Abstract Data Types (ADT)

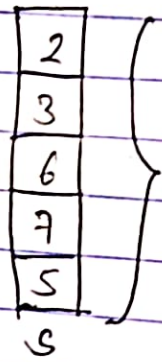
An abstract data types consists of two parts :-

i) a value definition and

ii) an operator definitions associated with value definition

eg:- stack, queue, linked-List.

Stack



model (collection of values)
operation (push, pop)

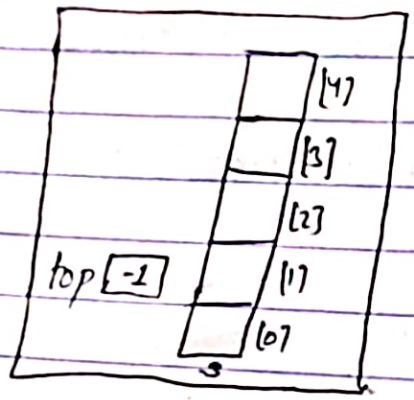
eg push (s, 5)
push (s, 7)
push (s, 6)
push (s, 3)
push (s, 2)
push (s, 14) // stack overflow

eg. pop (s) // return 2
pop (s) // return 3
pop (s) // return 6
pop (s) // return 7
pop (s) // return 5
pop (s) // stack underflow

Scanned by CamScanner

An ADT is a mathematical model together with operation associated with it.

top = -1



eg:- push (s, 5) top++;
s[top] = 5
push (s, 7) top++;
s[top] = 7

```
#define STACKSIZE 5
struct stack
    { int s[STACKSIZE];
      int top;
    };
void main ()
{   struct stack st~~First~~;
    st~~First~~. top = -1;
    push (st, 5);
    push (st, 7);

}
```

```
/*
void push (struct stack
          st, int value
{
    st.top ++;
    if (st.top >= STACKSIZE
    {
    printf (" Stack Overflow
     exit (1);
    }
    else
    { st. s[top] = value
```

Scanned by CamScanner

```
~~void~~ int pop (struct stack st)
{
    if (st.top == -1)
    { printf (" Stack Underflow");
      exit (1);
    }
    return (st.s [st.top--]);
}

void push (struct stack st, int value)
{
    if (st.top == (STACKSIZE -1))
    { printf (" Stack Overflow ");
      exit (1);
    }
    st.s [++st.top] = value;
}
```

Scanned by CamScanner

```
# define sz 5
struct stack
{
    int top;
    int s[sz];
};
void main ()
{
    struct stack st;
    st. top = -1;
    push (&st, 5);
    push (&st, 7);
    push (&st, 4);
int Is Empty (&st);
    pitem = pop (&st);
    display (st);
    printf ("\n Popped item = %d", pitem);
}
void push (struct stack *st, int value)
{ if (st -> top == sz-1)
    { printf ("Stack overflow");
      exit (1);
    }
    st -> s [++st -> top] = value;
}
int pop (struct stack *st)
{ if (st -> top == -1)
    { printf ("Stack overflow");
      exit (1);
    }
    return (st -> s[st -> top --]);
}
```

Scanned by CamScanner

David Malan @nhuzaa

```
int IsEmpty (struct stack st)
  { if (st.top == -1)
      return 1;
      else
      return 0;
    }

int topStack (struct stack st)
  { return (st.[   ]); }

void displayStack (struct stack *st)
  { int i;
    for (i = st -> top ; i >= 0 ; i--)
      { printf ("\n %d", st -> s [i] ); }
  }
```

Scanned by CamScanner

Application of Stack

i) Balanced Parenthesis

* Algorithm to check balanced parenthesis

```
valid = true
S = empty stack
while (we do not read the entire string)
    symbol = read the next symbol from input string
if (symbol == '(')
    push (S, symbol
```

Scanned by CamScanner

Infix, Postfix and Prefix Expressions (Application of Stack)

→ Evaluating a postfix expressions
→ Converting an expression from infix to postfix

let us consider a postfix expression

eg: 623 + - 382 / + * 2$3 +

Precedence

exponent ($) R → L
Multiplication & division (*, /) L → R
Addition & Subtraction (+, -) L → R

Associativity

Exponents
left to right
right to left

eg 5+3*6
= 5 + 18
= 23

eg: 5*3/3
= 15/3
= 5

eg 2^{53} or 2$5$3
= $(2)^{53}$
= 2^{125}

operands
2 + 5
↑
operator

Scanned by CamScanner

Symbol	opnd 1	opnd 2	value	opndstk
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
−	6	5	1	1
3				1, 3
8				1, 3, 8
2				1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2				7, 2
$	7	2	49	49
3				49, 3
+	49	3	52	52

Scanned by CamScanner

532 * 2 / + 32 $ 2 * +

symbol	opnd 1	opnd 2	value	opndstk
5				5
3				5, 3
2				5, 3, 2
*	3	2	6	5, 6
2				5, 6, 2
/	6	2	3	5, 3
+	5	3	8	8
3				8, 3
2				8, 3, 2
$	3	2	9	8, 9
2				8, 9, 2
*	9	2	18	8, 18
+	8	18	26	26

---

532 * 2 / + 32 $ 2 * +

=> 562 / + 32 $ 2 * +

=> 53 + 32 $ 2 * +

=> 832 $ 2 * +

=> ~~834 * +~~ 892 * +

=> ~~8 (12) +~~ 8 (18) +

=> 26

Scanned by CamScanner

5 9 2 $ 3 / 4 * + 6 +

Symbol	opnd 1	opnd 2	value	opndstk
5				5
9				5, 9
2				5, 9, 2
$	9	2	81	5, 81
3				5, 81, 3
/	81	3		5, 27
4				5, 27, 4
*	27	4	108	5, 108
+	5	108	113	113
6				113, 6
+	113	6		119

5 9 2 $ 3 / 4 * + 6 +
=> 5 (81) 3 / 4 * + 6 +
=> 5 (27) 4 * + 6 +
=> 5 (108) + 6 +
=> (113) 6 +
=> 119

Scanned by CamScanner

4 2 3 2 $ $ 4 / +

symbol	opnd1	opnd2	value	opndstk
4				4
2				4, 2
3				4, 2, 3
2		2		4, 2, 3, 2
$	3	9	9	4, 2, 9
$	2	4	512	4, 512
4	512	128		4, 512, 4
/	4			4, 128
+				4, 132

4 2 3 2 $ $ 4 / +

4 2 9 $ 4 / +

⇒ 4 (512) 4 / +

⇒ 4 (128) +

⇒ 132

Scanned by CamScanner

62 /84 /+93/ −

symbol	opnd1	opnd 2	value	opndstk
6				6
2				6,2
/	6	2	3	3
8				3,8
4				3,8,4
/	8	4	2	3,2
+	3	2	5	5
9				5,9
3				5,9,3
/	9	3	3	5,3
−	5	3	2	2

Scanned by CamScanner

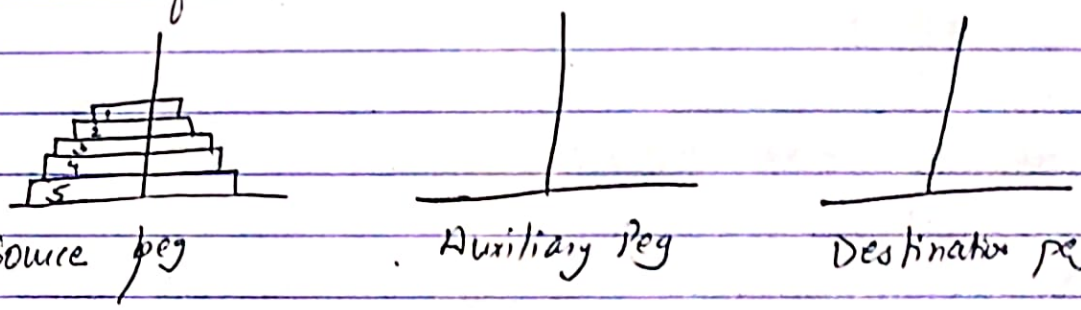
Recursion

→ Whenever a function calls itself for smaller instance is called recursion

→ recursive uses stack

→ approaches for problem solving
- * iterative approach
- * recursive approach

Tower of Hanoi



(1, 2, 3, 4)	S → A	[D]	⇒	(1,2,3)	S → A D	[A]
5	S → D	~~[A]~~		4	S → A	
(1,2,3,4)	A → D	[S]		(1,2,3)	D → A	[D]
				⇓		
				(1,2)		

Scanned by CamScanner

Factorial

$n! = n * (n-1) * (n-2) * \ldots * 3 * 2 * 1$ for $n \geq 1$

$n! = 1$ for $n = 0$

Iterative approach to calculate factorial

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n \geq 1 \end{cases}$$

Program

```
#.....
int fact (int n);
int main ()
{
    int x;
    cout << " Enter the value of x : ";
    cin >> x;
    cout << " The factorial of "<<x<<" is " << fact (x);
    return 0;
}
```

```
int fact (int n)
{
    int r=1;
    if (n==0)
    return (1);
    while (n>=1)
    {
        r = r*n;
        n--;
    }
    return r;
}
```

```
int fact (int n)
{
    if (n==0)
    return 1;
    else
    return n*fact(n-1);
}
```

Scanned by CamScanner

Product

result = a * b
= 5 * 6

$$a*b \begin{cases} a & \text{if } b = 1 \\ a + a*(b-1) & \text{otherwise } (b \geq 2) \end{cases}$$

```
int product (int a, int b)
{   if (b == 1)
        return (a);
    return (a + product (a, b-1))
}
```

Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21

```
int fibonacci (int n)    /* n is non-negative number */
{   if (n <= 1)
        return n;
    return (fibonacci (n-1) + fibonacci (n-2));
}
```

<u>For n = 5</u>

5 <= 1 . => return fibonacci(4) + fibonacci (3)

4 > 1 3 > 1

ret fib (3) + fib (2) ----> fib (2) + fib (1)

3 2 > 1

fib (1) + fib (0)

return 1 return 0

Binary Search

a	3	7	9	17	20	27	43	49	54	62
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	↑ low				↑ mid					↑ high

x (search element) = 9

mid = (low + high) / 2 = (0+9)/2 = 4

```
void main ()
{ int a[] = {3, 7, 9, 17, 20, 27, 43, 49, 54, 62};
  int low = 0, high = 9, position;
  int x;
  printf("Enter the value for X:");
  scanf("%d", &X);
  position = BinarySearch (a,x, low, high);
  printf(" The position for element %d is %d ",x,position);
}

int BinarySearch (int a[], int x, int low, int high)
{ int mid;
  if ~~(l>h)~~ (low > high)
      return (-1);
  mid = (low + high)/2
  if (X == a[mid])
      return (mid);
  else if (x < a[mid])
      return (BinarySearch (a, X, low, mid-1));
  else
      return BinarySearch (a, X, mid+1, high);
}
```

Scanned by CamScanner

For X = 54

```
int B_S (a, 54, 0, 9)
{   0<9
    mid = 4
    X > a[4]
    return B_S (a, 54, 4+1, 9)
```

```
int B_S (a, 54, 5, 9)
{   3<9
    mid = 7
    X > a[7]
    return B_S (a, 54, 7+1, 9)
}
```

```
int B_S (a, 54, 8, 9)
{   8<9
    mid = 8
    X == a[8]
}   return 8
```

Scanned by CamScanner

Queue

Stack – LIFO FILO
Queue – FIFO LILO

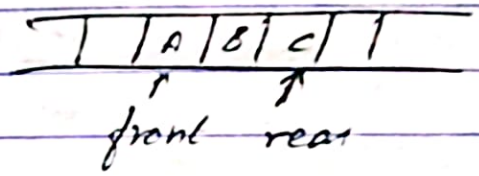
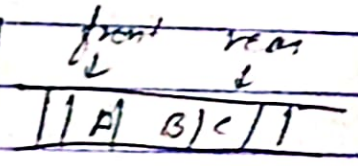


fig:- Instance of a queue

Operations associated with queue

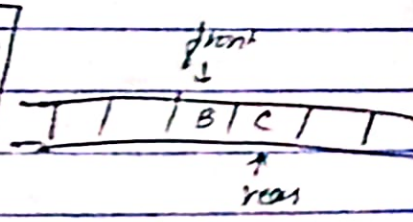
1.) insert (q, item) eg

```
insert (q, A)
insert (q, B)
insert (q, C)
```



2.) x = remove (q)

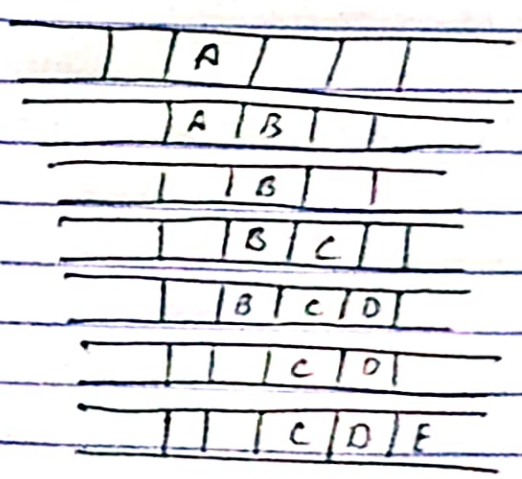
```
x = remove (q)
```

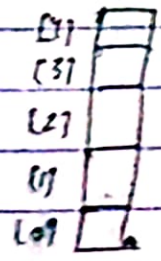


3) empty (q) — true (1), false (0)

