Data Structure

Page No.
Date

- Data Structure :- Data structure is logical & Mathematical Model of storing & organizing data in a particular way that it can be required for design & emplimentation of algorithm

BookMark

* Basic Terminology

1. Data :- set of values / value
2. Data items :- Single unit of value
3. Record :- Collection of various data item
4. File :- Collection of record of one type
5. Field :- Single elementary unit representing Attribute of a entity
6. Information :- Data with attributes or meaningfull data.

Classification of Data Structure (Ds)

Primitive Ds
- int, float, char, dolible, boolean, etc.
- Directly interact with the system or Machine / Hardware part
- Primitive Ds are the Ds whose property or operation aldeready known to the compiler or computer system.

Non-Primitive Ds.
- derived from primitive Ds.
- Non-Primitive Ds not directly interact with Machine
- With the help of Primitive Ds it can be interact with the machine

Linear Non-Primitive Ds
- Array
- Linked list
- Stack
- Queue

Non-linear non....
- Tree
- Graph

Page No.
Date

2-3M

Book Mark →

Linear - Non Premitive Ds	Non-linear Non Primitive
① All element are arrange in linear Order. Each element has Successor & Predecessor except First & last element ex :- Array.	① This data structure does not form a sequence ex :- Tree
② Single level involve	② Multi level involve
③ Data element in single run traversed	③ Can't be traversed in single run.
④ Use in Software development	④ Use in AI ; DIP (Artificial Intelligence ; digital image processing)

☆ Data structure Operations :-

① Traversing
② Searching ← linear, binary
③ Inserting
④ Deleting
⑤ Sorting
⑥ Merging

① Traversing :- Visiting each & every element at least once or exactly once.

ex:- Array

10	20	30	40		
1	2	3	4	5	6

Index →

Algorithm:- St-1 Read [A16]
St-2 - for k=1 to K≤N
St-3 - display [A16]
St-4 :- exist.

```
#include<stdio.h>
void main()
{
int A[7] ; {10,20,30,40,50};
for (i=1 ; i<=7 ; i++)
 {
  pf ("%d", A[i])
 }
}
```

big of n - O(n)

③ Inserting :- Adding an element to linear array.

There are three cases

① Insert an element at begining
② Insert an element at end
③ Insert an element at given location

i) Iseet an element at begining :-

Time comp.

O(1) - N — best case

10	20	30	40	50		
1	2	3	4	5	6	7

item 11

O(N) - N+1 — worst case

11	10	20	30	40	50	
1	2	3	4	5	6	7

K++;

K=3

item 11

O(n) - Caverage case.

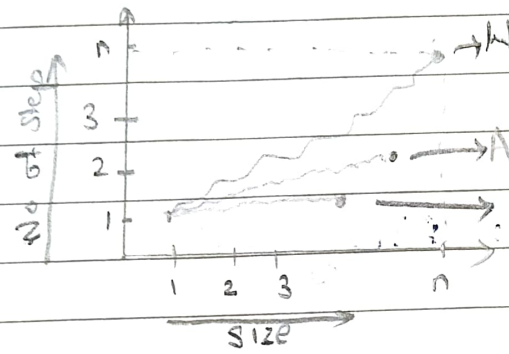
10	20	11	30	40	50
1	2	3	4	5	6

Page No.
Date

Analysise of Algorithm < memory - Space complicity
Time - Time complicity

① Best case :- Minum steps for instant n.
② worst :- Max^m step — /
③ Average :- Avg. step — /

* Efficiency Algorithm :- less time + less memory



$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < n^n$

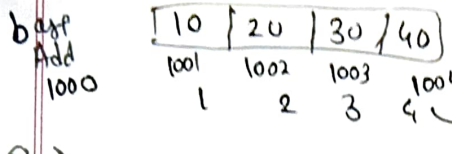
Best case Average case worst case

Notation → Space complexity
Time complexity
> Asymptotic Notation
Big oh (O)

* Algorithm - Insertion (Index location)

Insert (A, N, K, item)
Array · total no. · which Atom to insert

Step :- ① Set J = N
② Repeat step ③ & ④ while while while J >= K
③ set A[J+1] = A[J]
④ J = J-1
⑤ set A[K] = item
⑥ Set N = N+1
⑦ exit.



