear DS.
1] Array.
2] Structure / union.
3] Linked list.
4] Stack.
5] Queue.

2] Non-Linear / advanced data structure.
Data structure in which data element get stored in memory non-linear and hence can be accessed non-linearly.

Following are example of non-linear DS.
1] Tree 3] binary heap
2] graph 4] hash table.

* The importance of data structure is to not learning any programming language is used to learn algorithm.

Data structure and algorithm can be impleme-nted in any programming language such as C, C++, java, python.

* What is program? [Written for Machine]

program is a finite set of instruction written in any programming language.

* What is Algorithm? [Written For Users]

- Algorithm is a set of instruction written in english language. It followed to do specific task.

- program is an implementation of algorithm.

* For example.

Write a algorithm for sum of element of array.

Step 1 : Initially take sum as 0.
Step 2 : Traverse an array and add the array element sequentially.
Step 3 : Return sum.

* Psuedo Code.

Psuedo code is a special form of a algorithm

```
Algorithm ArraySum (arr, n)
{
        Sum = 0;
        For (index = 1 ; index <=n ; index ++)
        {
            Sum + = arr [index]
        }
        return sum;
}
```

* Basic Concept of Data structure.

1] Addition.
Adding one or more element in the array by adding method.

2] Deletion.
Deleting elements from the array element.

3] Sorting
Assending / descending order of array element.

4] Searching.
Find element in array.

5] Traversal.
Visiting each element in array exactly once.

6] Array.
Array is a linear or basic data structure which is collection of logical related similar types of data element. get stored in memory at contiguous memory location.

```
int arr [5]; // array notation.
```

Array notation converted to pointer internally.

7] Structure.
It is linear / basic data structure which is collection / list of logically related similar and dissimilar of data element get stored in memory collectively as single / record.

Employee 1 ⟶ 1 Bhakti 15000

```
Struct Employee
{
    int empid;      // 4 bytes
    char name[32];  // 32 bytes.
    float salary;   // 4 bytes
};
```

Size of structure = Sum of size of all mem
sizeof (struct Employee) = 40 bytes.

8] Sorting.

to arrange data element in collection in asc
descending.

1) selection sort.
2) Bubble sort
3) Insertion sort
4) Merge sort
5) Quick sort

If one problem has many solutions we nee
to select efficient solution out of them. a
to decide which solution / algorithm is
efficient one we need to do there analysi

* Searching.

To Search / find an element (key element) in a given collection / list / set of elements.

1. Linear search.
2. Binary search.

* Analysis of Algorithm.

There are two types of analysis of algorithm

1. Time complexity.
2. Space complexity.

Work of calculation / determining how much time i.e computer time and space (computer memory) it needs to run to completion.

1] Time complexity.
It is amount of ~~space~~ time i.e computer time it needs to run for completion.

2] Space complexity.
It is amount of space (memory) to run to completion.

1] Linear Search / sequential search.

- Linear search is a very simple search algorith
In this type of search a sequential Search i
made over all items one by one.

- Every item is checked and if a match is found then that particular item is returned otherwise the search continues till the end of the data collection.

Algorithm

Linear Search (Array A, value X)

Step 1 : Set i to 1
Step 2 : if i > n then go to step 7
Step 3 : if A[i] = x then go to step 6
Step 4 : Set i to i+1
Step 5 : Go to step 2
Step 6 : print element x found at index i
go to step 8.
Step 7 : print element not found.
Step 8 : Exit.

Pseudocode

```
Algorithm LinearSearch (A, Key n)
{
    for (index = 1; index <= n; index ++)
```

```
    {
        if (key == A[index]) // If key found
            return true; // key is found
    }
    return False; // key is not found.
}
```

* Best case occure if key is found at a position 1. means in first comparison.

If sizeOf array = n
no of comparison = 1
This is a Best case.

* Worst case occurs if either key is found at last position or key does not exist.

If sizeOf array = n
no. of comparison = n
This is a worst case.

* Time Complexity.

1] Best case time complexity is if an algorithm takes minimum amount of time to run to completion.

2] Worst case time complexity is if an algorithm takes maximum amount of time to run to completion.

3] Average time complexity is if an algorithm takes neither min or max amount of time to run to completion.

* Asymptotic Analysis.

It is mathematical way to calculate time complexity and space complexity of algorithm without implementation in any programming language.

Assumption :

If running time of an algorithm having additive / substractive / multiplicative / divisive constant it can be neglated.

For Example.

$O(n+3) \longrightarrow O(n)$
$O(n-5) \longrightarrow O(n)$
$O(n/3) \longrightarrow O(n)$
$O(2*n) \longrightarrow O(n)$

There are 3 asymptotic notation.
1) Big Omega (Ω)
2) Big Oh (O)
3) Big theta (Θ)

1) Big omega is used for best case time complexity.

- big omega represent as asymptotic lower boun

2) Big oh (O)
- is used to represt worst case time complexity
- big oh represent upper bound.