---

insert (q, A)
insert (q, B)
x = remove (q);
insert (q, C);
insert (q, D);
x = remove (q);
insert (q, E);





front = 0
rear = -1

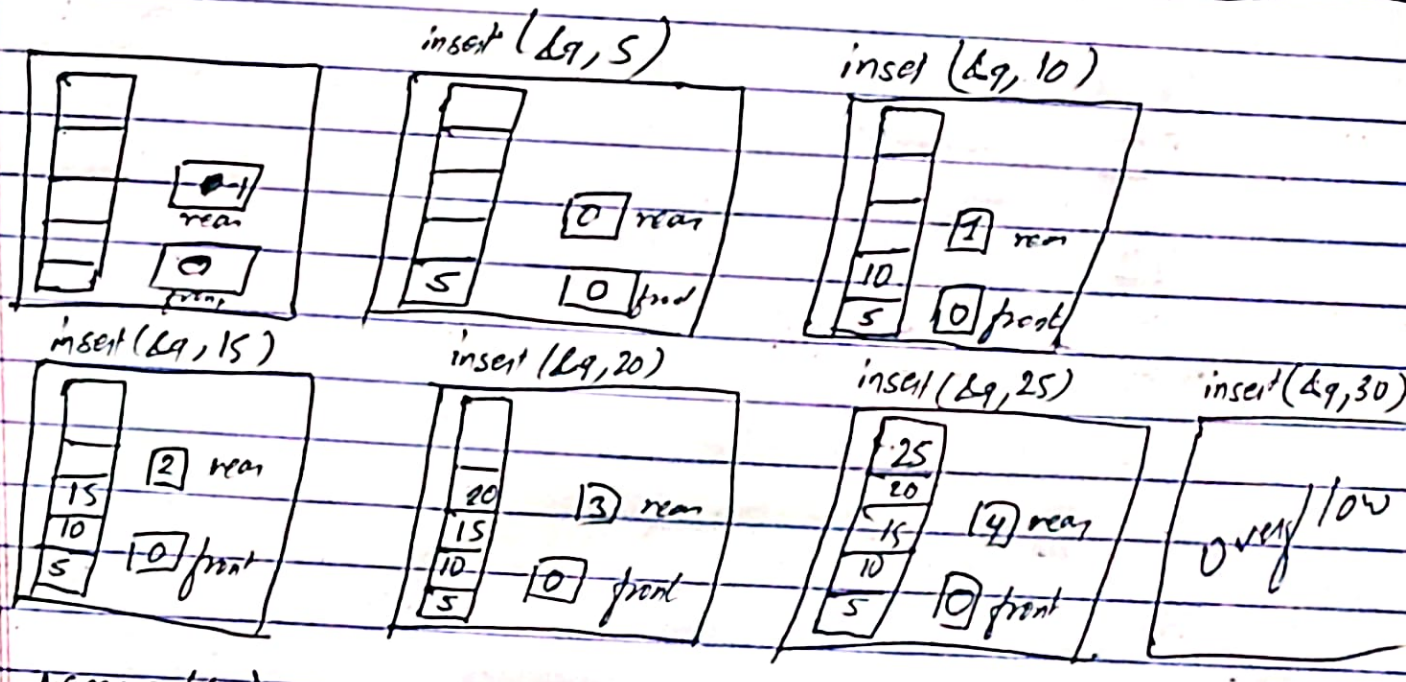
Scanned by CamScanner

C Program for queue

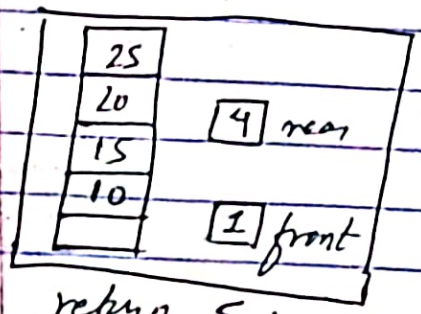
```
#define MAXQUEUE 5
struct Queue
{ int items [MAXQUEUE];
  int front, rear;
};

void inset (struct queue *pq, int t)
{
   if (pq -> rear == MAXQUEUE -1)
   { printf (" QUEUE overflow "); }
     exit (1); }
  pq -> items [++pq -> rear] = t;
}


int remove (struct Queue *pq)
{
   if (pq -> rear < pq -> front)
   { printf (" Queue Underflow");
     exit (1);
   }
  return (pq -> items [pq -> front++]);
}
```

Scanned by CamScanner

```
void main ()
{
    struct Queue q ; int x ;
    q.front = 0 ;
    q.rear = -1 ;
    insert (&q, ) ;
    :
x = remove (&q);

}
```



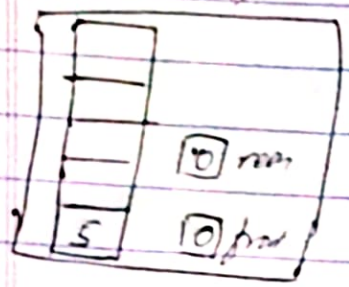
remove (&q)



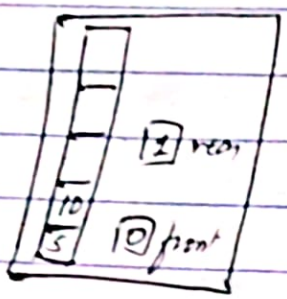
return 5;

Scanned by CamScanner

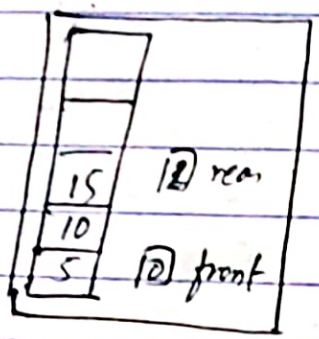
insert (Lq, 5) ·



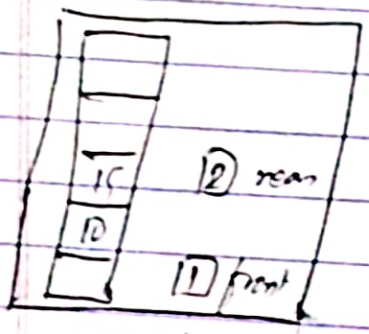
insert (Lq, 10);



insert (Lq, 15)

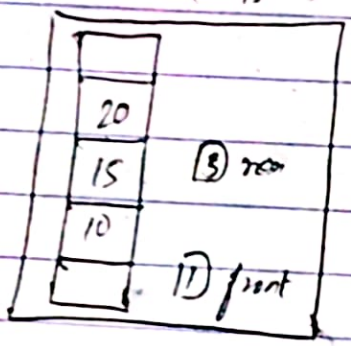


x = remove (Lq)

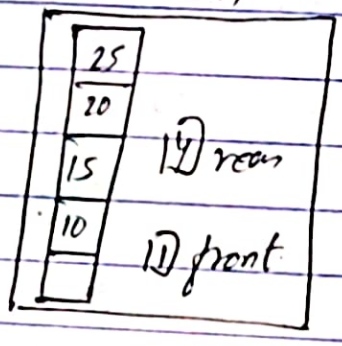


x = 5

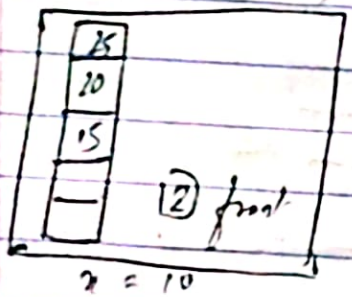
insert (Lq, 20)



insert (Lq, 25)



x = remove (Lq)



x = 10

Scanned by CamScanner

Queue and its sequential representation (array representation)

empty (q)
insert (q, x)
x = remove (q)

To overcome with St Queue overflow & Underflow

* Solution 1

remove operation
→ shift the entire queue
(shift is complex if size is large)

* Solution 2

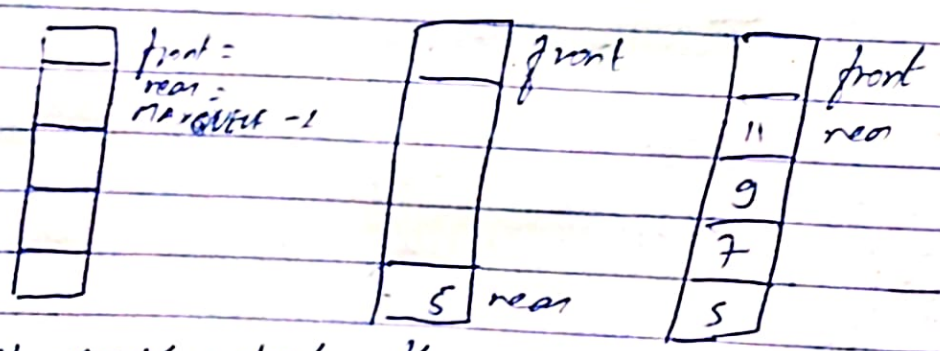
→ Queue as circle
• setup as rear = front = Maxsize - 1

Program (Cyclic)

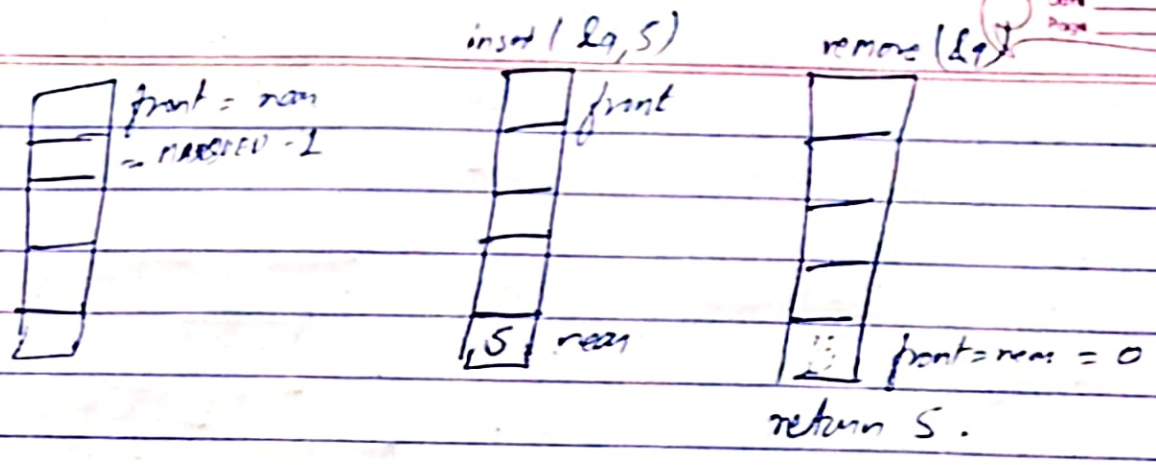
```
#define MAXQUEUE 5
struct queue
{
  int items [MAXQUEUE]
  int front, rear;
}
void main ()
{
  struct queue q;
  q.front = q.rear = MAXQUEUE - 1;
  insert (&q, 5);
  insert (&q, 7);
  . . . . . .

}
```

Scanned by CamScanner

```
void insert (struct queue *pq, int x)
{
    if (pq->rear == MAXQUEUE -1)
            pq->rear = 0;
    else
            (pq->rear)++;

    if (pq->rear == pq->front)
    {
        printf("Queue Overflow");
        exit(1);
    }
    pq->items [pq->rear] = x;
}

int remove (struct queue *pq)
{
    if (pq->front == MAXQUEUE -1)
            pq->front = 0;
    else
            (pq->front)++;
    return (pq->items [pq->front]);
}
```



* If rear+1 = front, then such case is known as queue overflow.

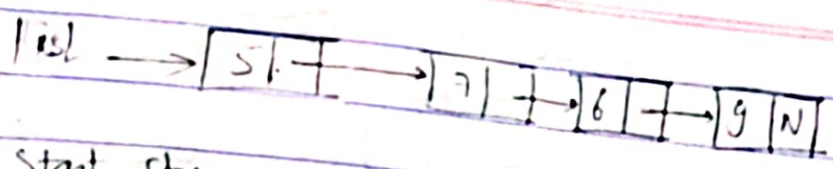


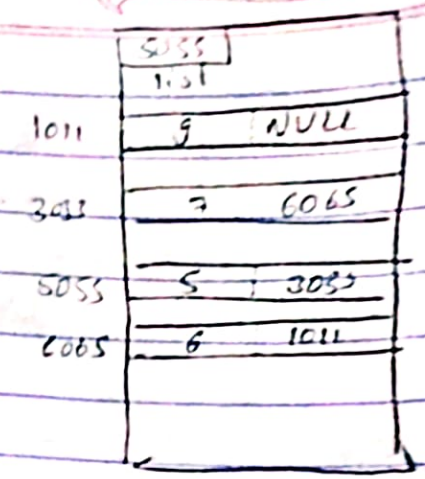
Priority Queue

a) Ascending Priority Queue (Pr 182)