Page No.
Date

Goal e→

Q1) A {-50 , ... 50}
lower bound (lb) upper bound (ub)

Base Address = 999

Size of an element = 10 bytes

find location L(A/49)

→ formula ⇒ $A(l) = b + w(i - lb)$
base Address, size of element, lb

$= 999 + 10(49 - (-50))$

$= 999 + 10(99)$

$= 999 + 990$

$= 1989$

Q2) Find out location A {1, ... 100}

Base Address 1000

Size of element = 4 bytes, L(A/50)

→ $A(l) = b + w(i - lb)$

$= 1000 + 4(50 - 1)$

$= 100 + 4(49)$

location $= 1196$.

Page No.
Date

Code for Insertion, Deletion

```
#include <stdio.h>
  void insert (int [], int);
  Void main ()  Void delete (int a[], int len)
   {
   int a[10], len, ch;
  Pf ("enter no. of elements you want in array:");
  Sf ("%d", &len);
  pf ("Enter element :");
    for (i=0; i<len; i++)
   {
   Sf ("%d", &a[i]);
   }
   do
   Pf ("which operation you want to perform");
   pf ("1 Insertion");
    pf ("2 deletion");
    pf ("3 Display");
    pf ("4. Exit");
     sf ("%d", &ch);
    switch (ch)
     {
    case 1: insert (a, len);        Insertion
        break; case2: deletion ( (a, len));   deletion
        break;
    Case 3: Display ();
          breaks;

    Case 4: Pf ("Exit");
          break;
```

ch → choise

```
default : pf ("Invalid choise");
          break;
        }
While (ch! =4);
```

Insertion —

```
Void insertion(int a[]; int len)
{
int i, pos, num;
pf (" which element you want to insest");
Sf ("Enter post n");
 Sf ("%d", &pos);
for (i = len-1 ; i >= pos; i--)
{
a [i+1] = a[i] ;
}
a [pos] = num;
len ++;
pf("Array after insertion");
 for (i=0; i< len; i++)
 {
 pf ("%d \t", a[i]);
    }
```

deletion —

Start ... ✱

```
Void delete (int a[], int len)
{
int pos ,i;
pf ("enter post n of a element you want
     to delete");
Sf ("%d", &pos);
for (i=pos; i < len-1 ; i++)
```

```
{
a[i] = a[i+1];
}
pf("Array After deletion: ");
len = len-1;
for(i=0; i<len; i++)
{
pf("%d\t", a[i]);
}
}
```

- Linear search & Binary search ~~/sequen~~

1) Linear seach / Sequencial search

Time complexity

	a	10	20	30	40	50	60
		0	1	2	3	4	5

⇒ Best case O(1) Worst case O(n) Average case O(n)

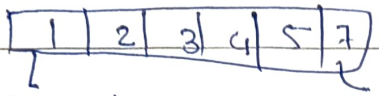
Algorithm :- L Search (A, N, item)
step 1 :- Repeat step 2 for i=0 to N-1
step 2 :- If A[i] == item
{
loc = i;
break;
}
step 3 :- If (i==N)
pf ("Element not found")
else
pf ("Element found at loc")
Step 4 :- exit

2) Binary search (also called divide & conquer)

↳ Array must be sorted

Time complexity

Best case: $O(1)$ Worst case: $O(\log n)$ Average: $O(\log n)$



Lower bound of Lower

Upper bound of Upper

#imp- data structure में जहाँ log होगा, तो वहाँ $\log_2$ (base 2)

$\frac{n}{2^x} = 1$

$n = 2^x$

$\log n = \log 2^x$

$\log_2 n = x \log_2 2$

$\log n = x$

Algorithm :- B search (A, low, high, item)

Step 1 : mid = (low + high)/2

Step 2 :- Repeat step 3 & 4 while low ≤ high & a[mid] ≠ item

Step 3 :- if (item < a(mid)) then

set high = mid - 1

else

set low = mid + 1

Step 4 :- set mid (low + high)/2

Step 5 :- if a[mid] == item then

set loc = mid

step 6 :- exit.