3) Big theta (θ)
- This notation represent an average case time complexity.
- big theta represent tight bound.

2] Binary Search Algorithm / Logarithmic Algorithm.

i) Binary search is a fast search algorithm with run time complexity of O(log n), This search algorithm works on the principal of divide and conquer.

ii) To apply binary search on an array prerequisite is that array elements must be in sorted manner.

iii) In this algorithm, in first iteration, by means of calculating mid position big size array gets divided into two subarray's.
   1) Left Subarray
   2) Right subarray

iv) Key element gets compared with element which is at mid position, if Key is found at mid position in very first iteration in only 1 no. of comparison, then it is considered as a best case and in this case an algorithm takes O(1) time.

v) If Key is not found in the first iteration then either it will get searched into left subarray or into the right subarray by applying same logic repeatedly till either key is not found or till max the size of any subarray is valid i.e size of subarray is greater or equal to 1.

* Algorithm.

Binary Search.

Step 1:
Accept Key from user.

Step 2:
- calculate mid pos by formula
  ⇒ mid = (left + right) / 2

- Step 3: By means of calculating mid position big size array gets divided logically into two subarray's.
  i) left subarray
  ii) Right subarray.

- left subarray is from left to mid-1 and right subarray is from mid+1 to right.

i) For left subarray
Value of left remains as it is, value of right is equal to mid-1.

ii) For Right subarray.
Value of right remains as it is, value of left is equal to mid+1.

Step 3:
Compare value of key with element at mid position if key matches with element at mid position return true.

Step 4:
If key do not matches then search key either into left subarray or into the right Subarray.

Step 5:
Repeat step-2, step 3 and step 4 till either key not found or max till subarray is valid if subarray is invalid then return false indicate key not found.

if (left <= right) ⟹ subarray is valid.
if (left >= right) ⟹ subarray is invalid.

Pseudocode.

```
binarySearch ( arr, size)
    loop until beg is not equal to end
midIndex = (beg + end)/2
  if (item == arr [midindex])
       beg = midindex + 1
  else
      end = midindex - 1
```

* Beg equal to key search by user.

* Recurssion Pseudocode.

```
binarySearch (arr, item, beg, end)
  if beg <= end
     midindex = (beg + end)/2
   if item == arr (midindex)
     return midindex
   else if item < arr [midindex]
      return binarySearch (arr, item, midindex+1,
                                                end)
   else
    return binarysearch (arr, item, beg, midindex-1

return -1.
```

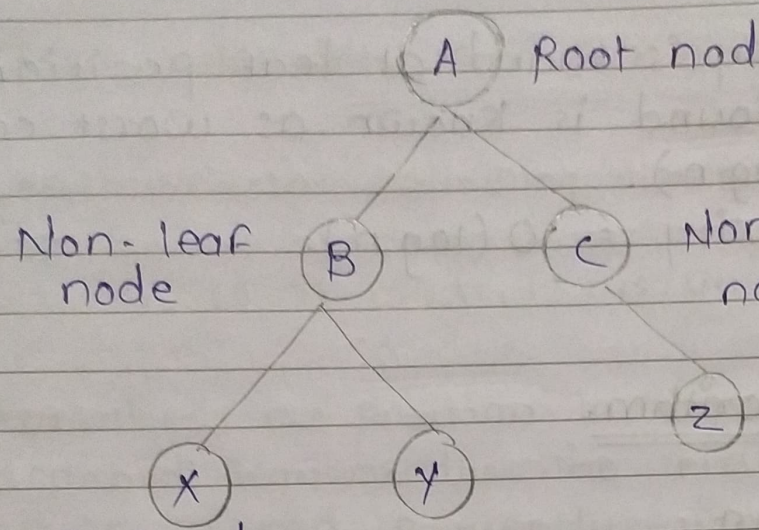

Fig: binary search.

In a binary tree, any element is at either one of the following position.

1) Root pos.
2) Leaf pos.
3) Non-leaf pos.

inorder - L, ROOT, R
preorder - ROOT, L, R
postorder - L, R, ROOT

1) Root pos/ root node.
First position.
node which is not having further child node which means leaf node
node which is having child node is called non-leaf node.

1] If key found at root position is called best case means O(1), no of comparison for input size array = 1.
time complexity = $\Omega(1)$.

2] If key found at non-leaf position is called average case $O(\log n)$,
time complexity = $\theta(\log n)$

3] If either key is found at leaf position or key is not found is known as worst case. means O(log n).
time complexity = O(log n).

* Sorting Algorithm.

1) Selection Sort.

i) Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison based algorithm in which the list divided into two parts at the left end and the unsorted part at the right end. initially sorted part is empty and unsorted part is the entire list.

ii) The smallest element is selected from the unsorted array and swapped with the leftmost element and that element beome a part of sorted array. this process continues moving unsorted array.

iii) The algorithm is not suitable for large data sets as its average and worst case complexities are of $O(n^2)$ where n is the number of item.

* How selection sort works?

Consider the following element in array.