Scanned by CamScanner

List

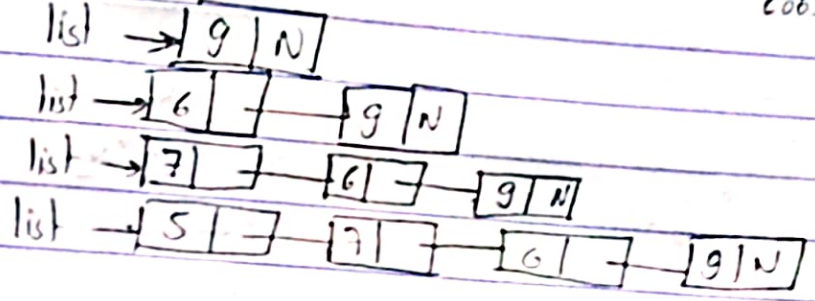




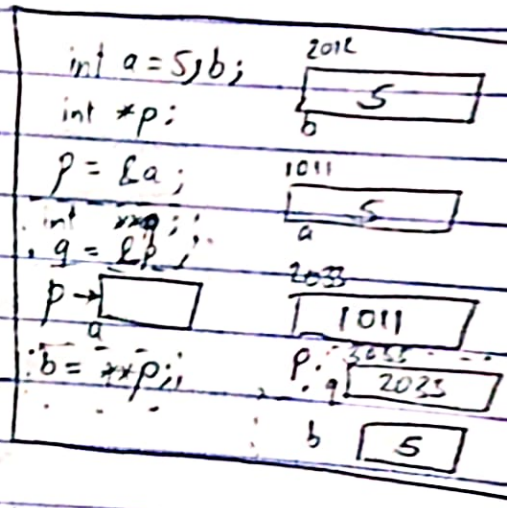
Start step

list → N

add at beg



Program

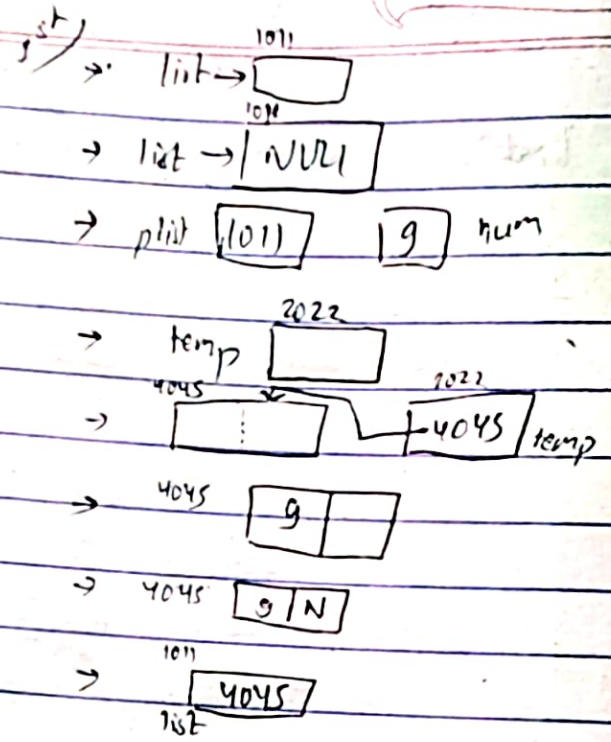



```
struct node
  {
      int data;
      struct node *next;
  };
void addatbeg (struct node **plist, int num)
  {
      struct node *temp;
      temp = (struct node *) malloc (sizeof (struct node));
temp-> data = num; temp -> next = *plist;
      *plist = temp;
  }
  void display (struct node *r)
  { printf (" list -> ")
    while (r != NULL)
    { printf ("%d -> ", r -> data); r = r -> next; }
    printf (" NULL");
  }
```

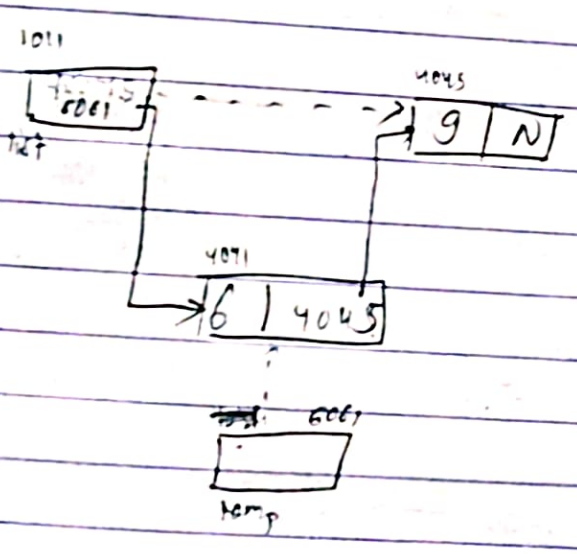
list → 7 → 3 → 5 → NULL

```
void main ()
{ struct node *list;
  list = NULL;
  add_at_beg (&list, 9);
  addatbeg (&list, 6);
  add atbeg (&list, 7);
  addatbeg (&list, 5);
  display (list);
  addatend (&list, 9);
  addatpos (&list, 2, 13);
}
```

1st)



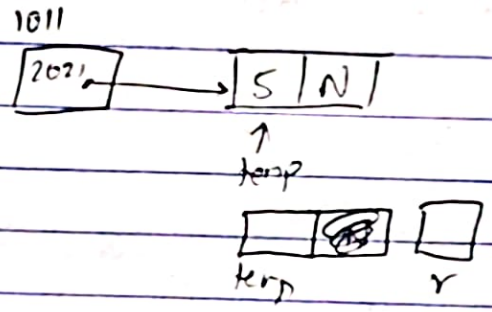
2nd)



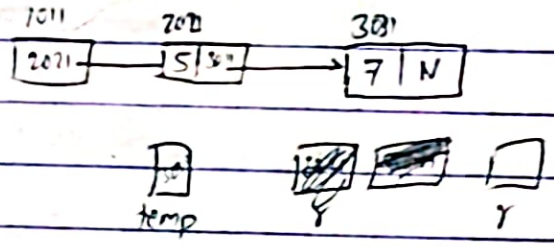

```
void addatend (struct node **plist, int num)
{
    struct node *temp, *r;
    temp = (struct node *) malloc (sizeof(struct node));
    temp -> data = num;
    temp -> next = NULL;
    r = *plist;
    if (r == NULL)
    {   *plist = temp; }
```

```
else
{   while (r -> next != NULL)
            r -> r -> next ;
    r -> next = temp ;
}
}
```

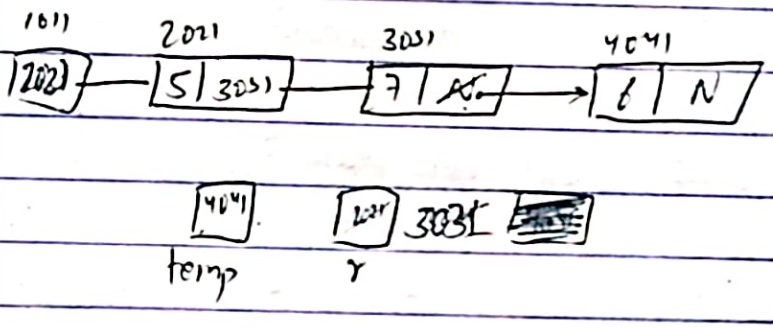
i.) addatend (&list, 5)



ii) addatend (&list, 7)

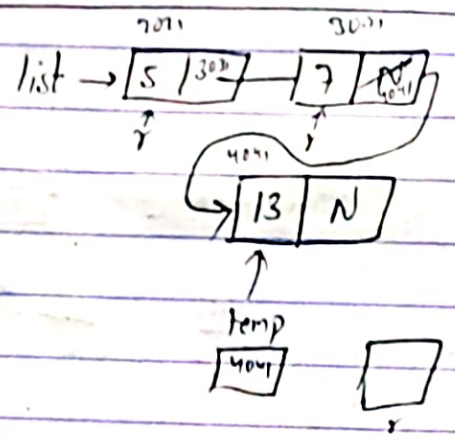


iii) addatend (&list, 6)




```
void addatpos (struct node **plist, int pos, int num)
{   struct node *temp, *r ;
    temp = (struct node *) malloc (sizeof (struct node)) ;
    temp -> data = num ;
    r = *plist ;
    ~~while~~ for (i=1 ; i<pos ; i++)
        {
            if (r == NULL)
            {   printf ("Insufficient numbers of node") ;
                return ;
            }
```

```
        r = r -> next;
    }
    temp -> next = r -> next;
    r -> next = temp;
}
```



Dynamic Memory Allocation

```
void main()
{
  int a;
  a = 5;
  printf("Value of a = %d", a);
}
```

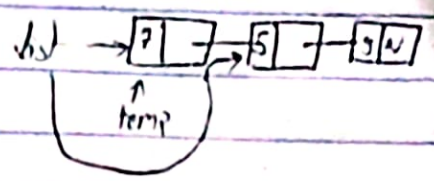
```
void main()
{ int *p;
  p = (int *)malloc(sizeof(int));
  *p = 5;
printf("Value at address %u is %d",
}
```

Scanned by CamScanner

Deletion at link list

Delete from beginning

```
void delfrombeg (struct node **plist)
{
    struct node *temp;
    temp = *plist;
    if (temp == NULL)
    {   printf ("List is empty");
        return;
    }
    *plist = temp -> next;
    free (temp)
}
```



Delete from end

```
void delfromend (struct node **plist)
{
    struct node *temp, *prev;
    temp = *plist
    if (temp == NULL)
    {   printf ("List is empty");
        return;
    }
    else if (temp -> next == NULL)
    {   *plist == NULL;
        free (temp);
    }
    else
    {   while (temp -> next != NULL)
        {   prev = temp;
            temp = temp -> next;
        }
```

Scanned by CamScanner

```
            prev->next = NULL;
            free (temp);
        }
    }
```

Delete a given node

```
void delanode (struct node **plist, int num)
{
    struct node *prev, *temp;
    temp = *plist;
    if (temp == NULL)
    {
        printf ("List is empty");
        return;
    }
    while (temp != NULL)
    {
        if (temp->data != num)
        {
            prev = temp;
            temp = temp->next;
        }
        else
        {
            if (temp == *plist)
            {
                *plist = temp->next;
                free (temp);
            }
            else
            {
                prev->next = temp->next;
                free (temp);
            }
        }
    }
```

Scanned by CamScanner

linklist implementation of Stack

top → NULL

push (&top, 5); top → [5 | N]

push (&top, 7); top → [7 |] → [5 | N]

push (&top, 9); top → [9 |] → [7 |] → [5 | N]

```
struct node
{   int data;
    struct node *next;
};
void main ()
{ struct node *top = NULL ; int x;
    push (&top, 5);
    push (&top, 7);
    push (&top, 9);
    display (top);
    x = pop (&top);
    printf ("\nPopped item = %d", x);
}
void push (struct node **~~top~~top, int num)
{
    struct node *temp;
    temp = (struct node *) malloc (sizeof (struct node));
    temp -> data = num;
    temp -> next = *p~~list~~top;
    *p~~top~~top = temp;
}
```

Scanned by CamScanner

```
int pop (struct node **ptop)
{
    struct node *temp;
    int x;
    if (*ptop == NULL)
    {
        printf ("\n Stack Underflow");
        return (-1);
    }
    temp = *ptop;
    *ptop = (*ptop) -> next;
    x = temp -> data;
    free (temp);
    return (x);
}

void display (struct node *st)
{
    struct node *r = st;
    printf ("\n STACK TOP ");
    while (r != NULL)
    {   printf ("\n %d", r -> data);
        r = r -> next;
    }
    printf ("\n END \n\n");
}
```

Scanned by CamScanner

Implementation of Queue

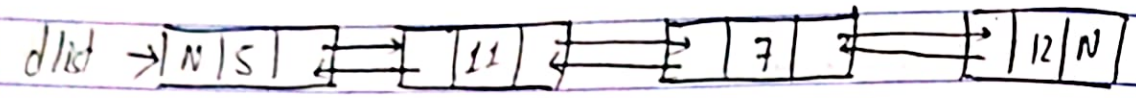
```
#include <stdio.h>
#include <malloc.h>

Struct node
{    int data;
     struct node *next;
};

void insert (struct node **qfront, struct node **qrear, int num)
{
  struct node *temp;
  temp = (struct node *) malloc (sizeof(struct node));
  temp -> data = num;
  temp -> next = NULL;
  if (*qrear == NULL)
  {    *qfront = *qrear = temp; }
  else
  {  (*qrear) -> next = temp;
       *qrear = temp;
  }
}
void display (struct node *qfront, struct node *qrear)
{
    Struct node *r = qfront;
    printf ("\n\n QFront -->");
    while (r != NULL)
    {  printf ("%d -->", r->data);
       r = r -> next;
    }  printf (" END\n\n");
}
```

Scanned by CamScanner

```
void qremove ( struct node *qfront , struct node *qrear)
 {
   struct node *temp;
   int x;
   if (*qfront == NULL)
   { printf ("\n\t\t Queue Underflow \n");
     return (-1);
   }
   temp = *qfront;
   *qfront = (*qfront) -> next;
    x = temp -> data;
    free (temp);
    return x;
 }

 void main ()
  {
    struct node *qfront , *qrear;
      qfront = qrear = NULL;
      insert (&qfront , &qrear , 5);
      insert (&qfront, &qrear , 7);
      insert (&qfront , &qrear, 9);
      insert (&qfront, &qrear, 4);
      display (qfront, qrear);
      int x = qremove (&qfront ,&qrear);
      printf ("\n The removed data is %d ", x);
      display ( qfront , qrear);
  }
```

Scanned by CamScanner

Doubly Linked list




```
struct dnode
  { struct dnode *prev;
    int data;
    struct dnode *next;
  }

void addatbeg (struct dnode **pdlist, int num)
  { struct dnode *temp;
    temp = (struct dnode *) malloc (sizeof(struct dnode));
    temp -> prev = NULL;
    temp -> ~~prev~~ data = ~~NULL~~ num;
    temp -> next = (*plist);
    if ( (*pdlist) ~~-> prev = temp~~ != NULL)
        (*pdlist) -> prev = temp;
    *pdlist = temp;
  }

void display (struct dnode *dlist)
  {
    while (dlist != NULL)
      { printf("\t %d", dlist -> data);
        dlist = dlist -> next;
      }
  }
```

Scanned by CamScanner

```
void addatend (struct dnode **pdlist , int num)
{ struct dnode *temp , *r;
  temp = (struct dnode *) malloc (sizeof(structdnode));
  if ( *pdlist == NULL)
  {   temp -> prev = NULL;
      temp -> data = num;
      temp -> next  NULL;
      * pdlist = temp;
  }
  else
  {
      r = *pdlist;
      while  (r -> next ! = NULL)
      {   r = r -> next;
      }
      r -> next = temp;
      temp -> prev = r;
      temp -> data = num;
      temp -> next = NULL;
  }
Void main ()
{
  Struct dnode *dlist;
   dlist = NULL;
  addatbeg (&dlist , 12);
  addatbeg (&dlist , 7);
  addatbeg (&dlist, 11);
  add atend (&dlist , 13);
  display (dlist);
}
```