Imp Question There are 16 element in array how many comparison will the element get searched

→ $\log n = \log 16 = \log_2 16 = 4 \log 2$

↳ ∴ 4 no. of comparison

Linear & Binary

```
#include<stdio.h>
void main()
{
int a[25], high, low, item, n, num, i, ch, mid, fea;
pf("1. Linear search \n");
pf("2. Binary search \n");
pf("Enter the choice : ");
sf("%d", &ch);
if(ch==1)
{
pf("Enter the no. of element in the array : ");
sf("%d", &n);
pf("Enter the sorted array : ");
for(i=0; i<n; i++)
sf("%d", &a[i]);
pf("Enter the item to be searched : ");
sf("%d", &item);
for(i=0; i<n; i++)
{
if(a[i]==item)
{
pf("item found at posth %d", i+1);
break;
}
}
if(i=n)
pf("item not found");
{
if(ch==2)
{
pf("Enter the no. of element in the array");
```

```
sf("%d",&n);
pf("Enter the sorted array:");
for(i=0;i<n;i++)
  sf("%d",&a[i])
pf("item to be searched:");
sf("%d",&item);
low=n-1;
mid=(high+low)/2;
while(high<=low)
{
if(item==a[mid])
{
pf("item found at postion %d",mid+1);
break;
else if(a[mid]>item)
low=mid-1;
else high=mid+1;
mid=(high+low)/2;
}
}
}
```

O/P

1. Linear search
2. Binary search

enter the choice: 1
enter the no. of element in the array: 5
enter the sorted array: 1, 2, 3, 4, 5
enter the item to be search: 4
item found at position 4

1. Linear Search
2. Binary sear...

Unit-2 Stack

Page No.
Date

★ ADT Stack :-
ADT (Abstract Data type)

Push()
Pop()
peek()
display()

Imp MCQ - Stack Principle → LIFO or F.I.L.O.
Last in first out First in Last out

Stack { overflow, Is Empty, underflow }

Peek = top most element.

Time complexity

	Best case	Worst case	Average case
Push	O(1)	O(n)	O(1)
pop	O(1)	O(1)	O(1)
peek	O(1)	O(1)	O(1)

Defn - ADT :- Abstract collection of data elements and their accessing function where their are not concerned about how the accessing function will be implemented. It refer to as ADT.

Also we are not concerned with time & space complexity at abstract level

Eg:- Array

explanation :- The result of operation like insertion, deletion, search, sort we will get, but how it process behind their we will not come to know.

ADT stack :- Stack is an ordered list of similar type of element in which an element may be ~~to~~ inserted or deleted only at one end called as top

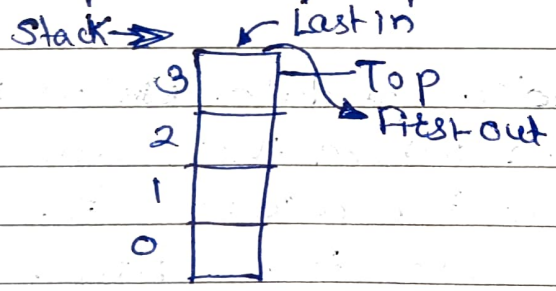
Imp # It follow the Principle LIFO or FILO

last in first out First in last out

Imp Interview — Example of stack :- Stack of chair, Stack of DVDs, Stack of plates, stack of browser page

→ Push() & pop() are used to in stack (Function)

→ Top always point to last element of the stack.



Peek algorithm →

Begin
if top = -1 then stack empty
item = stack [top]
return item
End.

{globally declared → main से बाहर लिखना.

Page No.
Date

Algorithm of push, Pop, peek →

① Push →

Step 1 :- check whether stack is Full or Not.
(overflow) Step 2 :- otherwise, increment the top by 1.
step 3 :- Insert the element.
stack [top] = element (insert)

② Pop →

Step 1 : check whether stack is empty or not
i.e if (top == -1)
(underflow) step 2 :- otherwise, decrement top by 1.

```
#include<stdio.h>
#define MaxSize 5
  void push();
  void pop();
  void peek();
  void display();
  int stack [MaxSize];   int top = -1, inp_array [SIZE];
  int top = -1;
  void main()
  {
  int ch;
  do {
   pf ("which operation do you want to perform: ");
   pf ("1. push");
   pf ("2. pop");
   pf ("3. peek");
   pf ("4. display");
   sf ("%d", &ch);
```

```
Switch (ch)
 {
case 1: push ();
  break;
case 2: pop();
  break;
case 3: peek();
  break;
case 4: display;
  break;
case 5: pf ("exit");
  break;
default pf ("invalid choice");
}
} while (ch! =5 );
  }

Void push()
  {
  int x;
  if (top == size-1)
  {
   pf ("\n overflow !!");
  }
  else
  {
  pf ("\n Enter the element to be added
    onto the stack :");
  sf ("%d", & x);
  top = top+1;
```

```
inp_array [top] = x;
}
}
void pop()
{
if (top == -1)
{
pf ("\n underflow!!");
}
else {
pf ("\n Popped element : %d", inp_array [top]);
top = top - 1;
}
}
void peek()
{ int i;
if (top - i + 1 <= -1)
{
pf ("stack Underflow\n");
return,
}
else
{
return stack [top - i + 1]);
}
}
```

```
void display ()
  {   int i;
if (top == -1)
{
 pf ("\n Underflow!! ");
}
else
 {
pf ("\n Elements present in the stack: \n");
for (int i = top; i >= 0; --i)
printf ("%d \n", inp.arr[i]);
 }
 }
```

