Data Structures & Algorithms

PAVAN [redacted]

Course Outcomes:-

CO1:- Explain sorting and searching Techniques (K_2)

CO2:- Demostrate single linked link, Double linked list (K_3)

CO3:- Interpret the basic operations on stack's and queus (K_3)

CO4:- Demostrate Binary ~~theories~~ trees and Binary search Trees (K_3)

CO5:- Compare Binary trees and self balanced trees with appropriate examples (K_4)

CO6:- Illustrate various graph algorithms (K_3)

Scanned by CamScanner

→ Algorithm:-

It is defined as a step by step procedure to perform a task.

we can write algorithm in any one of the following three ways

① Sequence

② Selection / Decision

③ Iteration / Repetition

→ write an algorithm to swap 2 numbers using a third variable.

Step-1:- Start

Step-2:- Declare a, b, t values

Step-3:- Input a, b.

Step-4:- Perform a=t
a=b
b=t

Step-5:- print a, b

Step-6:- Stop.

→ Selection:-

Here a condition is checked if it is true then some statments are executed otherwise someother statments are executed.

→ write an algorithm to find largest of 2 numbers

Step-1:- Start

Step-2:- Input a, b.

Step-3:- If (a > b)
print "a is big"
else
print "b is big".

Step-4:- Stop.

Scanned by CamScanner

→ Write an algorithm to accept year of birth and check whether the persons birthyear is leap year or not.

Step-1 :- Start
Step-2 :- Declare year
step-3 :- Input y
Step-4 :- if(y%.4==0 && y%.100!=0 || y%.4==0)
print "Given year is leap year".
else
print "Not leap year".
step-5 :- stop.

Iteration / Repetition :-

If the algorithm want to access more no of values a given structure (data structure) then we have to iterate the elements in the structure.

→ write an algorithm to print values from 1 to n

Step-1 :- start
Step-2 :- Declare i
Step-3 :- Read i=1
Step-4 :- Repeat the following steps until i
step-5 :- print i
step-6 :- Calculate i++
step-7 :- stop.

→ write an algorithm to find sum of digits of n natural numbers.

Step-1 :- start
step-2 :- Declare i, n, sum
step-3 :- Read i, n, sum=0, i=1
Step-4 :- Repeat the following steps
until i<=n

Scanned by CamScanner

step-5:- sum=sum+i

step-6:- calculate i++

step-7:- print sum

step-8:- stop.

Data Structure:- It is defined as the organizing data in a specific manner.

Eg:- Array is an example of data structure

Structures and Unions also examples.

Searching:-

Searching can be performed on 2 ways on 1D arrays

(i) Linear - If the Key element is searched in the given list of array elements then it is called linear search (or) sequential search.

Algorithm:-

Step-1:- Start

Step-2:- Declare a, i, n, Key, found

Step-3:- Read a, i=0, n, Key, found=0

Step-4:- Repeat step 5 until i<n

Step-5:- if a[i] matches with Key

print "element found"

else

print "element not found".

Step-6:- Stop.

Program:-

```
#include<stdio.h>
main()
{
    int a[i], a[10], i, n, Key, found=0;
    printf("enter 'n' size");
    scanf("%d", &n);
    Printf("enter array elements");
    for(i=0; i<n; i++)
    {
```


Scanned by CamScanner

```
scanf("%d", &a[i]);
}
printf("enter Key value");
scanf("%d", &Key);
for(i=0; i<n; i++)
{
  if(a[i]==key)
  {
    printf("Element found");
    found=1;
    break;
  }
}
if(found==0)
  printf("element not found");
}
```

Binary searching:-

This searching methodology divides the array i 2 types by calculating the mid value. It perform searching only on sorting array.

→
```
#include<stdio.h>
main()
{ int i,n, first=0, last=n-1, Key, a[10], mid;
  printf("enter n value");
  Scanf("%d", &n);
  for(i=0; i<n; i++)
  {
    printf("enter elements in a[%d]", i);
    scanf("%d", &a[i]);
  }
printf("enter Key value");
```


Scanned by CamScanner

```
scanf("%d", &key);
while (first<=last)
{
  mid = (first+last)/2;
  if(a[mid] < key)
  first = mid+1;
  else if (a[mid]==key)
  {
   printf("element found at %d", mid);
   Break;
  }
  if(first>last)
  else
  last = mid-1;
  }
  if(first > last)
  printf("element not found");
  getch();
}
```

$f(n) = \log n$.

