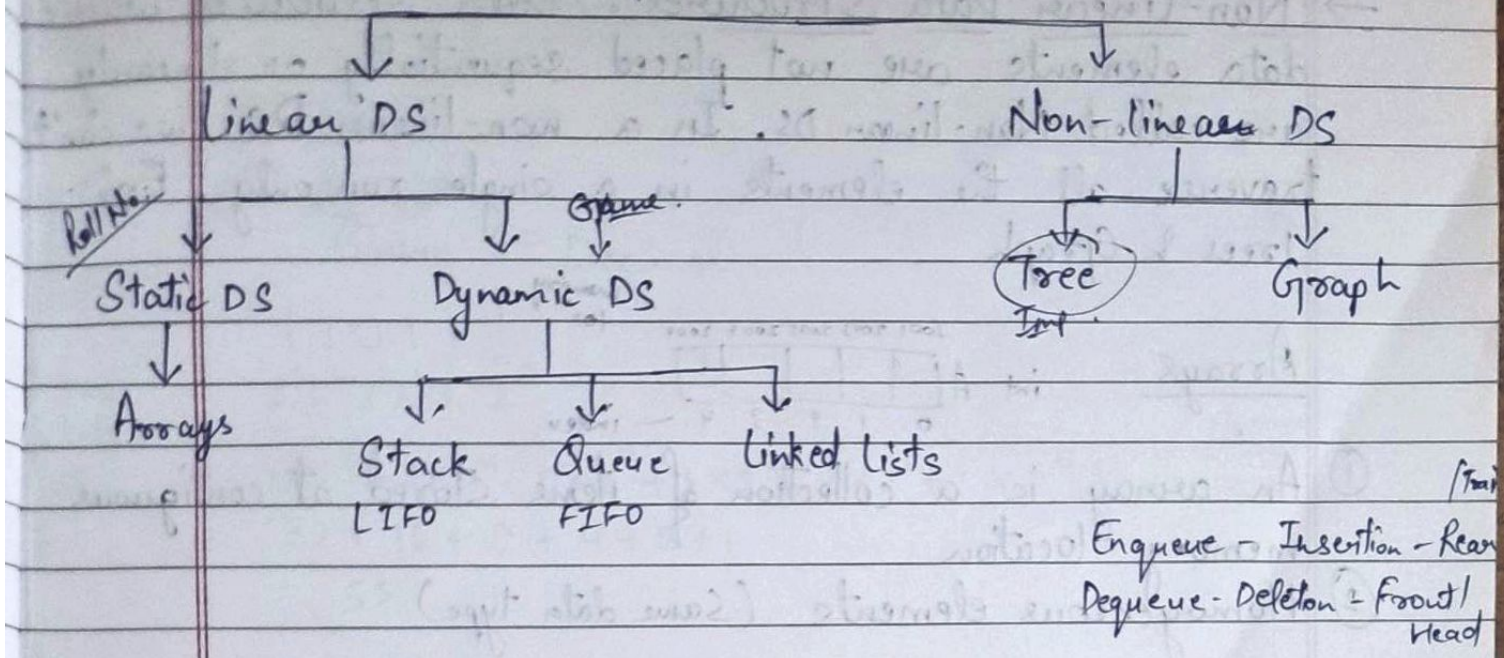
Data Structures

Page No.
Date :



Enqueue - Insertion - Rear
Dequeue - Deletion - Front/Head

* Data Structure :- It is a storage that is used to store an organised data. It is used for processing, retrieving & storing data. It is a way of arranging data on a computer so that it can be accessed & update efficiently.

→ Linear data structure :- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous & next adjacent element. Ex :- Arrays, Stack, Queue, Linked lists.

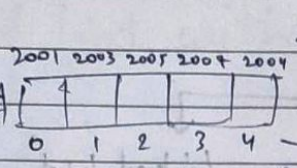
① Static Data Structure :- It has a fixed memory size. It is easier to access the elements in a static data structure. Ex :- Arrays (Roll No.).