Page No.
Date

☆ Applications of stack :-

① Conversion of Arithmetic Expression :-

Three types of Expression.

a + b

operand / operator

ⓐ Infix Expression

ⓑ Prefix expression (polish notation)

ⓒ postfix expression (Reverse Polish notation)

- 5 - operators

↑ → Exponential operator

* / → multiplication / division

+ / - → addition / subtraction

Precidence ⇒ ① ( )

highest ② ↑

not highest ③ * /

lowest ④ + / -

Exam

Conversion

ⓐ Infix Expression

Syntax

Operand1 operator Operand 2

ex:- a + b

ex:- (x - y) - (p + q)

A - B

ⓑ Prefix Expression

Syntax

Operator operand 1 operand 2

ex:- + a b

Ex:- - A B

- - xy + pq

ⓒ Post-fix Expression.

Syntax :- Operand 1 operand 2 Operator

ex:- ab+

AB-

XY - PQ + -

Page No.
Date

Conversion (6 type)

Gate Exam Ques

Imp ① Infix to postfix Using Stack

- Algorithm for Infix to postfix using stack =

Step 1 :- Push '(' on to the stack & ')' to end of expression.

Step 2 :- Scan the expression left to right and repeat step 3 to 6.

Step 3 :- If an operand encountered, add to postfix expression

Step 4 :- If left parenthesis encountered push to stack

Step 5 :- If operator encountered, then

(a) repeatedly pop from stack add to postfix expression for each operator same or highest precedence.

(b) Add operator to stack

Step 6 :- If right parenthesis encountered then

(a) Pop from stack & add to postfix expression until left parenthesis

(b) remove left parenthesis

Step 7 :- Exit.

- Pre/Post - Not Bracket
- Infix - Bracket

Page No.
Date

Q. Imp

Eg:- ① $\underbrace{(A * B)}_{P} + \underbrace{(C * D)}_{q}$

$\{ pq+ \quad AB*CD*+ \}$

Symbol	Stack	postfix exp.
	(	
(	((	
A	((	A
*	((*	A
B	((*	AB
)	(	AB*
+	(+	AB*
(	(+(	AB*
C	(+(	AB*C
*	(+(*	AB*C
D	(+(*	AB*CD
)	(+	AB*CD*
)		AB*CD*+

V. Imp

eg:- (2) A * (B + D) / E - F * (G + H / K))

Symbol	stack	postfix expression
	(	
A	(	A
*	(*	A
(	(* (	A
B	(* (	AB
+	(* (+	AB
D	(* (+	ABD
)	(*	ABD +
/	(/	ABD +
E	(/	ABD + E
-	(-	ABD + E / *
F	(-	ABD + E / * F
*	(- *	ABD + E / * F
(	(- * (	ABD + E / * F
G	(- * (	ABD + E / * F G
+	(- * (+	ABD + E / * F G
H	(- * (+	ABD + E / * F G H
/	(- * (+ /	ABD + E / * F G H
K	(- * (+ /	ABD + E / * F G H K
)	(- *	ABD + E / * F G H K / +
)		ABD + E / * F G H K / + *

bada Precedence Add chota aaya to pop.

Page No.
Date

Ans - abcd*+e++fg/+

eg:- (3) a+(b+c*d+e)+f/g)

Symbol	stack	postfix expression
	(	
a	(	a
(	((	a
b	((	ab
+	((+	ab
c	((+	abc
*	((+*	abc
d	((+*	abcd
+	(+	abcd*+
e	(+	abcd*+e
+	(+	abcd*+e
)	(	abcd*+e+
+	(+	abcd*+e+
f	(+	abcd*+e+f
/	(+/	abcd*+e+f
g	(+/	abcd*+e+fg
)		abcd*+e+fg/+

Page No.
Date

Ex :- (4) A $ B X C - D + E / F / (G + H))