Data structure:-

A Data structure is a group of elements that are put together under one name and which defines a particular way of storing and organizing data in a computer. So that it can be used efficiently.

Eg:- Arrays, linked list, Stacks, Trees, Queues etc.

Applications of data structures

1. Compiler Design
2. Numerical Analysis
3. Artificial Intelligence
4. O.S
5. DBMS (Data Base managment System)
6. Graphics.

Scanned by CamScanner

Operations of data structure:

(i) Traversing:- It means to access each data item exactly once so that it can be processed.

(ii) Searching:- It is used to find the location of one (or) more data items that satisfied the given constrain.

(iii) Insertion:- It is used to add new data items to given list of data items.

(iv) delection:- It is used to remove a particular data item from given collection of data item.

(v) sorting:- Data items may be arranged in some order like ascending order (or) descending order.

(vi) Merging:- List of two sort data items can be combined to form a single list of sorted data items.

→ Approaches of designing an algorithm. A complex algorithm is divided into smaller units called modules. This process is called modularization.

The two strategies that are used to write algorithms are (i) top-down

(ii) Bottom-up.

Analyzing an algorithm:-

We can analyse the efficiency of algorithm using two factors (i) Time complexity

(ii) Space complexity.

Time Complexity:

It is the running time of the program as a function of Input size.

Space Complexity:-

It is the amount of computer memory that

Scanned by CamScanner

of input size.

The space needed by a program depends on the following 2 parts (i) fixed part (It includes the space for storing Instructions, constants, variables and structured varbles.

(ii) Variable part:- It is the space needed for recursion stack and for structured variables that are allogated dynamically.

Worst case time complexity

This denotes the behaviour of an algorithm with respect to the worst possible of the input instance. It is an upper bound on the running time for any input.

Average case time Complexity:-

It is an estimate of the running time for any average Input.

Best case time Complexity:-

It is used to analyze an algorithm under optimal conditions. It is a lower bound on the running time for any Input.

Calculating time and space complexity.

The time and space complexity can be expressed using a function f(n) where n is the Input size. If a function is linear without any loop (or) recursion the efficiency of that algorithm (or) running time of algorithm can be given as no. of Instructions it contain.

There are different types of loops. If the algorithm contains loops then the efficiency of the algorithm may vary depending upon no. of loops and running time of each loop.

Scanned by CamScanner

for(i=0; i<10; i++)

In the case of linear loops the no. of iterated by the loop is called as loop factor. The f(n) = loop factor.

In the above example f(n)=10

So in general terms the function for linear loops f(n)=n.

Eg:- for(i=1; i<=100; i+=2)

Here the no. of iterations is half the no. of loop. ∴ The efficiency is f(n)=n/2

2) Logarithmic loop:-

The loop controlling variable is either multiplied divided for each iteration of the loop.

for(i=1; i<=1000; i*=2)

for(i=1000; i>=1; i/=2)

Both the loops are iterated 10 times it is expre logarithm notation as $\log_2^{1000}$, where n is 1000 and the loop controlling variables value for each ite Therefore in general terms the function for logarit loops is f(n)=log n

3) Nested loops:-

We can find the complexity of nested loops by mul no. of iterations in outer loop with no. of iterations Inner loops.

There are 3 types of Nested loops

(i) linear logarithmic loop.

(ii) Eg:- for(i=0; i<10; i++) for(j=0; j<=10; j*=2)

→ The no. of iterations of outer loop is n. And the of iterations of inner loop is log n. In general ter

Scanned by CamScanner

In this loop, the no. of iterations of outer loop are equ to no. of iterations of inner loop. The efficiency here is $f(n) = n \cdot n = n^2$.

Eg:- for(i=0; i<10; i++)
    for(j=0; j<10; j++)

(iii) Dependent Quadratic loop:-

In this loop, the no. of iterations in the inner loop are dependent on outer loop.

Eg:- for(i=0; i<10; i++)
    for(j=0; j<=i; j++) } Flyod's Triangle

We can calculate the average no. of iterations of inner loop by adding all the iterations of inner loop by no. of iterations of Inner loop.

i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9
j=0	j=0,1	0,1,2	0,1,2,3						
1	2	3	4	5	6	7	8	9	10

$$\rightarrow \frac{1+2+3+4+5+6+7+8+9+10}{10} = 5.5 = \frac{11}{2} = \frac{10+1}{2} = \frac{n+1}{2}$$

$$\therefore f(n) = n \cdot \frac{n+1}{2}$$

Asymptotic Notations :-

There are 3 types of Asymptotic Notations. These are used to determine the behaviour of an algorithm.