② Dynamic Data Structure :- In dynamic DS, the size is not fixed. It can be randomly updated during the run time which may be considered efficiently.
Ex :- Games

Page No.
Date :

→ Non-linear Data Structures :- Data structures where data elements are not placed sequentially or linearly are called Non-linear DS. In a non-linear DS, we can't traverse all the elements in a single run only. Ex:- Trees & Graph

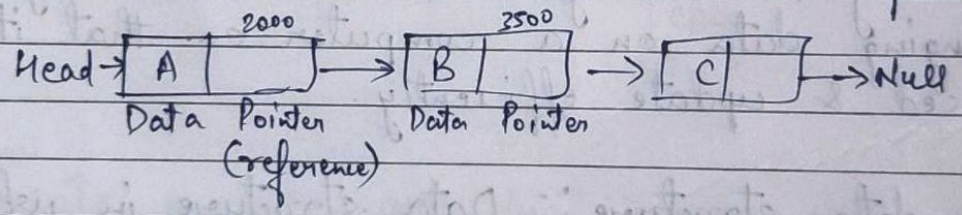
Arrays



① An array is a collection of items stored at contiguous memory location
② Homogeneous elements (Same data type)

→ Index :- Each value is uniquely identified by index.

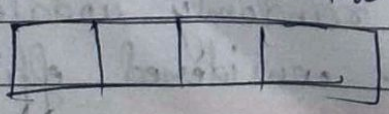
→ Linked lists :- A linear DS in which the elements are not stored at continuous memory location. The elements in a linked lists use pointer



→ Stack :- ① A stack is a linear DS which follows LIFO (Last In First Out) (Ex:- Bottle)
② Insertion :- Push
Deletion :- Pop
③ Happen on the same end (Top).

→ Queue :- A queue is a linear DS that is open at both ends and operations are performed in FIFO (First in first Out)

Enqueue -> Insertion -> Rear / Trail / back
Dequeue -> Deletion -> Front / Head

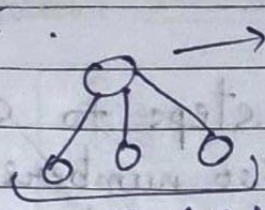


Date 23/9/22

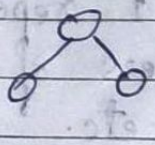
* Non-linear Data Structure

↳ Tree

↳ Graph

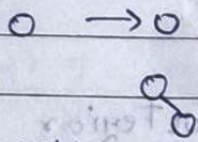


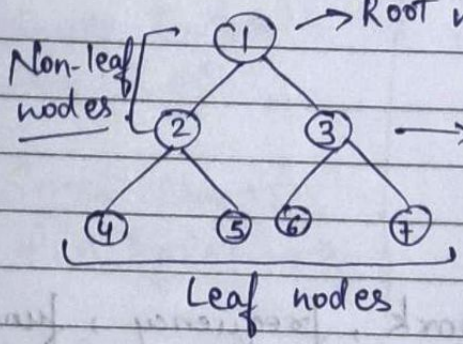
3 children → Ternary Tree



Binary Tree → Atmost 2 childr

0, 1, 2





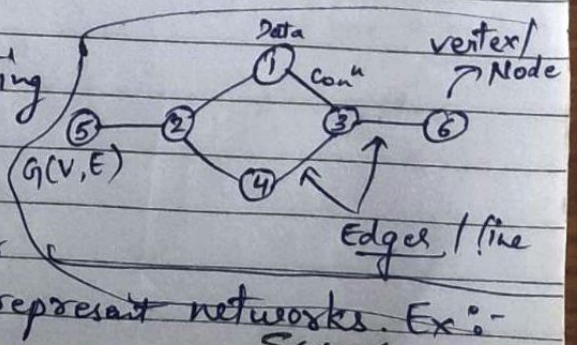
Tree

① Binary Tree is a tree data structure (non-linear) with atmost two children (0, 1, 2). Since each element in a binary tree can have only 2 children; we typically name them left child, right child.