Symbol	Stack	postfix expression
	(	
A	(	A
$	($	A
B	($	A B
X	(X	A B $
C	(X	A B $ C
-	(X -	A B $ C X
D	(X -	A B $ C X D
+	(- +	A B $ C X D
E	(- +	A B $ C X D E
/	(- + /	A B $ C X D E
F	(- + /	A B $ C X D E F
/	(- + / /	A B $ C X D E F
(	(- + / / (	A B $ C X D E F
G	(- + / / (	A B $ C X D E F G
+	(- + / / (+	A B $ C X D E F G
H	(- + / / (+	A B $ C X D E F G H
)	(- + / /	A B $ C X D E F G H +
)		A B $ C X D E F G H + / / -

Page No.
Date

Conversion of infix to prefix

Algorithm ⇒

step 1 ⇒ Reverse the infix expression
(a+b)*c → c*(b+a)

Step 2 ⇒ Apply infix to postfix algorithm to obtain postfix expression.

step 3 ⇒ Reverse that postfix expression to obtain prefix expression.

Ex:→ (a+b)*c

P *c

*PC

prefix → *+abc (a+b)*c → c*(b+a))

symbol	stack	postfix expression
	(	
c	(	c
*	(*	c
(	(*(	c
b	(*(	cb
+	(*(+	cb
a	(*(+	cba
)	(*	cba+
)		cba+*

→ cba+*

→ reverse

→ *+abc

Page No.
Date

Ex ⇒ (d-c) * (b-a)

x * y

* xy

* - dcy

Prefix → * - dc - ba

(d-c) * (b-a) ⟶ (a-b) * (c-d)

Symbol	stack	postfix expression
	(	
c	((	
a	((	a
-	((-	a
b	((-	ab
)	((	ab-
*	(*	ab-
(	(*(	ab-
c	(*c	ab-c
-	(*c-	ab-c
d	(*c-	ab-cd
)	(*	ab-cd-
)		ab-cd-*

⟶ ab-cd-* reverse ⟶ * - dc - ba.

Application

(2) Evaluation of postfix expression

Algorithm :-

Step 1 :- Add closing round bracket at the end of the expression.

Step 2 :- Scan the expression from left to right until ')' round bracket encountered.

Step 3 :- If an operand encountered ~~pot to~~ push it to stack.

Step 4 :- If an operator encountered then

(i) Pop top two operand from stack

~~(ii) First~~ Pop operand is denoted by OP_1, second pop operand is denoted by OP_2

(ii) Evaluate operand$_2$ ⊕ OP_1

(iii) Put that answer to stack

Step 5 :- Top of the stack will be the Final Answer.

Step 6 :- Exit.

Ex → ① 5 6 2 + * 12 4 / -)

Symbol	stack	
5	5	
6	5 6	
2	5 6 2	6+2 = 8
+	5 8 (op2 op1)	
*	40 (op2 op1)	5 x 8 = 40
12	40 12	
4	40 12 4 (op2 op1)	12/4 = 3
/	40 3 (op2 op1)	40 - 3
-	37	
)	37	

② 4 5 4 2 ^ + * 2 2 ^ 9 3 / * -)

Symbol	stack	
4	4	
5	45	
4	454	
2	4542 (op2 op1)	4^2.
^	4516 (op2 op1)	5+16
+	421	
*	4212	
2	4212 2 (op2 op1)	2^2 =
2	4214	
^	4849	
9	4849 3 (op2 op1)	9/3 = 3
3	4843 (op2 op1)	21*3 = 63
/	463	
*	40	
-	40 63	

refer to practical page

Page No.
Date

Ex 2 → 100 200 + 2 ÷ 5 × 7 + 100 200 + 2 ÷ 5 × 7 +)

Symbol	stack	
100	100	
200	100 200 (op2 op1)	200+100=300
+	300	
2	300 2	
+	302	
2	302 2 (op2 op1)	302/2 = 151
÷	151	
5	151 5 (op2 op1)	151×5=755
×	755	
7	755 7 (op2 op1)	755+7=762
+	762	
100	762 100	
200	762 100 200 (op2 op1)	100+200=300
+	762 300	
2	762 300 2 (op1)	300÷2=150
÷	762 150 (op2)	
5	762 150 5 (op2 op1)	150×5=750
×	762 750	
7	762 750 7	
+	762 757 (op2 op1)	
)		