14, 33, 27, 10, 35, 19, 42, 44

For the first position in the sorted list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and then find that 10 is the lowest value.

[14], 33, 27, 10, 35, 19, 42, 44

So, we replace 14 with 10 after one iteration 10, which happens to be the minimum value in the list, appears in the first position of sorted list.

10, [33], 27, 14, 35, 19, 42, 44.

Here 10 got his first position or perfect position for rest of array where 33 is residing, we start scanning the rest of the list in a linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap 33 & 14.

10, 14, 27, 33, 35, 19, 42, 44

After two iteration, two least values are positioned at the beginning in a sorted manner.

Above process is applied to the rest of the items in array.

* Algorithm.

Step 1:
Set MIN to location 0.

Step 2:
Search the minimum element in the list.

Step 3:
Swap with the value at location MIN.

Step 4:
Increment MIN to point to next element.

Step 5:
Repeat until list is sorted.

Pseudocode.

```
procedure : selection sort.
list      : array of items.
n         : size of list.

for i=1 to n-1

  min = i

for j = i+1 to n
   if list [j] < list [min] then
      min = j ;
   end if
  end for

   if indexmin ! = i then.
      swap list [min] and list [i]
   end if
    end for.

end procedure.
```

* Important key points.

1] Selection sort best case $\Omega(n^2)$
2] Selection sort worst case $O(n^2)$
3] Selection sort Average case $\theta(n^2)$
4] In selection sort magnitude of time complexities is same in all cases.

2] Bubble Sorting Algorithm.

i) bubble sort is a simple sorting algorithm This sorting algorithm is comparison-based algorithm.

ii) In bubble sort each pair of adjacent elements is compared and the element are swapped if they are not in order.

iii) (This algorithm is not suitable for large data sets) as the average and worst case complexity are of $O(n^2)$ where n is the number of items.

iv) In this algorithm, in every iteration elements which are two consecutive positions gets compared, if they are already in order then no need of swapping between them.

v) But, if they are not in order i.e if prev position element is greater than its next position, element then swapping takes place, and by this logic in first iteration (largest element get settled at last position. First Iteration)

vi) In second iteration second largest element gets setteled at second last

position and so on, in max (n-1) no. of iteration all elements gets arranged in a sorted manner.

* How bubble sort works?

Consider following element for array.

14, 33, 27, 35, 10

bubble sort starts with very first two elements, comparing them to check which one is greater.

14, 33, 27, 35, 10

In this case, value 33 is greater than 14, so it is already in sorted location next we compare 33 with 27.

14, 33, 27, 35, 10

We find that 27 is smaller than 33 and these two values must be swapped. the new array should look like this,

14, 27, 33, 35, 10

Next we compare 33 and 35 we find that both are in already sorted position.

14, 27, 33, 35, 10

We know that 10 is smaller 35, hence they are not sorted. we swap these values. we find that we have reached the end of the array after one iteration

14, 27, 33, 10, 35

After one iteration, the array should look like above.

To be precise, we are now showing how an array should look like after each iteration after the second iteration, it should look like this.

14, 27, 10, 33, 35

Notice that after each iteration, at least one value moves at the end

After third iteration

14, 10, 27, 33, 35

And when there is no swap required, bubble sort learns that an array is completly sorted.

10, 14, 27, 33, 35.

* Algorithm.

```
begin BubbleSort (list)

    for all elements of list
        if list[i] > list[i+1]
            swap (list[i], list[i+1])
        end if
    end for

    return list.
end Bubble sort.
```

* Pseudocode.

```
procedure bubblesort (list: array of items)
    loop = list.count;
    for i=0 to loop-1 do;
        swapped false

    for j=0 to loop-1 do;

        if list[j] > list[j+1] then
            swap (list[i], list[j+1])
            swapped = true
        end if
    end for
```

```
        if (not swapped) then
            break
        end if
    end for
end procedure return list
```

* Important key points.

1) Bubble sort best case $\Omega(n)$
2) Bubble sort worst case $O(n^2)$
3) Bubble sort average case $\theta(n^2)$
4) Bubble sort is not efficient for already sorted input array by design but it can be implemented efficiently for already sorted input sequence by using logic.
5) Large values are always sorted first.
6) It only takes one iteration to detect that a collection is already sorted.
7) The best time complexity for Bubble sort is $O(1)$
8) The space complexity for bubble sort is $O(1)$ because only single additional memory space is required.

classmate
Date
Page

3] Insertion Sort.

i) This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted.

ii) For example, the lower part of an array is maintained to be sorted.

iii) An element which is to be 'inserted' in this sorted sub list, has to find its appropriate place and then it has to be inserted there. Hence the name is, insertion sort.

iv) The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array) This algorithm is not suitable for large data sets.

v) In this algorithm, in every iteration one element gets selected as a key element and key element gets inserted into an array as its appopriate position towards left hand side element in a such way that element are at left side are arranged in a sorted manner. and so on.

vi) In max (n-1) no. of iteration all array element get arranged in sorted manner.