Basic operation of binary tree: insertion, deletion, searching & traversing

Graph

A graph is a non-linear DS consisting of vertices (nodes) & edges (lines). The graph is denoted by G(V, E). Every vertex and edge can be labelled or unlabelled. Graphs are used to represent networks. Ex:- Facebook network.



Ex:- priory
S_1 : Read A
S_2 : Read B
S_3 : Sum = A+B
S_4 : Print(Sum)

Date:

* Algorithm

→ Finite set of steps to solve a problem.
→ Ex:- Sum of two numbers. We can write code in any programming language by using algorithm.
→ Unambiguous :- Don't repeat instructions.
→ Analysis is a process of comparing two algorithms w.r.t time, space etc.

Analysis
- Priory → Before execution
- Posterior → After execution
  - i_3 → 0.4s
  - i_5 → 0.2s
  - i_7 → 0.1s

  (processor)

Priory
- How many times an instruction work, frequency, func^n calling
- Ex:- Fact(n) → Recursion (how many times f^n is calling)

1) Approximate value
   → Independent of HW
   → uniform value

Posterior
2) Exact value
   → Dependent
   → non uniform value

Asymptotic Notations.

Date 28/9/22.

* Various Properties of Asymptotic Notations.

	Reflexive $x=x$	Symmetric $x=y, y=x$	Transitive $x=y, y=z, x=z$
Big Oh (O) $f(n) \le g(n)$ $x \le y$	$1 \le 1$ ✓	$3 \le 4$ $4 \le 3$ ✗	$3 \le 4 \le 5$ ✓
Big Omega (Ω) $f(n) \ge g(n)$ $x \ge y$	$1 \ge 1$ ✓	$4 \ge 3$ $3 \ge 4$ ✗	$5 \ge 4 \ge 3$ ✓
Theta (Θ) $g(n) \le f(n) \le g(n)$ $x=y$	$1 = 1$ ✓	$4 = 4$ $4 = 4$ ✓	$3 = 3 = 3$ ✓
Small Oh (o) $f(n) < g(n)$	$1 < 1$ ✗	$2 < 5$ $5 < 2$ ✗	$6 < 7 < 8$ ✓
Small Omega (ω) $f(n) > g(n)$ $x > y$	$1 > 1$ ✗	$5 > 2$ $2 > 5$ ✗	$8 > 7 > 6$ ✓

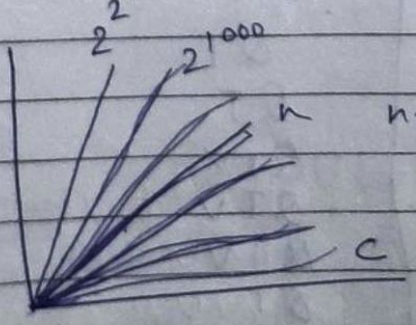
* Comparison of Various Time Complexities.

$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$

n^k: polynomial time; 2^n: exponential time; n^n: factorial time; 2^{2^n}: Double exponential time.

Ex:- $O(10^3)$ or $O(10^0)$

$O(1) = O(1)$



$\log_{10} 100 = 2$.

$10^2 = 100$

$\log_2 8 = 3$.

$\log_2 1000$

* Infix to Postfix conversion using Stack.

Infix → A+B

Postfix → AB+

Prefix → +AB
Operator → operand

Precedence order	Associativity
( )	^ → Right to left → push
^	/* → left to Right
/*	+- → Left to Right → pop & push
+-	

11/10/22

Higher priority → Push
lower priority → Pop & Push

* Infix to Prefix conversion.