Stack
Correct code:-

```
#include<stdio.h>
#define size 5
 void push();
 void pop();
 void peek();
 void display();
int top=-1, inp_array[size];
 void main()
 {
 int ch;
do {
pf("which operation do you want to perform:");
pf("1 push");
pf("2 pop");
pf("3. peek");
pf("4. display");
Sf("%d",&ch);
switch(ch)
 {
case 1: push();
break;
Case 2: pop();
break;
case 3: peek(); break;
case 4: display();
    break;
Case 5: pf("Exit");
   break;
default: pf("invalid choice");
} while(ch!=5);
}
```

```
void push ()
 {
 int x;
 if(top = size-1)
  {
  pf("\n overflow!! ");
 }
 else {
  pf("\n enter the element
 to be added on the stack:");
 sf("%d ",&x);
  top = top+1;
  inp-arr[top] =x;
  }
 }
 void pop()
  {
 if (top == -1)
  {
 pf("\n Underflow!! ");
 }
else {
pf ("\n Delete element: %d", inp-arr[top]);
top = top-1;
 }
 }
void peek ()
  {
 int i;
if (top= =-1)
 {
 pf ("stack underflow\n");
}
else {
 pf("%d", inp-arr[top]);
 }
 }
void display ()
int i;
if (top == -1)
 {
pf (top == -1)
 {
pf ("\n stack underflow!! ")
 }
else {
pf ("\n elements int in
 the stack : \n");
for (int i=0; i<top; i++)
pf("%d \n", inp.array[i]
 }
 }
```

✱ Queue

→ Queue :- Ordered list of similar data elements.

End for Insertion :- REAR

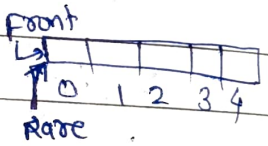
End - for Deletion :- FRONT

Queue → example :- Movie ticket, ATM line, etc.

principal :- FIFO. (First in First out)

Insertion :- enqueue ()
deletion :- dequeue ()
peek () :- displays front element of queue
display :- displays all the elements of queue.

→ Queue (Algorithm) for Insertion



Step:-1) ~~To insert~~ Initialize front = 0
rear = -1

MAXSIZE = 5

Step:-2) If rear = = maxsize - 1
pf ("Queue is Full" / Queue overflow)
else
increment rear by 1
queue [rear] = item.

rear → insert
front → delete

→ Algorithm for deletion.

step 1 :- If (front == 0 || front > rear)
print "Queue is empty" / Queue is underflow.
else
item = queue[front]

step 2 :- Increment front by 1.

Program :-

```
#include<stdio.h>
#define MAXSIZE [5]
void enqueue();
void dequeue();
Void peek();
Void display();
int queue[MAXSIZE];
int front=0, rear=-1;
void main()
{
int ch; do {
pf("1. Insertion");
pf("2. deletion");
pf("3. peek");
pf("4. display");
pf("5. exit");
sf("%d", &ch);
} while
switch(ch)
{
case 1: Insertion;
break;
```

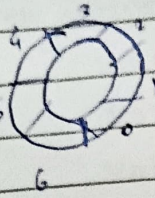
```
case 2: delesm;
  break;
case 3: peek;
    break;
case 4: display;
    break;
case 5: exit;
  break;
default pf ("invalid choice");
} while (ch != 5);
  }

Void enque ()
  {
  int item;
  if (rear == MAXSIZE - 1)
    {
    pf
  }
  else
    {
    pf ("
    sf ("%d", &item);
    rear = rear + 1;
    queue [rear] = item;
}
```

draw back → Queue :- Can't utilise empty space, memory...

★ Circular Queue :-

→ Rep" of Array.



→ ★ Draw Back of Simple Queue / Linear Queue
→ In normal queue we can insert the element until queue becomes full, but once queue becomes full we can not insert the next element even if there is space in front of the queue

So, wastage of memory occurs in simple queue to utilise this memory the new concept occurs i.e Circular Queue.

→ Circular Queue also known as Ring Buffer, Circular buffer.

→ In Circular Queue last position is connected to first position to make a queue circular.
The Main Advantage of circular queue is Utilization of memory.

Algorithm of Insertion ⇒

Step 1 :- If (Rear + 1) % N = F
(R; mod ↓; N → maxsize)
Display "Overflow"

Step 2 :- If F = -1, R = -1 set F = R = 0 → No need
else
R = (R+1) % N

Step 3 :- set Q[R] = item

Step 4 :- exit.

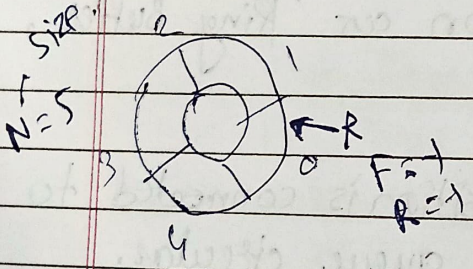
Deletion Algorithm.

