Unit 1

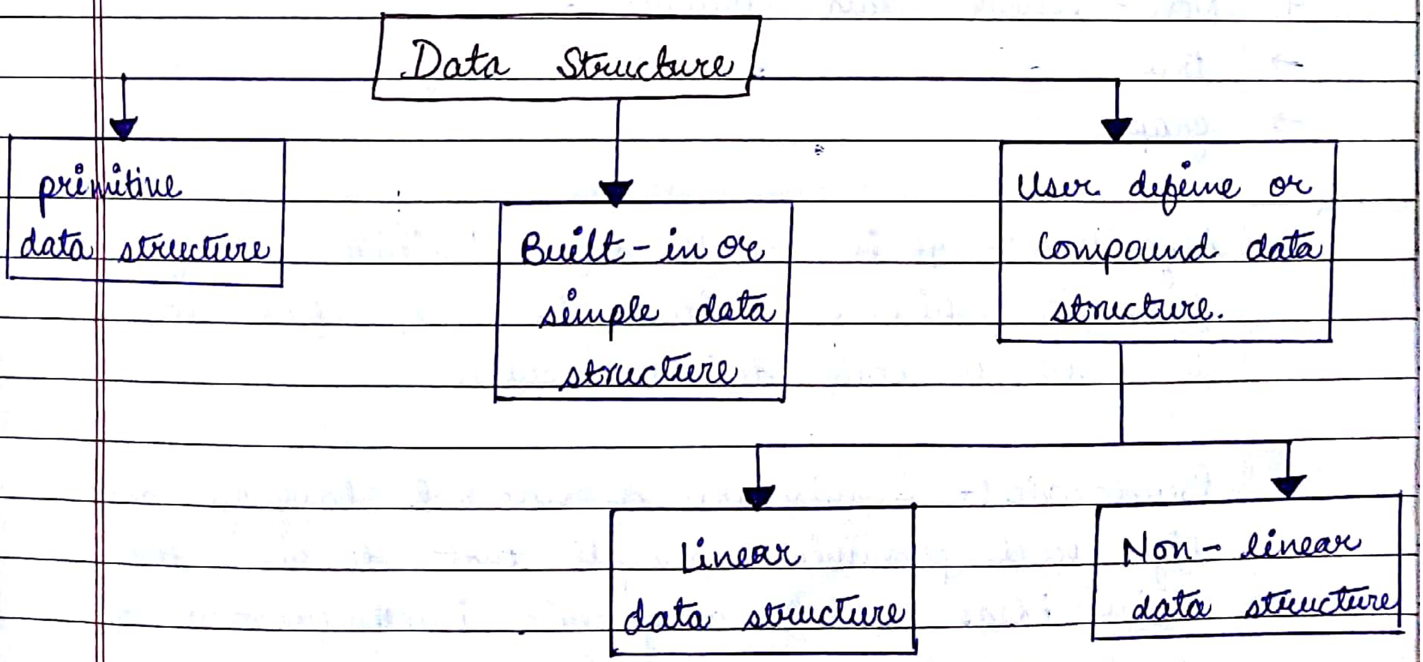
Data structure and Algorithms :-

Data Structure :- The logical or methematical model of a perticular organisation of data is called Data structure. It is a way of organising the data that consider not only the item stored but also there relationship to each other.

Data structure specified two things :-

1. Organised data :- (i.e. data stored + Relationship between data elements)
2. Allowed operations :- (eg:- Insertion, Deletion, transversion).

Categories of D.S. :-



1. Primitive data structure :-
→ Integer
→ floating
→ characters
→ Boolean

2. Built-in or simple data structure :-
→ Array
→ ~~stack~~ string
→ structure

3. Linear Data structure :-
→ Array
→ link list
→ stack
→ Queue

4. Non-linear data structure :-
→ tree
→ graph

X

Algorithm :- It is a well define / finite set of instruction which are executed in a specified sequence in order to obtain desired results.

Pseudocode :- A mixture of natural language and high level programming concepts that describes the main ideas behind a generic implementation of a datastructures or algorithms.

It is more structured than algorithm but less formal then a programming language.

Characteristics of Algorithm :-

1) Input :- Each and every algorithm must take 0 and more input.

2) Output :- Each and every must give 1 or more output.

3) Definiteness :- Each and every instruction must be well defined, Unambiguous.