Scanned by CamScanner

Sorting

1) Bubble Sort
2) Selection Sort
3) Insertion Sort

Internal	→ Primary memory
External	→ Secondary memory

Selection Sort

Step 1 | 5 | 7 | 9 | 2 | 4 |

c=1	5	7	9	2	4
c=2	5	7	9	2	4
c=3	2	7	9	5	4
c=4	2	7	9	5	4

Step 2 | 2 | 7 | 9 | 5 | 4 |

c=2	2	7	9	5	4
c=3	2	5	9	7	4
c=4	2	4	9	7	5

Step 3 | 2 | 4 | 9 | 7 | 5 |

c=3	2	4	7	9	5
c=4	2	4	5	9	7

Step 4 | 2 | 4 | 5 | 9 | 7 |

c=4	2	4	5	7	9

```
void main ()
{
    int a[5] = {5,7,9,2,4}, n=5;
    printf ("\n Before Sort : \n");
    display (a, n);
    printf ("\n After sort : \n");
    selection (a, n);
    display (a, n);
```

Scanned by CamScanner

```
void selection (int a[], int n)
{
    int s, c, temp; /* s=step  c= comparison */
    for (s=1 ; s<=n-1 ; s++)
    {
        for (c=s ; c<=n-1 ; c++)
        {
            if (a[s-1] > a[c])
            {
                temp= a[s-1];
                a[s-1] = a[c];
                a[c] = temp;
            }
        }
    }
}

void display (int a[], int n)
{
    int i;
    for (i=0 ; i<n ; i++)
    {
        printf("%4d ", a[i]);
    }
}
```

Scanned by CamScanner

Insertion Sort

a[] = | 9 | 2 | 7 | 3 | 4 |

Step 1

| 9 | 2 | 7 | 3 | 4 |

| 9 | | 7 | 3 | 4 | temp = 2

| 2 | 9 | 7 | 3 | 4 |

Step 2

| 2 | 9 | | 3 | 4 | temp = 7

| 2 | | 9 | 3 | 4 |

| 2 | 7 | 9 | 3 | 4 |

Step 3

| 2 | 7 | 9 | | 4 | temp = 3

| 2 | 7 | | 9 | 4 |

| 2 | | 7 | 9 | 4 |

| 2 | 3 | 7 | 9 | 4 |

Step 4

| 2 | 3 | 7 | 9 | | temp = 4

| 2 | 3 | 7 | | 9 |

| 2 | 3 | | 7 | 9 |

| 2 | 3 | 4 | 7 | 9 |

Scanned by CamScanner

```
void insertion (int a[], int n)
{
    for (s=1; s<=n-1; s++)
    {
        for (j=s-1; j>=0 && a[j]>temp; j--)
        {
            a[j+1] = a[j];
        }
        a[j+1] = temp
    }
}
```

s=1

temp = a[1] = 2

j=0 && a[0] > temp

a[j+1] = a[j]

a[0] = temp = 2

s=2

temp = a[2] = 7

j=1 j>=0 && a[1] > temp

a[2] = a[1]

= 9

j=0 j>=0 && a[1] = 7

Scanned by CamScanner

Quick Sort

Partition

| 5 | 7 | 3 | 2 | 8 | 6 |

↓ partition

| 3 | 2 | 5 | 7 | 8 | 6 |

#- - -

```
void main ()
{
int x [] = { 10, 11, 13, 12, 18, 19, 15, 14 };
~~Quick So~~
printf ("Before Sort");
  Display (x);
printf ("After Sort");
  QuickSort (x, 0, 7);
  getch ();
}

void QuickSort (int x[], int lb, int ub)
{
    int p;
    p = partition (x, lb, ub);
    QuickSort (x, lb, p-1);
    QuickSort (x, p+1, ub);
}
```

Scanned by CamScanner

```
int partition (int x[], int lb, int ub)
{
    int a, temp, down, up;
    a = x[lb];
    down = lb; up = ub;
    while (down < up)
    {
        while (x[down] <= a && down < up)
            down++;
        while x[up] > a
            up--;
        if (down < up)
        {
            temp = x[down];
            x[down] = x[up];
            x[up] = temp;
        }
    }
    x[lb] = x[up];
    x[up] = a;
    return (up);
}
```

Scanned by CamScanner

d u

19 13 21 27 18 14 16 42 15 37

d u

19 13 21 27 18 14 16 42 15 37

d d u

19 13 21 27 18 14 16 42 15 37

d u

19 13 15 27 18 14 16 42 21 37 (Exchange)

d u

19 13 15 27 18 14 16 42 21 37

d u

19 13 15 27 18 14 16 42 21 37

d u

19 13 15 27 18 14 16 42 21 37 ~~(Exchange)~~

d u

19 13 15 16 18 14 27 42 21 37 (Exchange)

d u

19 13 15 16 18 14 27 42 21 37

d u

19 13 15 16 18 14 27 42 21 37

d,u

19 13 15 16 18 14 27 42 21 37

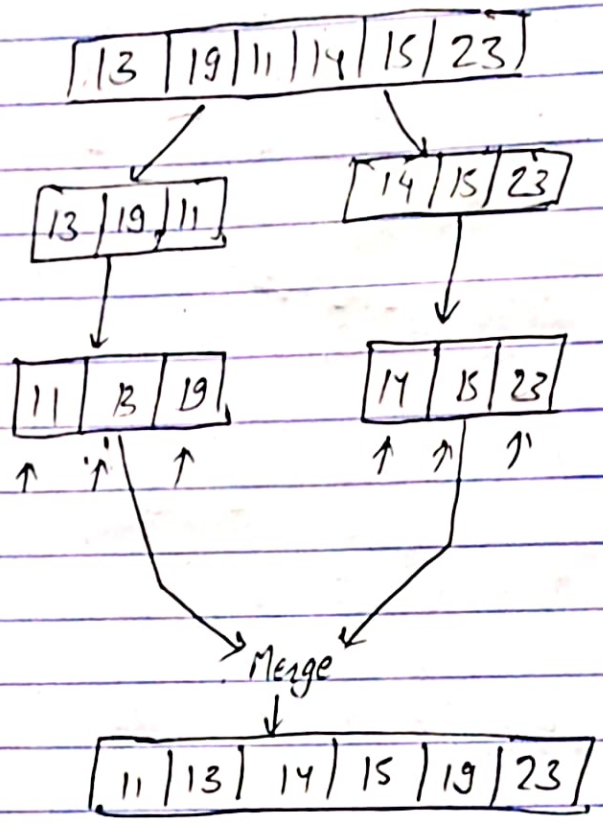
u d

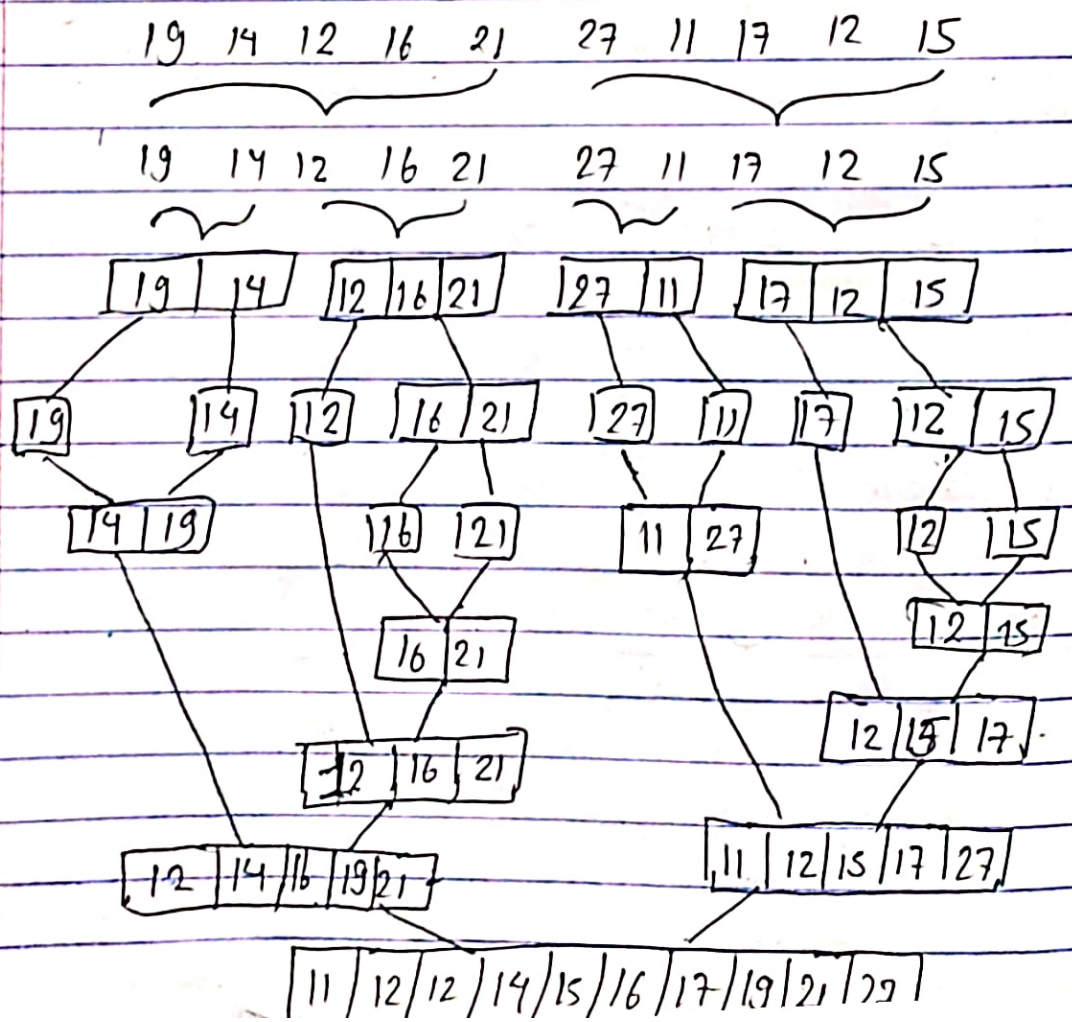
19 13 15 16 18 14 27 42 21 37

14 13 15 16 18 19 27 42 21 37

Scanned by CamScanner

Merge Sort





Scanned by CamScanner

```
int main ()
{   int a[] = {19, 14, 12, 16, 21, 27, 11, 17, 23, 15};
    int n = 10;
    MergeSort (a, 0, n);
    printf ("Array after sorting");
    displayArray (a, n);
}

void Mergesort (int a[], int l, int r)
{
    if (l<r)
    {   m = l+ (r-l)/2;
        MergeSort (a, l, m);
        MergeSort (a, m+1, r);
        Merge (a, l, m, r);
    }
}

void Merge (int a[], int l, int m, int r)
{   int i, j, k;
    int n1 = m-l+1;
    int n2 = r-(m+1); // r-m
    int L[n1], R[n2];
    for (i=0; i<n1; i++)    L[i] = a[l+i];
    for (i=0; i<n2; i++)    R[i] = a[m+1+i];
    i=0, j=0, k=l;
    while (i<n1 && j<n2)
    {   if (L[i] <= R[j]    a[k++] = L[i++];
        else    a[k++] = R[j++];
    }
    while (i<n1) { a[k++] = L[i++]; }
    while (j<n2) { a[k++] = R[j++]; }
}
```

Scanned by CamScanner

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

Scanned by CamScanner

Heap Sort

Complete Binary Tree (CBT)

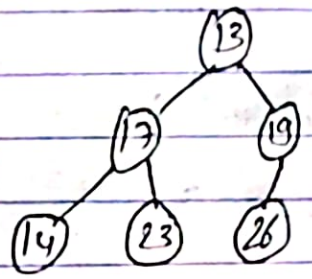


Fig: CBT

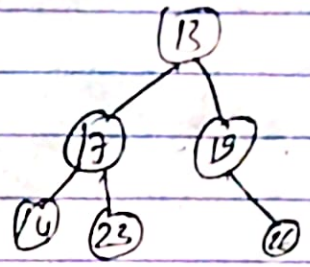
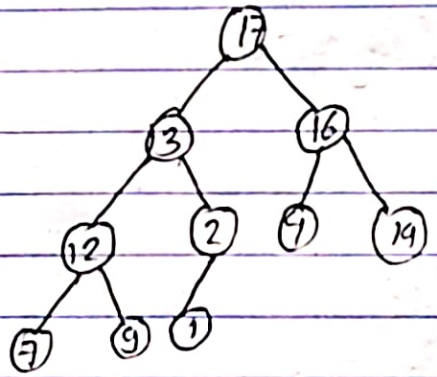