Step 1: if F = -1
display "Underflow"

Step 2:- set item = Q[F]

Step 3:- if F = R set F = R-1
else
F = (F+1) % N

Step 4:- exit.



App^n Circular Queue
→ Traffic Management:
→ CPU Management - OS.

Time Complexity →

Simple Queue →
~~Best Case~~
enqueue → O(1)
dequeue → O(1)
peek → O(1)
display → O(n)

Circular Queue →
enqueue → O(1)
dequeue → O(1)
peek → O(1)
display → O(n)

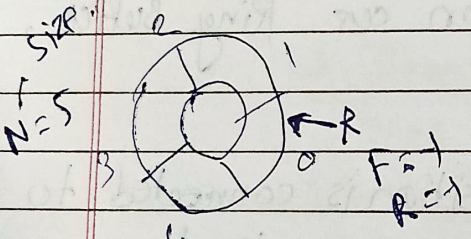
Deletion Algorithm.

Step 1: if F = -1
display "Underflow"

Step 2:- set item = Q[F]

Step 3:- if F = R set F = R-1
else
F = (F+1) % N

Step 4:- exit.



App^n Circular Queue
→ Traffic Management:
→ CPU Management - OS.

Time Complexity →

Simple Queue →
~~Best case~~
enqueue → O(1)
dequeue → O(1)
peek → O(1)
display → O(n)

Circular Queue →
enqueue → O(1)
dequeue → O(1)
peek → O(1)
display → O(n)

Page No.
Date

Program ⇒

```
#include<stdio.h>
#define MAXSIZE 5
    void enqueue();
    void dequeue();
    void peek();
    void display();
int cqueue[MAXSIZE]; int N, int front=0, rear=-1;
    void main()
    {
    int ch;
    do {
    pf("1. Insertion");
    pf("2. deletion");
    pf("3 peek");
    pf("4. display");
    pf("5. exit");
```

```
void enqueue() {
    int item;
    if (front == (Rear+1) % N)
    {
    pf("overflow");
    }
    else {
      pf("Enter item you want to insert");
    sf("%d", &item);
    if Rear = (Rear+1) % N;
```

```
[Rear] = item;
}

void dequeue()
{
if (f == -1)
{
pf("Underflow");
}
else {
item = cq[f];
pf("%d", item);
f = (f+1) % n); }
```

✱ Priority Queue (Theory Not to implement)

10	20	30	40
4	1	2	3

Assending - 1 - highest priority
4 - lowest

Implement ① Using Array
② Using Multi queue
③ Using linked list
④ Using Heap

→ Each element in the queue has associated priority & for dequeue highest priority element get deleted

→ Properties of Priority Queue :- FIFO (First in first out)

① Based on first in first out principle
② Every item has priority associated with it.
③ An element with high priority is dequeue before an element with lower priority
④ If two element has same property according to their order in their queue.

Two types ⇒ assending priority & descending priority

Assending ⇒

① Insertion - It can be normal or special
O(1) O(n)

② deletion - It can be normal or special
O(1) O(n)

Normal Insertion →
Special deletion
Special Insertion →
Normal deletion

Page No.
Date

* Implementation of priority

(1) By Using Array

ex :- | 5 | 2 | 9 | 6 | 8 | 7 | 4 | 10 |
2 1 2 3 1 2 1 2

Assending priority	Desending priority
2 8 4 5 9 7 6 — O(n)	6 5 9 7 10 2 8 4
1 1 1 2 2 2 3	3 2 2 2 2 1 1 1
deletion — O(1)	

(2) Multiple Queues

1 | 2 | 8 | 4 |

2 | 5 | 9 | 7 | 10 |

3 | 6 |

(3) Linked list

→ Every node is get divided into three part
(1) Information
(2) Priority.
(3) Address to the next node

start
→ | 2 | 1 | → | 4 | 2 | → | 3 | 3 | → | 29 | 4 | X |

Insert → | 5 | 2 |

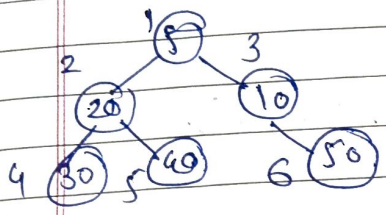
→ | 2 | 1 | → | 4 | 2 | → | 5 | 2 | → | 3 | 3 | → | 29 | 4 | X |