K + L - M * N + (O ^ P) * W / U / V * T + Q

Q + T * V / U / W *) P ^ O (+ N * M - L + K)

Input	Stack	Prefix expression
Q		Q
+	+	Q
T	+	QT
*	+*	QT
^	+*	QTV
/	+*/	QTV
U	+*/	QTVU
/	+*//	QTVU
W	+*//	QTVUW
*	+*//*	QTVUW

^ → pop & push
/* - push
+- - push

Spiral

Date .....

)	+*//*)	&TVUW
P	+*//*)	&TVUWP
^	+*//*)^	&TVUWP
O	+*//*)^	&TVUWPO
C	+*//*	&TVUWPO^
+	+ pop +	&TVUWPO^*//*
N	+*//*+	&TVUWPO^N
*	+*//*+*	&TVUWPO^N
M	+*//*+*	&TVUWPO^NM
-	+*//*+*-	&TVUWPO^NM
L	+*	
N	++	&TVUWPO^*//*N
*	++*	&TVUWPO^*//*N
M	++*	&TVUWPO^*//*NM
-	++-	&TVUWPO^*//*NM*
L	++-	&TVUWPO^*//*NM*L
+	++-+	&TVUWPO^*//*NM*L
K	++-+ pop	&TVUWPO^*//*NM*LK

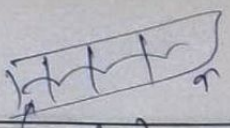
&TVUWPO^*//*NM*LK+-++

Reverse

++-+KL*MN*//*^OPWUVT&

Spiral

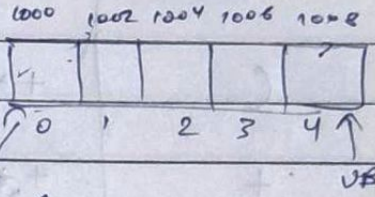
Date 10/10/22.



* Arrays.

An array is a data structure which is used to store homogeneous elements at contiguous memory location.

1. Homogeneous elements
2. Finite → Fixed size
3. Contiguous memory location
4. Ordered set → ordered index.
5. Random Access



Size of array = UB - LB + 1
= 4 - 0 + 1
= 5.

Arrays

One-dimensional

Multi-dimensional

2D → Matrix, Excel Sheet

3D → Games.

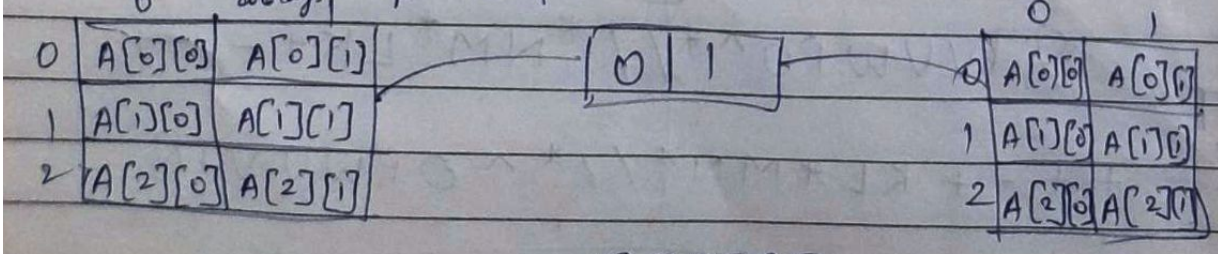
2D.

	0	1	2
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]

Total no. of elements
= No. of R × No. of C
= 2 × 3
= 6.

A[2][3]
- array name
- No. of rows
- No. of columns

3D. A[2][3][2]
- No. of arrays
- No. of R
- No. of C



A[0][1][1]

Spiral

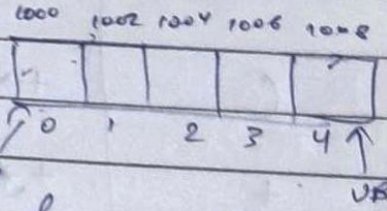
Date 10/10/22.