Fig: Not CBT

Max Heap



Array representation

	[0]	[1]	[2]	[3]	[4]	[5]
a	13	17	19	14	23	26

node i left child $2*i+1$
right child $2*i+2$

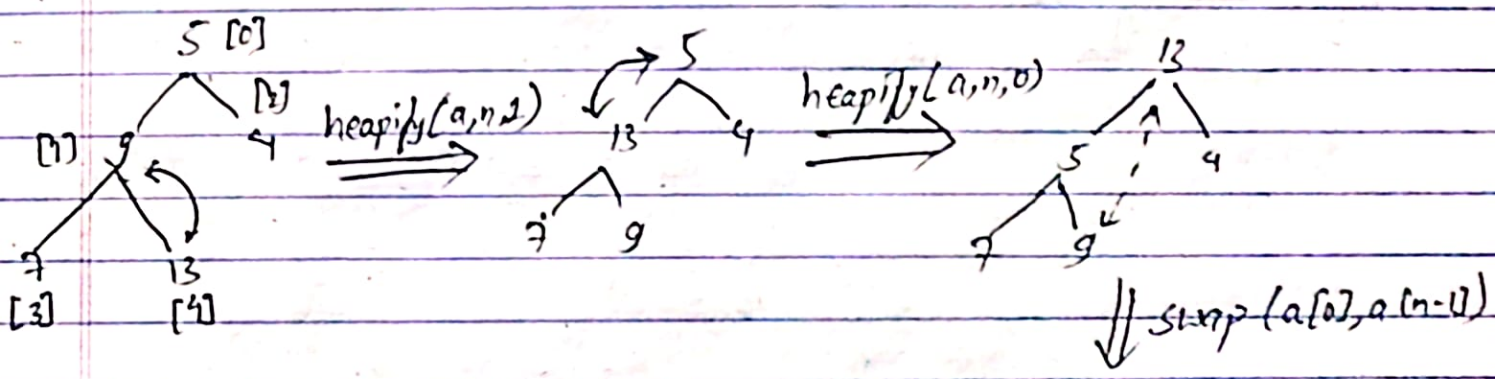
#

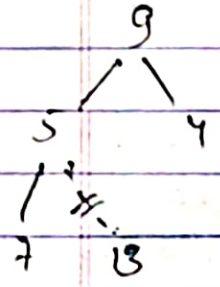
Heap Sort

```
#include <iostream>
using namespace std;
void display (int arr[], int n);
void heapify (int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i+1;
    int r = 2*i+2;
    if (l<n && arr[l] > arr[largest])
        largest = l;
    if (r<n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap (arr[i], arr[largest]);
        heapify (arr, n, largest);
    }
}
void heapsort (int arr[], int n)
{
    for (int i = n/2-1; i>=0; i--)
        heapify (arr, n, i);
    for (int i=n-1; i>=0; i--)
    {
        swap (arr[0], arr[i]);
        heapify (arr, i, 0);
    }
}
```

Scanned by CamScanner

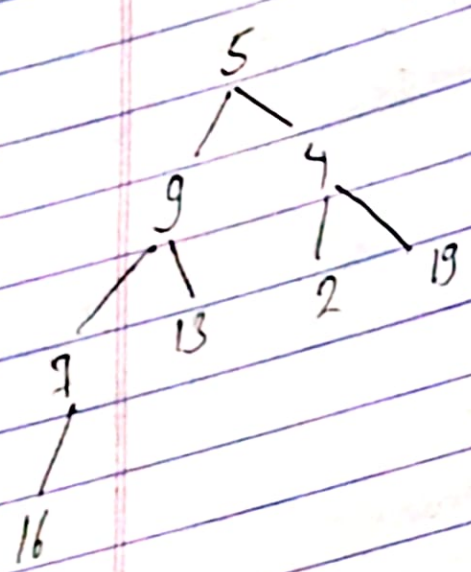
```
int main ()
{
    int arr [ ] = { 11, 9, 6, 8, 24, 36, 49 };
    cout << " Array Before Sorting\n";
    display ( arr, 7);
    heapsort (arr, 7);
    cout << " Array After Sorting \n";
    display (arr 7);
    return 0;
}
```



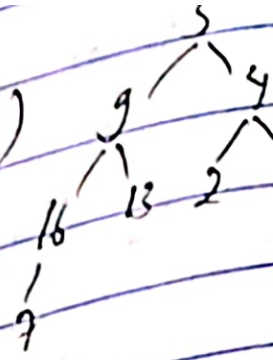


Scanned by CamScanner

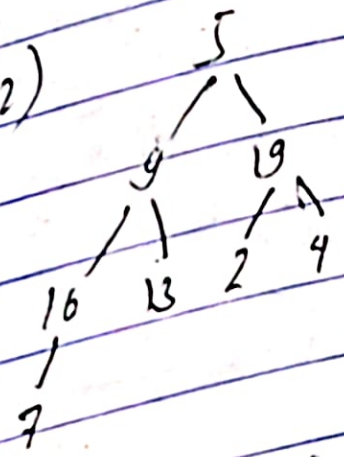
a [5 | 9 | 4 | 7 | 13 | 2 | 19 | 16]



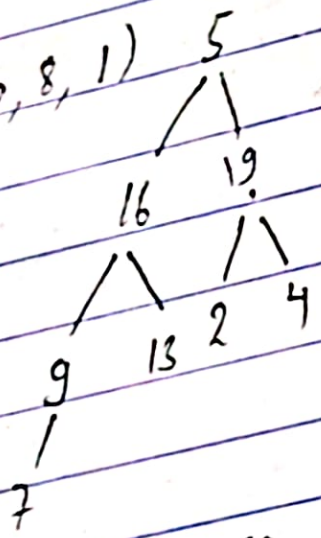
i=3 heapify (a,8,3)



i=2 heapify (a,8,2)

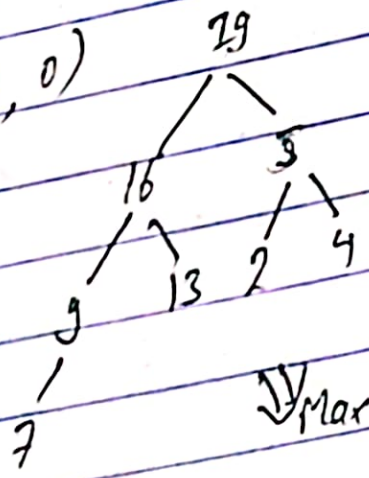


i=1 heapify (a,8,1)



⇒ heapify (a,8,3) (no

i=0 heapify (a,8,0)



⇒ heapify (a,8

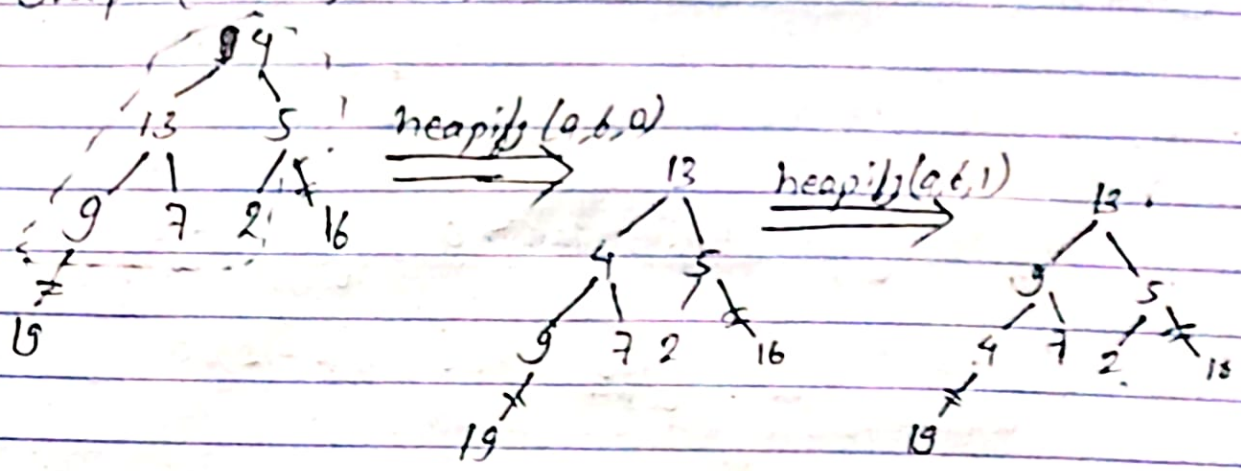
Max heap

swap (a[0], a[7])

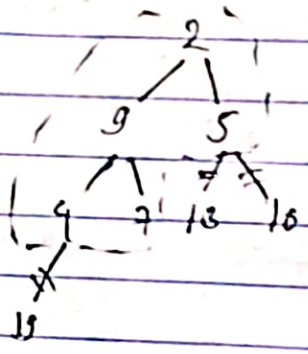
heapify (a,7,0)

heapify (a,7,1)

swap (a[0], a[6])



swap (a[0], a[5])



Scanned by CamScanner

Shell Sort

```
int shellSort (int arr[], int n]
{
    for (int gap = n/2 ; gap > 0 ; gap /= 2)
    {
        for (int i = gap ; i<n ; i+=1)
        {
            int temp = arr [i];
            int j ;
            for (j = i ; j>=gap && arr [j-gap] >temp;
                j -= gap)
                arr [j] = ~~temp~~ arr [j-gap];
            } arr[j] = temp;
    } }
    return 0;
}
void displayArray (int arr [], int n)
{   for (int i= 0 ; i<n ; i++)
        cout << arr[i] << " " ;
}
int main ()
{
    int arr [] = { 32, 65, 23, 44, 77, 37, 34, 54, 12, 135, };
    int n = 10;
    cout << "Array before sorting : \n";
    displayArray (arr, n);
    shellSort (arr, n);
    cout << "\n Array after sorting: \n";
    displayArray (arr, n);
    return 0;
}
```

Scanned by CamScanner

[0] [9]

arr | 32 | 65 | 25 | 44 | 77 | 37 | 34 | 54 | 12 | 13 | n = 10

gap = 5

i = 5 :- temp = arr[5] = 37

j = 5 :→ j >= gap && arr[j-gap] < temp

j = 0 → 0 <= gap

i = 6 :-

⋮

---

arr | 32 | 65 | 25 | 44 | 77 | 37 | 34 | 54 | 12 | 13 |

gap = 5

| 32 | 34 | 23 | 12 | 13 | 37 | 65 | 54 | 44 | 77 |

i = 2

gap = 2

| 23 | 34 | 32 | 12 | 13 | 37 | 65 | 54 | 44 | 77 |

i = 3

| 23 | 12 | 32 | 34 | 13 | 37 | 65 | 54 | 44 | 77 |

i = 4

j = 2 j = 4

| 23 | 12 | 13 | 34 | 32 | 37 | 65 | 54 | 44 | 77 | j = 4 | i = 4

| 13 | 12 | 23 | 34 | 32 | 37 | 65 | 54 | 44 | 77 | j = 2

Scanned by CamScanner

$i = 5$

13	12	23	34	32	37	65	54	44	77

$i = 5$

$j = 3$ $j = 5$ $j = 5$

13	12	23	34	32	37	65	54	44	77

$j = 5$

13	12	23	34	32	37	65	54	44	77

$j = 3$

$i = 6$

$j = 2$ $j = 4$ $j = 6$ $j = 6$

13	12	23	34	32	37	65	54	44	77

$i = 7$

$j = 3$ $j = 5$ $j = 7$ $j = 7$

13	12	23	34	32	37	65	54	44	77

$i = 8$

$j = 2$ $j = 4$ $j = 6$ $j = 8$ $j = 8$

13	12	23	34	32	37	44	54	65	77

$j = 9, 54$

$i = 9$

$j = 5$ $j = 5$ $j = 7$ $j = 9$ $j = 9$

13	12	23	34	32	37	44	54	65	77

gap = 1

Scanned by CamScanner

Searching

1) Linear Search

```
int search (int arr[], int n, int x)
{
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

Void main ()
{
    int arr[] = {37, 13, 41, 32, 27, 98}
    int n = 6;
    int x = 41;
    int pos = linearSearch (arr, n, x);
    if (pos == -1)
        printf ("Element not found");
    else
        printf ("Element is found at position %d", pos);
}
```

Scanned by CamScanner

2.) Binary Search

Recursive :-

```
int BinarySearch (int arr[], int l, int r, x)
{
    int m = (l+r)/2 ;   if (l <= r) {
    if (x == arr[m])
        return m ;
    else if ( x < arr[m]
        return (BinarySearch (arr, l, m-1, x));
    else
        return BinarySearch (arr, m+1, r, x);
}
```

Iterative :-

```
int BinarySearch (int arr[], int l, int r, int x)
{   int m ; ~~(l+r)/2~~ ;
    while (l <= r)
    {
        m = (l+r)/2 ;
        if (x == arr [m])   return (m);
        else if (x < arr[m])   r = m-1 ;
        else   l = m+1 ;
```

Scanned by CamScanner

Hashing Search Techniques

elements = { 17, 26, 33, 42, 94, 81, 45, 28, 19, 20 }

0	20
1	81
2	42
3	33
4	94
5	45
6	26
7	17
8	28
9	19

Hash table

key = Hash Function (element)