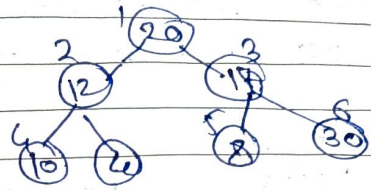
* deletion occurs from the front

(c) By using Heap :-

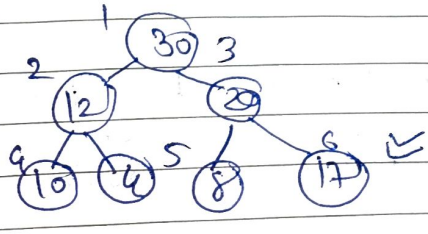
two types → Max. Heap
→ parent node is always greater than its child node

↓
Min. Heap
→ parent node is always less than its child node.





→ Greater
↓
convert



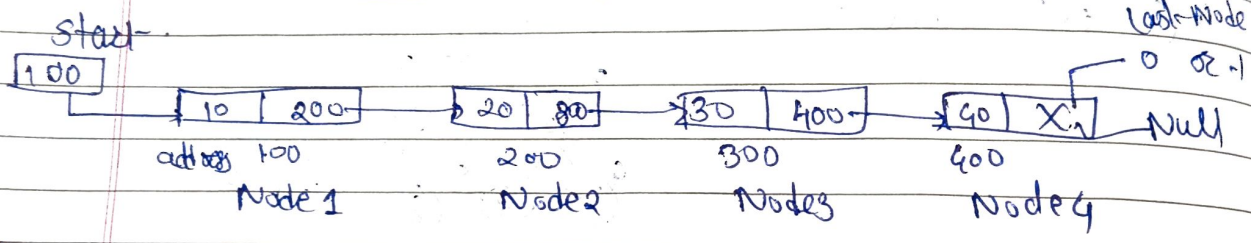
✱ Linked List

→ Also known as one way list
→ Is linear collection of data element called Node where, linear order given by pointer
→ Each Node is divided into two parts
  1) INFO - Information / data element
  2) LINK - Address of next node

node

INFO	LINK

Rep^n of linear linked list :-



✱ Advantages of Linked list

1) Dynamic Size :-
2) Ease of insertion or deletion

✱ Disadvantages

1) Random acess is not allowed
2) Extra memory used at every node

Array	Linked list
1) Static in nature	1) Dynamic in Nature
2) Insertion & delesion is not easy in array	2) Insertion & deletion is easy in linked list
3) Both the linear & binary seach techinque is possible.	3) Binary search not possible only linear seach is possible
4) We can access random element in array	4) Random access are not possible in linked list
5) it stores actual information only	5) It will take extra memory (data+pointer).
6) Extra memory is not used	6) Extra memory is used

⟹ <std lib.h>
1. malloc ()
2. calloc ()
3. realloc ()
4. free ()

* Dynamic memory allocation :-

It refers to allocating a memory to the variable at run time. So that the size of the variable can be change according to our need.
It uses four function from the header file
< stdlib.h >

(1) malloc ()
(2) calloc ()
(3) realloc ()
(4) Free ()

① memory allocation
② It reserved a block of memory
③ It returns a pointer of type void

Syntax :- ptr = (datatype *) malloc (byte-size);

ex :- 2 byte
ptr = (int *) malloc (n * size of (int)); pointer;

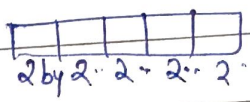
④ It does not initialize the block of memory (by garbage computer)
⑤ If allocation fail it writtens null void point.

(By default memory will take garbage value)

(2) Calloc () :-
① It stand for continuous memory allocation
② It reserved no. of block in memory
③ It returns a pointer of type void.

Syntax :- ptr = (datatype *) calloc (n, element size)

ex :- int -2
ptr = (int *) calloc (5, size of (int));



④ If allocation fail it written null point
⑤ It does not initialise block of memory by zero
⑥ It is used for complex data structure when we need more memory allocation.

(3) realloc()

① It stands for reallocation

② By using Malloc & Calloc we have allocated lots of memory, if we think it is ~~insificient~~ insufficient then we can call the func^n realloc().

③ It is used for changing the size of previously allocated memory

④ Syntax:-

realloc(ptr, size);

realloc(ptr, 5 * sizeof(int));

(4) Free()

→ It is use for realising the space allocated by dynamic memory allocation

Syntax:-

Free(ptr);