4) finiteness :- Each and every instruction must be completed in finite amount of time.

5) Effectiveness :- Basic, feasible with help of pen or paper we can easily trace one algorithm.

GCD :- (Greatest common Divisior) :-

Input → 2 integer m, n
Output → Largest integer that divide both.

$m = 36$ $n = 48$

$m = 2 \times 2 \times 3 \times 3$

$n = 2 \times 2 \times 2 \times 2 \times 3$

$GCD = 2 \times 2 \times 3$

$= 12$

Algorithm of GCD.

1. Factorize m

$m = m_1 \times m_2 \times \ldots \times m_n$

2. Factorize n

$n = n_1 \times n_2 \times \ldots \times n_n$

3. Identify common factor and multiply them to get the result.

```
Euclid (m, n)
{
  while m does not divide n.
  {
     r = n ~~does~~ mod m
     n = m
     m = r
  }
  return m
}
```

m = 36   n = 48

36 divides 48

r = 12

n = 36

m = 12

Complexity of an Algorithm:-

1. Time complexity:- The time complexity of an algorithm is the amount of CPU execution time it need to run to completion.

2. Space complexity :- The space complexity of an algorithm is the amount of memory it need to run to complition.

1. Space complexity :-

1. Fixed part
- → instruction space
- → simple variable
- → fixed sized components
- → constants.

2. Variable part
- → Dynamic memory variable
- → Refrence variable
- → Recursive stack.

```
Algo  sum (a, n)
{
      s: = 0
      for i: = 1  to  n do
          s: = s + a [i];
          return s;
}
```

2. Time complexity :-

Types of instruction
- → Read
- → write
- → function call
- → Assignment   x := a / x := 5 ;
- → Arithmatic exp. / calculation   x := x+5 / 6-a;

condition check If (x > 1) / while (i<5)
increment / decrement x++ , y--.

```
for (i:=1 ; i<=n ; i++)      → n+1
  printf ("%d", i);          → n
```

complexity ⇒ 2n+1.

```
Algo sum (a, n)
{
   s:=0;
   for i:=1 to n do
     s:=s+a[i];
     return s;
}
```

s/e (step per execution)	frequency	Total steps
0	—	
0	—	
1	1	1
1	n+1	n+1
1	n	n
1	1	1
0	—	

2n+3

```
Algo R sum (a, n)
{
    if (n ≤ 0) then
     return 0;
    else return (R sum (a, n-1) + a[n]);
}
```

s/e	frequency		Total steps	
	n=0	n>0	n=0	n>0
1	1	1	1	1
1	1	0	1	0
1 + tRsum (n-1)	0	1	0	1+ tRsum(n-1)
			2	2+ tRsum(n-1)

tRsum (n) = 1(2) + tRsum (n-1)
= 2 (2) + tRsum (n-2)
= 3 (2) + tRsum (n-3)
= 4(2) + tRsum (n-4)
= n(2) + tRsum (n-n)
= n(2) + 2   where tRsum(0)=2.

```
Algo Add (a, b, c, m, n)
{
    for i:=1 to m do
        for j:=1 to n do
            c [i,j] := a [i,j] + b [i,j];
}
```

s/e	frequency	Total steps
1	m+1	m+1
1	m(n+1)	m(n+1)
1	m×n	nm
		m+1 + mn + m(n+1)
		2mn + 2m + 1.

-X-

Asymptotic Notation:-

It is a formal way / Notation to speak about functions and classify them.

1. Theta Notation (Θ) → both upper and lower bound

   $\Theta(g)$:- { f/f is a non-negative function such that ∃ constants C_1, C_2 and n_0 such that
   $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for $n \geq n_0$}

2. Big-oh(g):- upper bound

   Big-oh (g):- { f/f is a non-negative function such that ∃ constant C_1, and n_0 such that
   $f_n \leq cg(n)$ for $n \geq n_0$}

3. Omega(g):- Lower bound

   omega(g) :- { f/f is a non-negative function such that ∃ constant C and n_0 such that
   $f_n \geq cg(n)$ for $n \geq n_0$}