index = element mod HashTable Size

17 % 10 = 7

26 % 10 = 6

33 % 10 = 3

and so on

searchelement = 17

Collision

Linear Probing

elements = {17, 42, 93, 62, 87, 92, 43, 85, 86, 49}

0	
1	92
2	93
3	85
4	86
5	87
6	42
7	43
8	17
9	62

Hashfunction = (sum of elements) mod 10

index = (1+7) mod 10

(4+2) mod 10

(9+3) mod 10

(6+2) mod 10

(8+7) mod 10

(9+2) mod 10

(4+3) mod 10

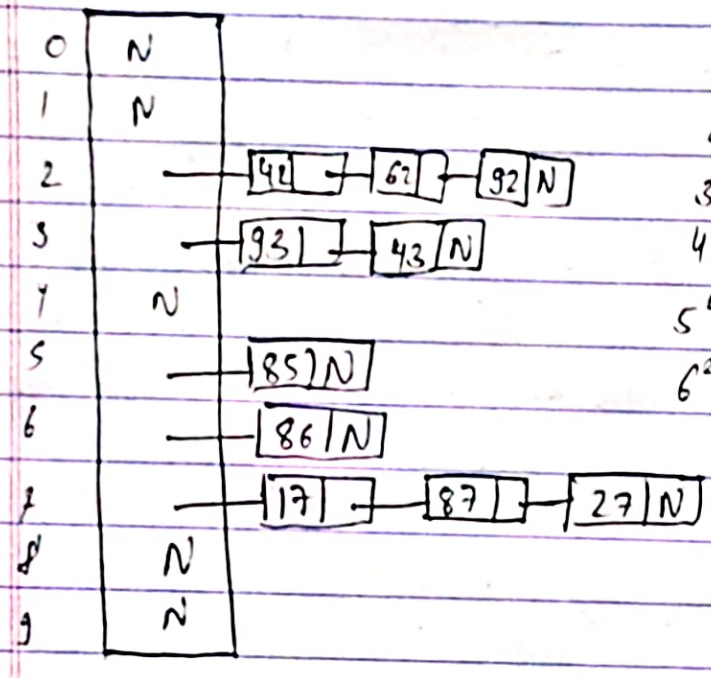
(8+5) mod 10

(8+6) mod 10

(4+9) mod 10

Scanned by CamScanner

Quadratic Probing



$1^2 = 1$

$2^2 = 4$

$3^2 = 9$

$4^2 = 16$

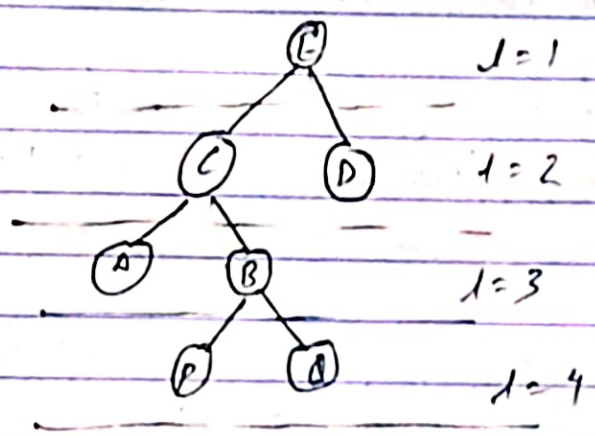
$5^2 = 25$

$6^2 = 36$

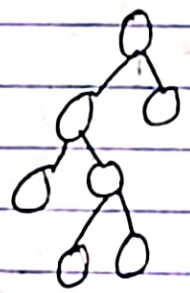
Scanned by CamScanner

Tress & Graph

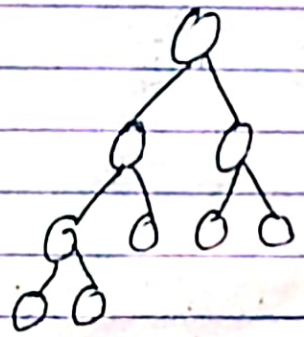
Height
Depth
Level



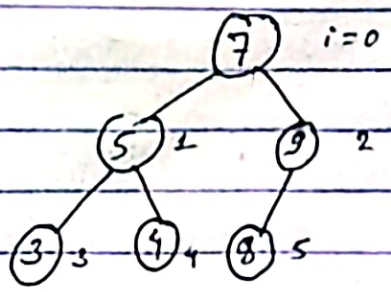
Strictly binary tree



Complete Binary Tree



Binary Search Tree (BST)



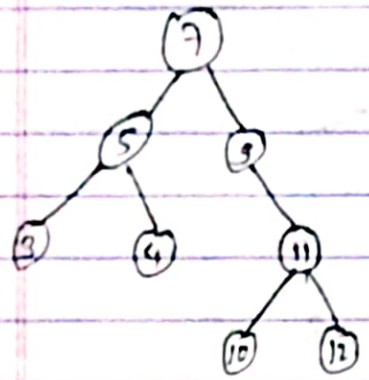
for : left child $2 \times i + 1$
right child $2 \times i + 2$

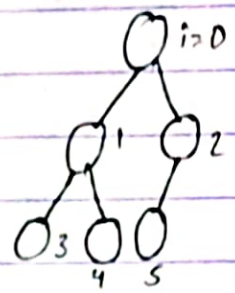
Scanned by CamScanner

Implementation of BST

- Array Implementation
  - Single Array
  - Multiple Array
- Link list Implementation

Single Array Implementation

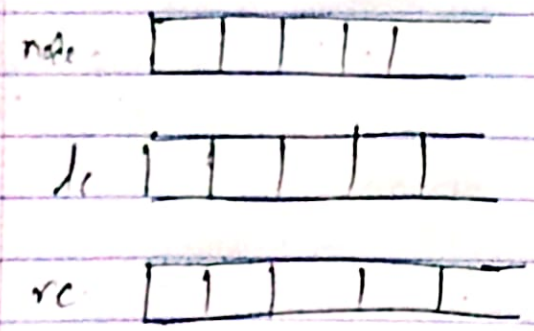




for i (index) left child $2*i+1$
right child $2*i+2$

7	5	9	3	4	'10'	11	'10'	'10'	'10'	'10'	'10'	'10'	10	12	'10'	'10'	'10'
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14			

Multiple Array



Linked List Representation

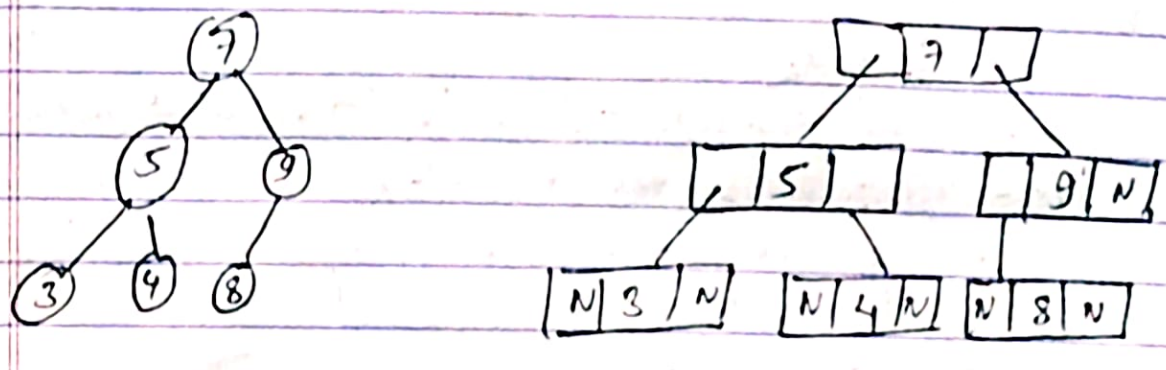


fig:- linked list representation of BST

Operations used in BST

1. Insertion
2. Deletion
3. Search

```
struct BSTnode
{
    struct BSTnode *left;
    int data;
    struct BSTnode *right;
}
void main ()
{ struct BSTnode *root = NULL;
    int a[] = {10, 12, 7, 6, 9, 2, 4, 18};
    for (i=0; i<8; i++)
        BSTInsert (a[i], &root);
}
```

Scanned by CamScanner

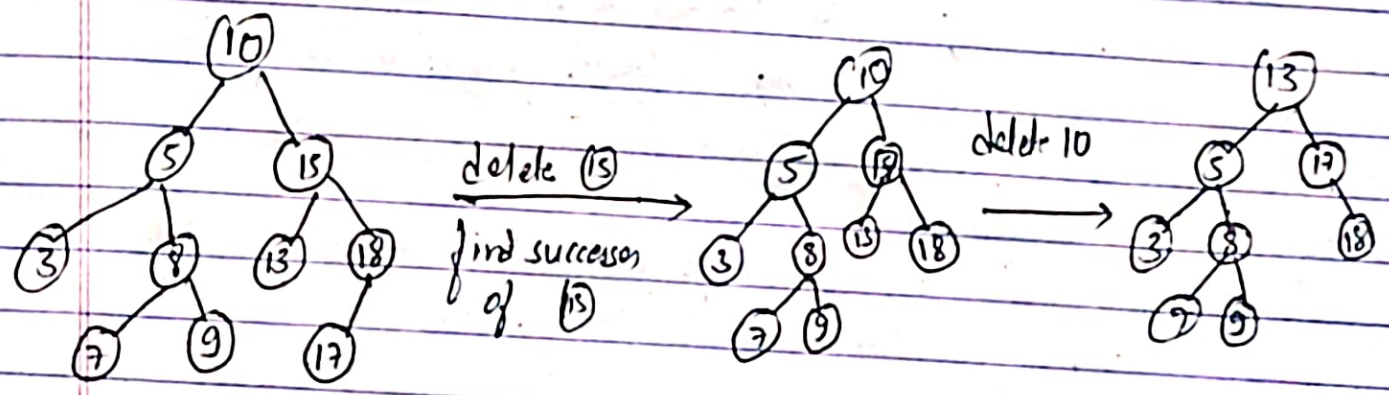
```
void BSTInsert (int val, struct BSTnode **proot)
{
    struct BSTnode *temp;
    temp = (struct BSTnode *) malloc (sizeof(struct BSTnode));
    temp -> left = temp->right = NULL;
    temp -> data = val;
    if (*proot == NULL)
        *proot = temp;
    else if (val < (*proot) -> data)
        BSTInsert (val, & ((*proot)->left));
    else
        BSTInsert (val, & ((*proot)->right));
}
```

```
void BSTInsert (int val, struct BSTnode **proot)
{
    if (*proot == NULL)
    {
        *proot = (struct BSTnode *) malloc (sizeof(struct BSTnode));
        (*proot)->left = (*proot)->right = NULL;
        (*proot) -> ~~val~~ data = val;
    }
    else if (val < (*proot) -> data)
```

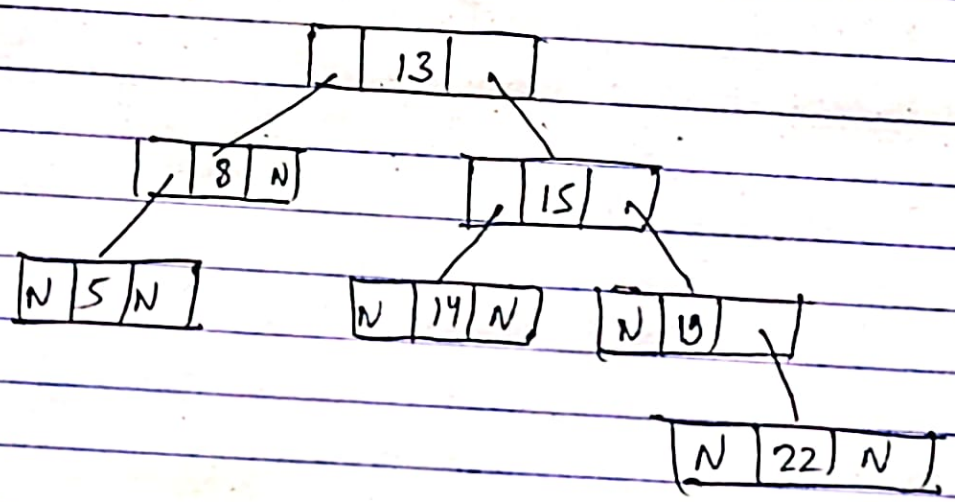
Scanned by CamScanner

Deletion

- → node not found (delete 21)
- → leaf node (delete 7)
- → single child (delete 18)
- → two child (delete 10)



~~Bin~~ Searching



Scanned by CamScanner

```
void SearchBST (struct BSTnode **proot, int val)
{
    struct BSTnode * current = *proot;
    while (current != NULL)
    {
        if (current -> data == val)
        {
            printf ("Node found \n");
            return ;
        }
        else if (val < current -> data)
            current = current -> left ;
        else
            current = current -> right ;
    }
    printf ("Node not found \n");
    return ;
}
```

Scanned by CamScanner

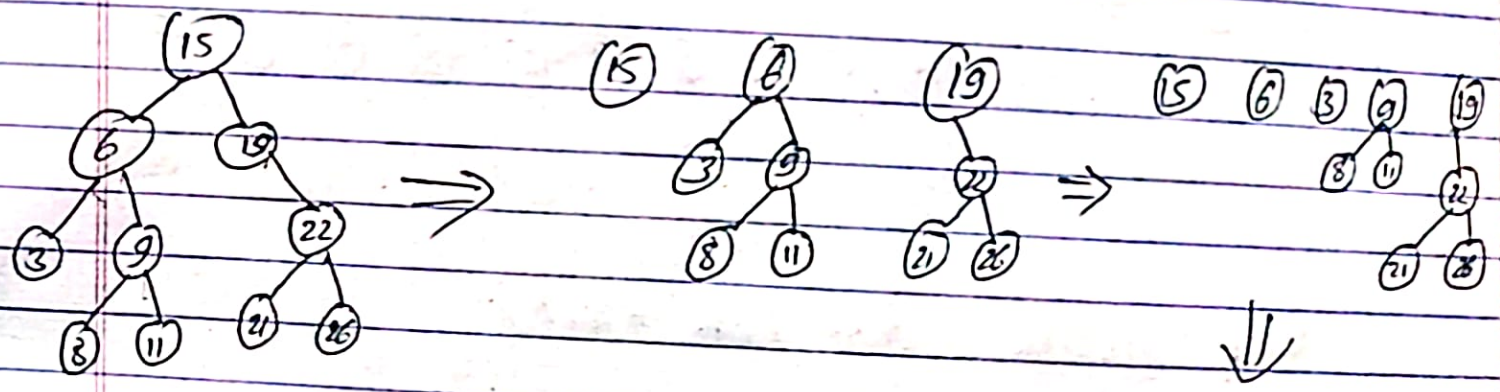
Traversal of BST

i) Preorder traversal CLR
ii) Inorder traversal LCR
iii) Postorder traversal LRC

C → root

~~Ex~~ Preorder Traversal

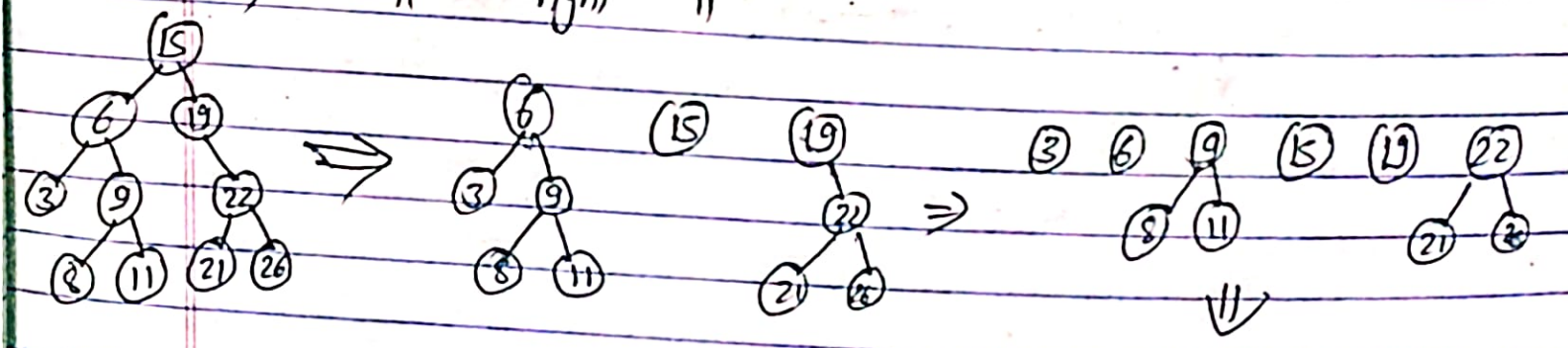
1) Visit root
2.) Traverse left subtree in Preorder
3.) Traverse right subtree in Preorder