Big O Notation :-

This Notation determines the worst case complexity of an algorithm.

Let us assume $f(n)$ and $g(n)$ are two functions and 'c' is a constant then the big O NOtation is defined as $f(n) = O(g(n))$ if and only if

Scanned by CamScanner

2) Big Ω (omega) Notation:-

This notation determines the best case com algorithm.

Let us assume f(n) and g(n) are two functions and Constant, then Big Ω Notation is defined as

$f(n) = \Omega(g(n))$

$f(n) \geq c \cdot g(n)$ for $c > 0 \& n \geq 0$

3) Big θ (Theta) Notation:-

This notation determines the average case compl of an algorithm. Let us assume f(n) and g(n) as f and c_1 and c_2 are two constants then θ is defi $f(n) = \theta(g(n))$ if and only if $c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$.

i.e $f(n) \geq c_1 \cdot g(n) \rightarrow \Omega$

$f(n) \leq c_2 \cdot g(n) \rightarrow O$

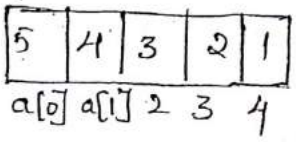
Hence Big θ lies btw Ω and O.

Sorting Techniques:-

Bubble sorting

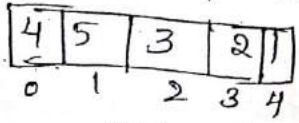
In Bubble sorting, consecutive adjacent pair of e in the array are compared with eachother if t element at lower index is greater than the ele at higher index then the two elements are int changed. this process will continue till the list

Eg:-



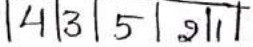
a[0] > a[1]

5 > 4 then swap



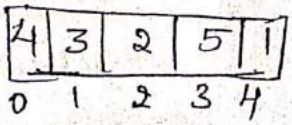
a[1] > a[2]

5 > 3



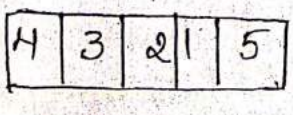
a[2] > a[3]

5 > 2



a[3] > a[4]

5 > 1



Scanned by CamScanner

cycle-1: a[0]>a[1]

4>3

3	4	2	1	5

a[1]>a[2]

4>2

3	2	4	1	5

a[2]>a[3]

3	2	1	4	5

a[3]>a[4]

4>5 (wrong)

cycle-2: | 3 | 2 | 1 | 4 | 5 |

a[0]>a[1]

3>2

2	3	1	4	5

a[1]>a[2]

3>1

2	1	3	4	5

a[2]>a[3]

3>4 (W)

a[3]>a[4]

4>5 (W)

Cycle-3:

2	1	3	4	5

a[0]>a[1]

2>1

1	2	3	4	5

a[1]>a[2] (W

a[2]>a[3] (W

a[3]>a[4] (W

Program:-

complexity is n^2

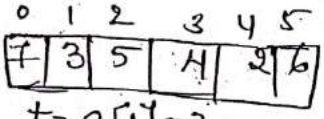
```
#include<stdio.h>
Void main()
{ int a[20],i,j,t,n;
  printf("enter n value");
  scanf("%d",&n);
  printf("enter array elements");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=0;i<n;i++)
  {
   for(j=0;j<n-i-1;j++)
    {
     if(a[j]>a[j+1])
      {
      t=a[j];
      a[j]=a[j+1];
      a[j+1]=t;
      }
    }
  }
printf("elements after sorting");
for(i=0;i<n;i++)
Printf("%d ...
```


Scanned by CamScanner

In Insertion sort technic the second element is compared to first element, if wrong then Interchange the elements ~~then~~ then third element ~~will~~ is c

In Insertion sort technic, 2nd element compar 1st element, if 2nd element < 1st element then i 2 elements, then 3rd element compared with < 1st element and placed the element in 1st pl This process is repeated till end of array.