* Arrays

O(n)
O(1) → random access

An array is a data structure which is used to store homogeneous elements at contiguous memory location.

1. Homogeneous elements
2. Finite → Fixed size
3. Contiguous memory location
4. Ordered set → ordered index.
5. Random Access



Size of array = UB - LB + 1
= 4 - 0 + 1
= 5.

Arrays

One-dimensional

Multi-dimensional

2D → Matrix, Excel Sheet

3D → Games

2D:

	0	1	2
0	A[0][0]	A[0][1]	A[0][2]
1	A[1][0]	A[1][1]	A[1][2]

A[2][3]
- array name
- No. of rows
- No. of columns

Total no. of elements
= No. of R × No. of C
= 2 × 3
= 6.

3D:

A[2][3][2]
- No. of arrays
- No. of R
- No. of C

	0	1
0	A[0][0]	A[0][1]
1	A[1][0]	A[1][1]
2	A[2][0]	A[2][1]

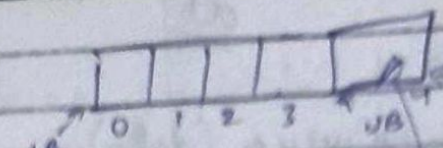
0 | 1

	0	1
0	A[0][0]	A[0][1]
1	A[1][0]	A[1][1]
2	A[2][0]	A[2][1]

A[0][1][1]

Date .....

* Insertion in Arrays



N = 4
N + 1 = 5
UB = 4
A[UB]

TC = O(1)

Insertion At End

```
INSARRAY (A, N, data)
1. If UB = N-1
   then print overflow & Exit
2. Read data
3. UB = UB + 1
4. A[UB] = data
5. Stop & Exit
```

Traversing

```
for (i = 0 ; i < n ; i++)
{ printf (" A[%d]", A[i]) ;
  }
```

* Queue.

- Enqueue()
- Dequeue()
- isfull()
- isEmpty()
- peek()

```
begin procedure peek
    return queue[front]
```

```
begin procedure is full
if rear equals to MAXSIZE
    return true
end if
else
    return false
end procedure
```

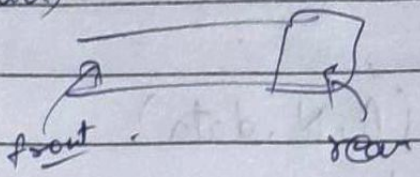
```
begin procedure is empty.
if front is less than MIN
    return true
end if
else
    return false
end procedure
```

Spiral

Enqueue

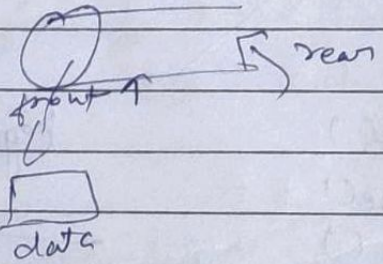
O(1)

```
procedure enqueue (data)
if queue is full
    return overflow
end if
rear ← rear + 1
queue[rear] ← data
return true
end procedure
```



Dequeue

```
procedure dequeue
  if queue is empty
     return underflow
  end if
  data = queue[front]
  front ← front + 1
  return data
  end procedure
```



* Applications of stack

1) Polish Notations (Prefix, Postfix, Infix)
2) Recursion

→ Recursion is the ability of a procedure either to call itself or to call other procedure which may result in call to original procedure.

→ Recursion is generally used for repititive combinations in which each action is defined in terms of previous results.

Spiral

Date .....

Properties of Recursion

① There must be decision criteria that stops the further call to the procedure, call base criteria/condition.

② Each time, a procedure calls itself either directly or indirectly, it must be nearer to the solution that is base criteria.

```
main()
{
  fact();
}
fact()
{
}
```

function calling

```
void fact()
{
  fact();
}
```

stack overflow