15 6 3 9 8 11 19 22 21 26

Inorder Traversal

1.) Traverse left subtree
2.) " root "
3) " right "

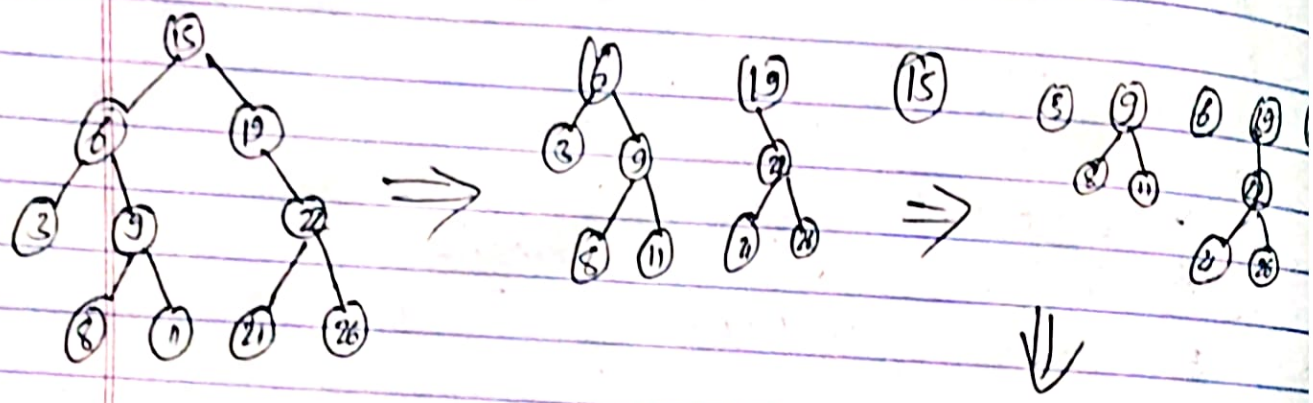


3 6 8 9 11 15 19 21 22 26

Scanned by CamScanner

Postorder Traversal

1.) Traverse left
2.) right
3.) root



3, 8, 11, 9, 6, 21, 26, 22, 19, 15

```
void preorder (struct node *root)
{
    if (root != NULL)
    {
        printf ("%d \t", root -> data);
        preorder (root -> left);
        // preorder (root -> left);
        preorder (root -> right);
    }
}
```

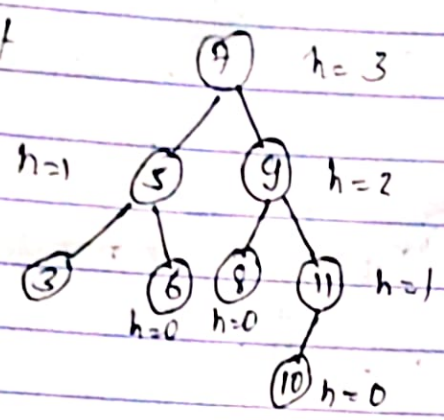
Scanned by CamScanner

```
void inorder (struct node * root)
{
    if (root != NULL)
    {
        inorder (root -> left);
        printf ("%d\t", root -> data);
        inorder (root -> right);
    }
}

void postorder (struct node *root)
{
    if (root != NULL)
    {
        postorder (root -> left);
        postorder (root -> right);
        printf ("%d\t", root -> data);
    }
}
```

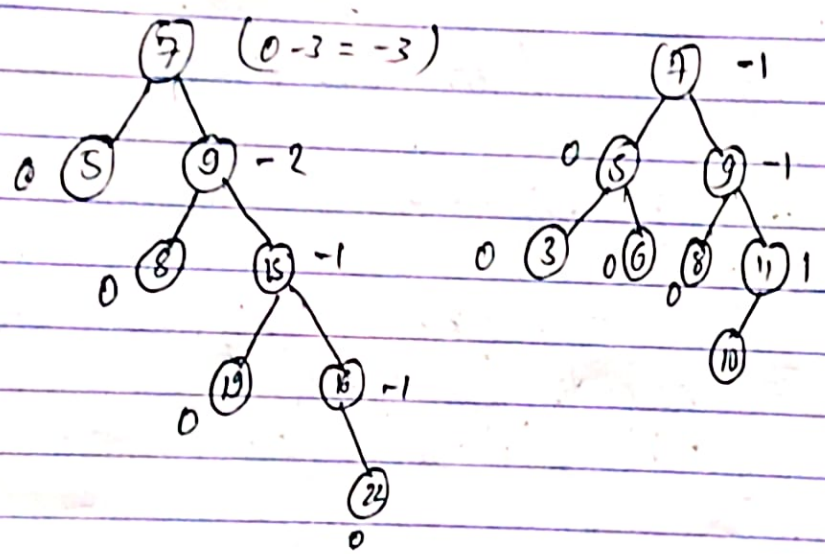
Scanned by CamScanner

Balanced BST (AVL Tree)

Height

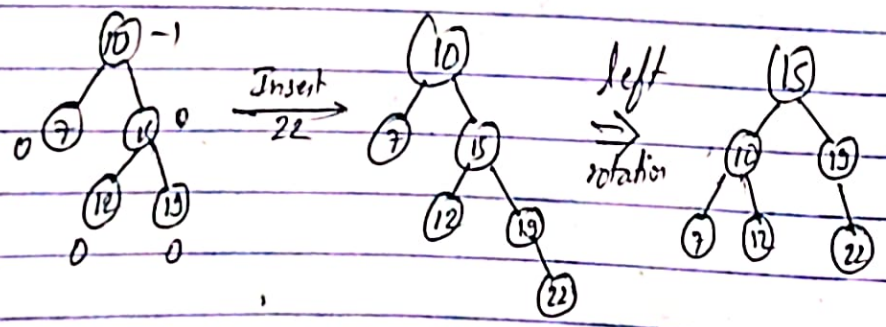


#

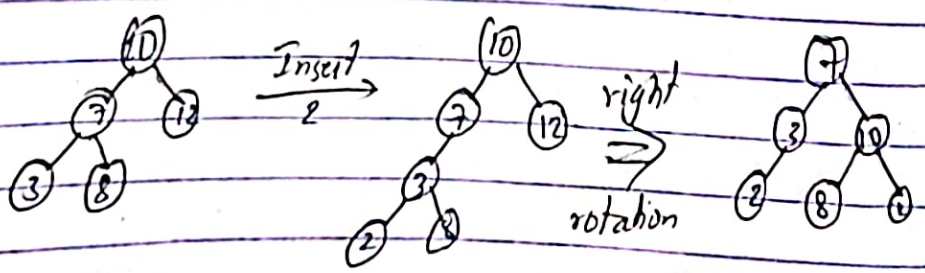


Insertion of a node in an AVL Tree

Case I



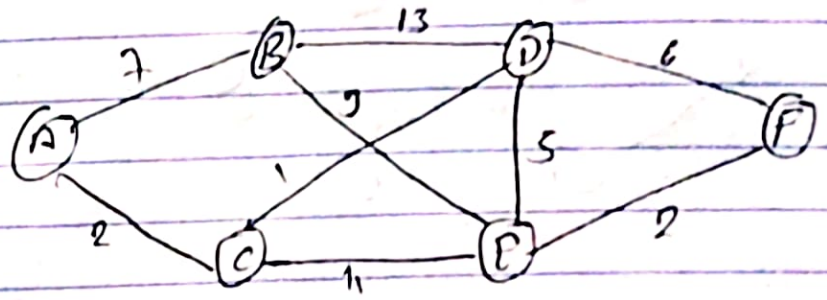
Case II



Scanned by CamScanner

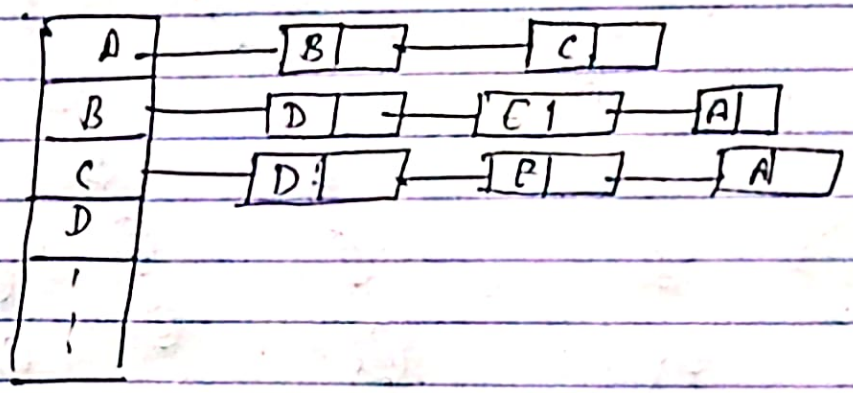
Graph

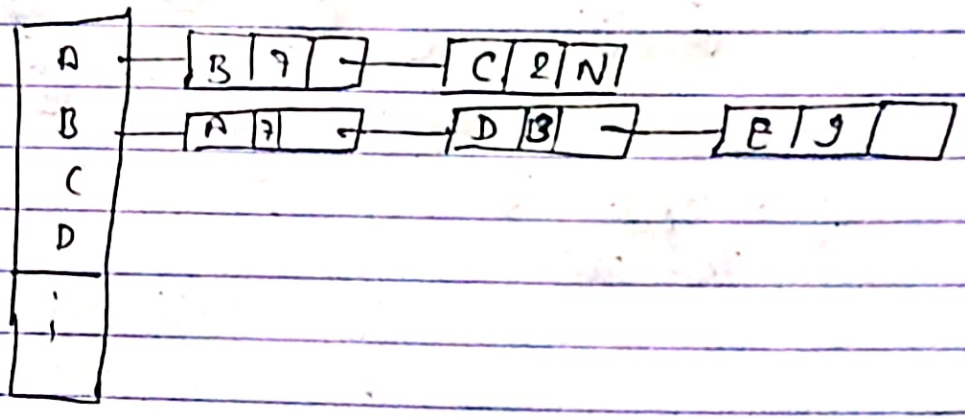
$G = (V, E)$



Representation of Graph

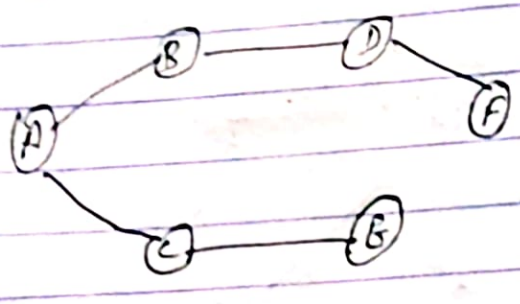
- → Adjacency matrix representation
- → Adjacency list representation





Scanned by CamScanner

Spanning Tree



Graph Traversal

- → Depth First traversal
- → Breadth " " "