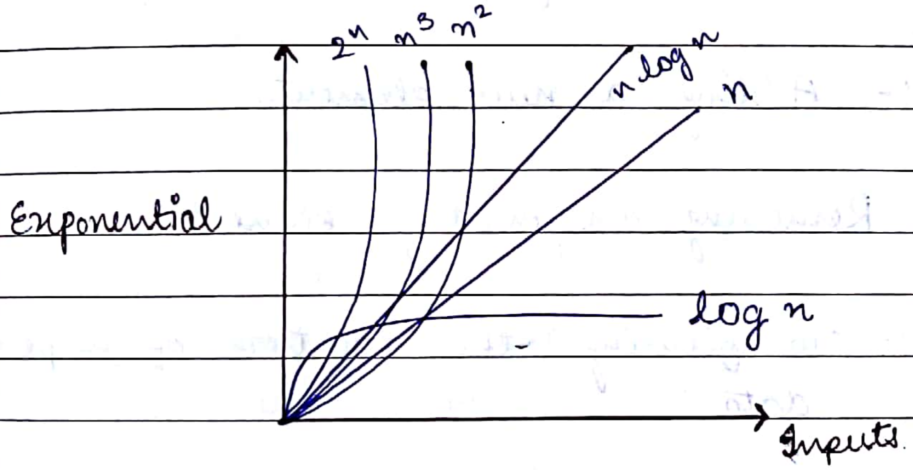
—x—

Growth of function:-

constant time $O(1)$ algo.
Logrithmic time $O(\log n)$ algo.
Linear time $O(n)$ algo

polynomial time $O(n^k)$ algo
Exponential time $O(K^n)$ algo

$\log n$	Input n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



Abstract :- It is a logical discription of how we view the data and operations that are allowed without regard to how they will be implemented. It decribes properties and operation done on that type data.

Array :- Array is a collection of similar type of data in which elements are stored in contigious memory location.

Where elements are retrive with the help of indexs.

Marks

5	15	20	3	8	7	6

(1) Name of the array (Name)
(2) Type of elements stored in the array (Data Types)
(3) Size of array / Max no. of element to be stored (Index Set).

for example :- int Marks[10];

Operations :-

1. Traversing :- Processing each element / visiting each and every element one by one.

2. Insertion :- Adding a new element.

3. Deletion :- Removing an existing element.

4. Searching :- To finding the location of a particular data.

5. Sorting :- Arranging in data in either ascending or decending.

6. Merging :- Combine two array into a single array.

X

1. Traverse an array :-
Here LA is a linear array with lower bound LB and upper bound UB.

algo :-

1. Set K := LB
2. Repeat step 3 & 4 K < UB
3. Apply process to LA [K]
4. Set K := K+1.
5. Exit.

2. Inserting an array :-

LA is linear array of N element and K is the integer such that K ≤ N.

This algorithm inserts an element (ELE) into the K^th position in LA.

algo.

Insert (LA, N, K, ELE)

1. Set J := N
2. Repeat steps 3 & 4 while J ≥ K
3. Set LA [J+1] := LA [J]
4. Set J := J-1
5. Set LA [K] = ELE
6. set N := N+1
7. Exit.

3. Deletion an array :-

algo.

Delete (LA, N, K, ELE)

1. Set ELE := LA [K].
2. Repeat for J := K to N-1.
3. Set LA[J] := LA [J+1]
4. Set N := N-1
5. Exit.

4. Reverse an array :-

Reverse (arr,
1. Set I:=1 and I:=N
2. repeat step 3 to 7 while (I>J)
3. Swap arr [I] and arr [J]
4. (i) Temp := arr [I]
5. (ii) Arr[I] := arr [J]
6. (iii) Arr [J]:= Temp
7. (iv) Set I := I+1
8. (v) Set J:J-1
9. Print arr.

---

5. Searching an array :-
Two types of searching
(i) Linear search
(ii) Binary search.

(i) Linear searching algorithm:-
Linear (DATA, ITEM, N, LOC)

1. Set LOC := 1.
2. Repeat while DATA [LOC] ≠ ITEM and LOC ≤ N)
Set LOC := LOC +1
3. If LOC = N+1
set LOC := 0
4. print LOC
5. Exit.