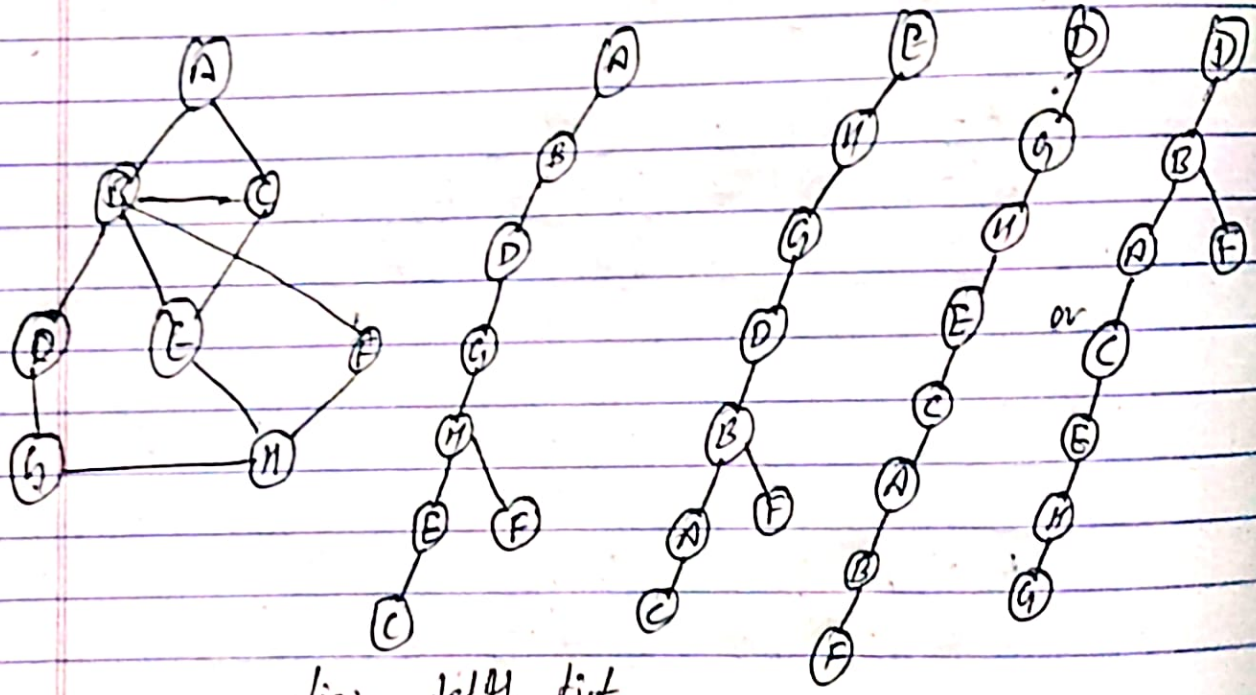
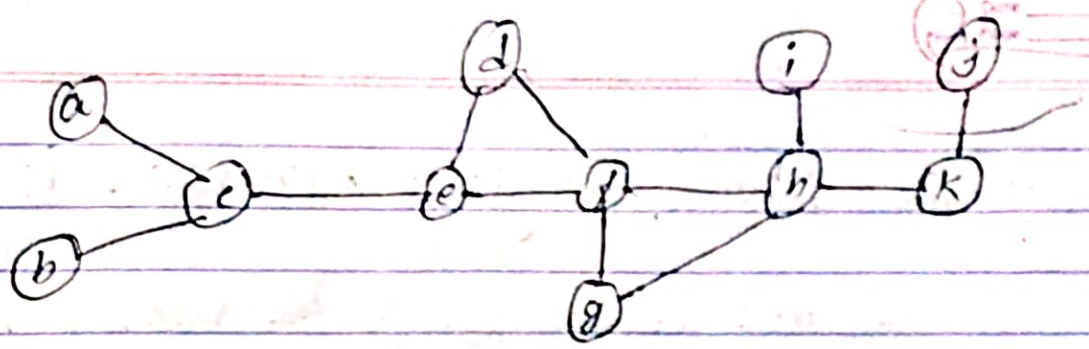
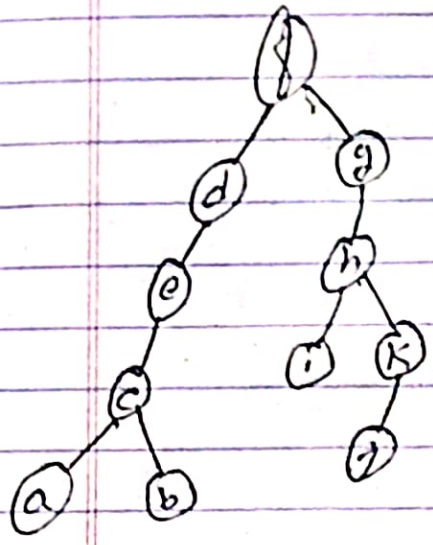
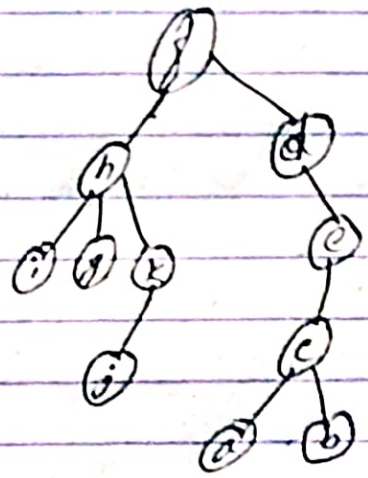


fig:- depth first traversal

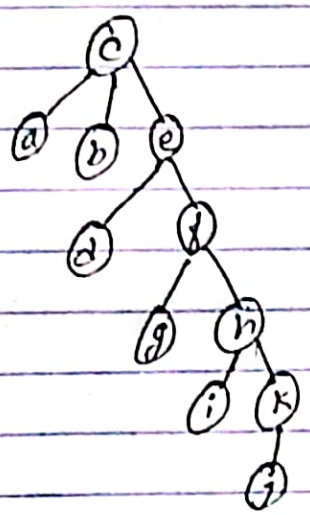




or,



Breadth first



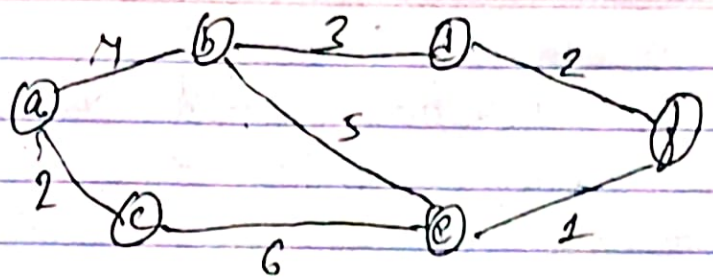
Scanned by CamScanner

```
procedure DFS (G: graph with vertices v1 v2 v3....
{
    T = tree consisting only the vertex v1
    visit (v1)
}
procedure visit (v: a vertex in G)
{
    for (each vertex w adjacent to v and not yet i
    {
        add w and edge {v, w} to T
        visit (w)
    }
}
```

Minimum Spanning Tree

→ Prim's Algorithm
→ Kruskal's Algorithm

Scanned by CamScanner



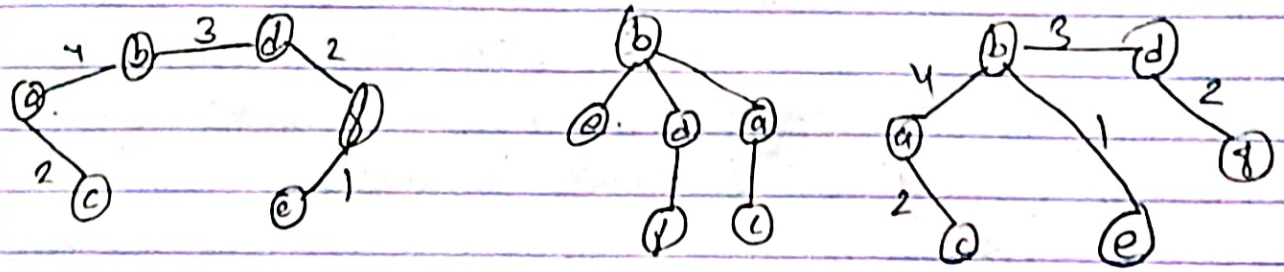
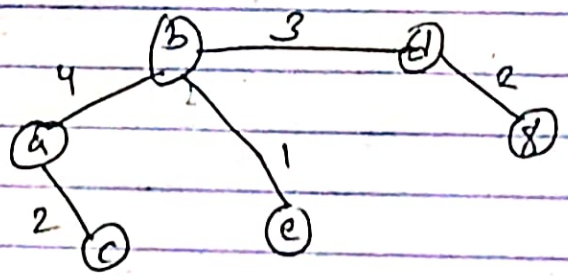
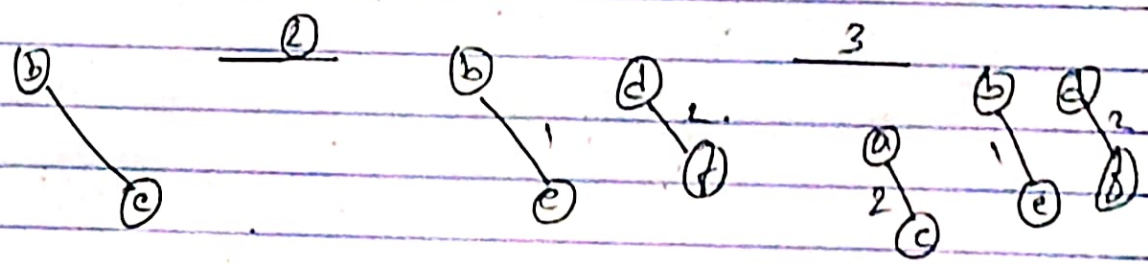
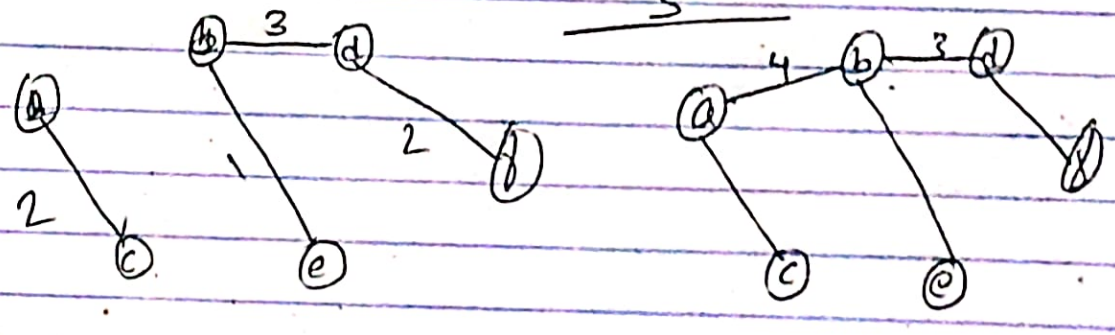


fig:- Minimum Spanning tree of Graph $G = (V, E)$ using Prim's algorithm



Step 2





Prims Algorithm

Input : G : weighted graph with n vertices

T = a minimum - weight edge

for (i = 1 to n-2)

{

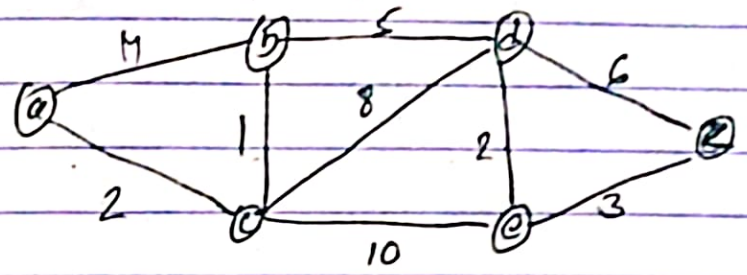
e = an edge of minimum weight incident to a vertex in T and not forming a cycle in T if added to T.

T = T with e added

}

Scanned by CamScanner

Dijkstra's Algorithm (Shortest Path Problem)



Find shortest Path from a to z

$S = \phi$

$l(a) = 0$

$l(b) = l(c) = l(d) = l(e) = l(z) = \infty$ /* large number */

First iteration

$S = \{a\}$
- b $l(b) = 4$
- c $l(c) = 2$
- d $l(d) = \infty$
- e $l(e) = \infty$
- z $l(z) = \infty$

Second

$S\{a, c\}$
- b $l(b) = 3$
- d $l(d) = 10$
- e $l(e) = 12$
- z $l(z) = \infty$

Third

$S\{a, c, b\}$
- d $l(d) = 8$
- e $l(e) = \infty$
- z $l(z) = \infty$

Fourth

$S\{a, c, b, d\}$
- e $l(e) = 10$
- z $l(z) = 14$

Fifth

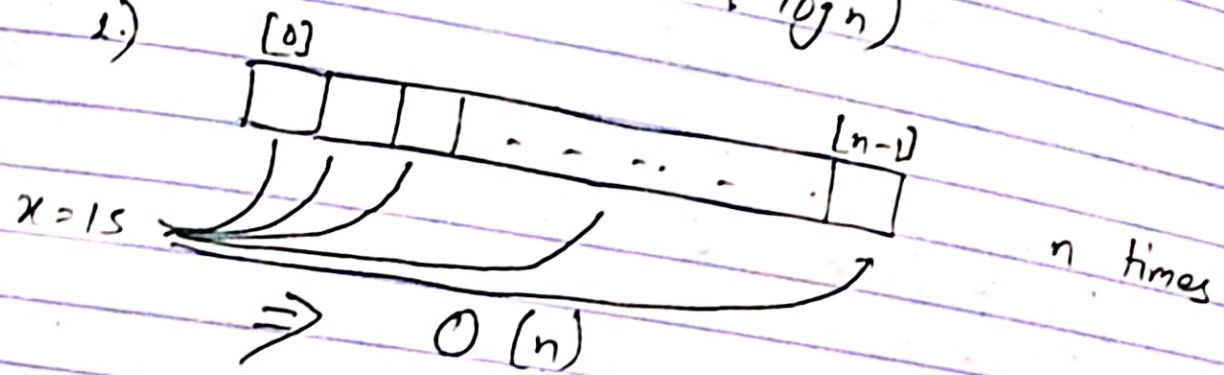
$S\{a, c, b, d, e\}$ — z $l(z) = 13$

Scanned by CamScanner

Big O Notation

1.) Linear Search $O(n)$
2.) Binary Search $O(\log n)$

1.)



$\Rightarrow O(n)$

$n \longrightarrow 1$ comparison

$\downarrow$

$n/2 \longrightarrow 2$ com

$\downarrow$

$n/4 \longrightarrow 3^{rd}$ comp

$\vdots$

$1 \longrightarrow k^{th}$ comp

$\Rightarrow O(k)$

$\Rightarrow O(\log n)$

$$\frac{n}{1} \quad \frac{n}{2} \quad \frac{n}{2^2} \quad \frac{n}{2^3} \quad \frac{n}{2^3} \quad \frac{n}{2^k} \quad \frac{n}{2^k} = 1$$

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

$$\therefore n = 2^k$$

$$k = \log_e(n)$$

Scanned by CamScanner

Bubble $O(n^2)$ $n = 100 \rightarrow 100$ steps

Quick Sort $O(n \log n)$ $10 \times \log(10) = 40$ steps

$f(n) = 3n^2 + n + 2$

$g(n) = n^2$

$f(n)$ is $O(g(n))$

$3n^2 + n + 2 \leq c \cdot n^2$ for $n \geq n_0$

$3n^2 + n + 2 \leq 7n^2$ for all $n \geq 1$

$3n^2 + n + 2 \leq 3n^2 + n^2 + 3n^2$

Scanned by CamScanner