(ii) Binary searching algorithm :-

★ Sorted data

BINARY (Data , LB , UB , Item , LOC)

1. Set Beg := LB , End := UB and mid := ⌊(Beg + End)/2⌋
2. Repeat steps 3 & 4 while (Beg ≤ End and DATA [mid] ≠ Item)
3. If ITEM < DATA [Mid] then
   Set End := mid − 1
   else
   set beg := mid + 1
4. Set mid := ⌊(Beg + End)/2⌋
5. If DATA [mid] = Item then
   Set LOC := mid
   else
   set LOC := Null
6. Print LOC
7. Exit.

—x—

6 Sorting an array :-

Bubble sort :-

Bubble (Data , n)

1. Repeat steps 2 & 3 for pass := 1 to n−1.
2. Set ptr := 1
3. Repeat while (ptr ≤ n − pass)
   (a) If data [Ptr] > data [Ptr + 1] , then
   {
   temp := data [Ptr]
   data [Ptr] := data [Ptr + 1]
   data [Ptr] := temp
   }

(b) Set Ptr := ptr + 1.
4. Exit

Strings:- A string is a sequence of characters. In computer science, strings are more often used than numbers. We have all used text editors for editings programs and documents. Some of the important Operations which are used on strings are: searching for a word, find and replace operations etc.

string operation:-

1. strlen () → Find length of the string.
2. strcpy () → Copy one string into another.
3. strcat () → Append one string after another.
4. strcmp () → Compare 2 string.
5. strrev () → Reverse a string.

1. Find the length of a string:-

Algo.

1. Set I := 1
2. Repeat while str [I] != '\0'
   I := I + 1
3. Print I − 1.
4. Exit.

2. Copy one string into another string:-

Algo

1. Set I := 1
2. Repeat step 3 & 4 while str [I] != '\0'
3. Str2 [I] := Str1 [I]
4. I := I + 1

5. Set Str 2 [I] := '\0'
6. Print str 2
7. Exit.

3. Append / add / concenterate two string:-
Algo.
1. Set l1 = strlen (str1) and l2 = strlen (str2)
2. Repeat for i := 1 to l1
   str 3 [I] : = str1 [I]
3. Set str 3 [I] = ' '
4. set I := I + 1 and J := 1
5. Repeat while (J <= l2)
   {
   Str 3 [I] := str2 [J]
   I : = I+1
   J : = J+1
   }
6. Str 3 [I] : = '\0'
7. Print str 3
8. Exit.

4. Compare two string:-
Algo.
1. Set l1 = strlen (str1) and l2 = strlen (str2).
2. If (l1 > l2) then
   set l := l1
   else
   set l := l2.
3. Repeat for I := 1 to l do
   if (str1 [I] != str2 [I])
   break;

4. Set C:= str1[I] − str2[I]
5. If (C>0) then
   print "str1 is greater than str2"
   else if (C<0) then
   print "str1 is smaller than str2"
   else
   print "str1 and str2 is equal"
6. Exit

5. Reverse a string:-
Algo.
1. Set len:= strlen(str)
2. Set I:=1 to J:= len
3. Repeat step 4 to 6 while (I<J) do
4. swap str[I] and str[J]
5. I:=I+1
6. J:=J+1,
7. Print str
8. Exit

6. Merging of an array:-
Algo.
1. Set I:=1, J:=1 and K:=1
2. Repeat step 3 to while
3. If A1[I] < A2[J] then
   {
   A3[K]:=A1[I]
   I:=I+1
   }
   else
   {A3[K]:= A2[J]

J:=J+1
}

4. If (I > n1) then
Repeat while (J ≤ n2)
{
A3[K] := A2[J]
J := J+1
K := K+1
}
else
{
Repeat while (I ≤ n1)
{
A3[K] := A1[I]
I := I+1
K := K+1
}

5. Repeat for K:=1 to n1+n2
Print A3[K]

6. Exit.

---

6. Pattern Matching in string :-

Algo.

1. Set K:=1 and Max := S-R+1
2. Repeat step 3 to 5 while K ≤ Max do.
3. Repeat for L:=1 to R do
If P[L] ≠ T[K+L-1], then
Go to step 5
4. Set INDEX := K and exit.
5. Set K := K+1
6. Set Index := 0
7. Exit.

Multidimentional array:-

Pointer and pointer arrays:-

Algo.

```
int i = 5;
int *ptr = &i;
```

1. Set First := group [L]
   Last := group [L+1] -1
2. Repeat for K := first to last
   print event [K];
3. Exit.

T –
Y =

—X—