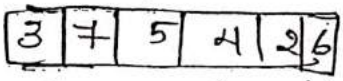
Eg:-



t = a[1] = 3

t < a[0]

5 < 7



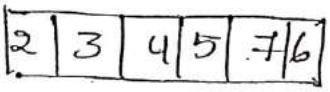
t = a[2] = 5

t < a[1]

5 < 7

t < a[3]

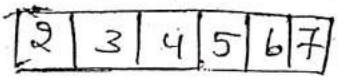
5 < 3 (W)



t = a[5] = 6

t < a[4]

6 < 7

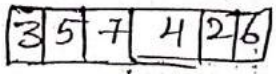


t < a[3] (F)

t < a[2] (F)

t < a[1] (F)

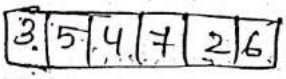
t < a[0] (F)



t = a[3] = 4

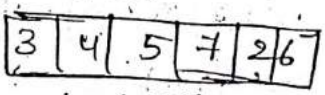
t < a[2]

4 < 7



t < a[1]

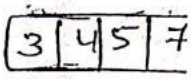
4 < 5



t < a[0]

t < 3

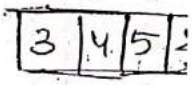
4 < 3 (W)



t = a[4

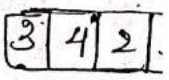
t < a[

2 < 7



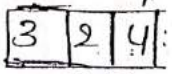
t < a

2 < 5



t < a[

2 < 4



t < a[0]

2 < 3

Scanned by CamScanner

Program logic:-

```
for(i=1; i<n; i++)
{
  t=a[i];
  j=i-1;
  while(j>=0 && a[j]>t)
  {
    a[j+1]=a[j];
    j=j-1;
  }
  a[j+1]=t;
}
```

a[0]	1	2	3	4	5
7	3	5	4	2	6

t = a[1], j = 0

a[j] > t

a[0] > a[1]

7 > 3

a[1] = a[0];

j = 0-1 = -1 (w)

a[j+1] = a[-1+1]

t = a[0]

t = 3

3	7	5	4	2	6

t = a[2], j = 1;

while(j>=0 && a[j]>a[j+1])

7 > 5

a[2] = a[1]

j = 1-1 = 0

a[j+1] = t

t = a[0+1]

t = a[1] = 5

a[2] = 7

3	5	7	4	2	6

t = a[3], j = 2

a[2] > a[3]

7 > 4.

a[3] = a[2]

7 = 4

j = 1

a[1] = t;

t = 4

0	1	2	3	4	5
3	5	4	7	2	6

t = a[4], j = 3

a[3] > a[4]

a[4] = a[3], j = 2

2 = 7

a[4] = 7 a[3] = t

t = 4

3	5	4	2	7	6

t = a[5], j = 4

a[4] > a[5] j = 3

7 > 6

a[5] = a[4] a[4] = t

a[4] = 6

a[5] = 7

3	5	4	2	6	7

a[2] = t, j = 1

a[1] > a[2]

5 > 4

a[2] = a[1] | j = 0, a[1] = 4

a[2] = 5 | t[1] = 4

3	4	5	2	6	7

a[3] = t, j = 2

a[2] > a[3]

5 > 2

a[3] = a[2] | a[2] = t, a[2] = 2

a[3] = 5

3	4	2	5	6	7

a[2] > a[1]

4 > 2 a[1] = t

a[2] = a[1]

Scanned by CamScanner

3	2	4	5	6	7

t=a[1]; j=0

a[0] > a[1]

3 > 2

a[1]=a[0], j=j-1 = 0-1 = -1

a[1]=3

a[j+1]=t ⇒ t=a[0]

⇒ a[0]=2

2	3	4	5	6	7

Selection sorting:-

It is the process of finding minimum to element every time and interchange it with index element. for every iteration the smaller comes to the begining of the array.

Code:-

```
for(i=0; i<n-1; i++)
 { small=i;
   for(j=i+1; j<n; j++)
   {
     if(a[j]<a[small])
       small=j;
   }
   t=a[i];
   a[i]=a[small];
   a[small]=t;
 }
```

Eg:-

9	6	7	2	1	3

i=0, 0<1; 6-1=5

↓

small=0;

j=1; 1<6

a[1]<a[0]

6<9

small=1

j=2.

a[j]<a[small]

a[2]<a[1]

7<6 (W)

j=3, 3<6

a[i]<a[small]

a[3]<a[1]

2<6

small=3

Scanned by CamScanner

j=4, 4<6
a[j] < a[small]
a[4] < a[3]
1 < 2
small = 4

j=5, 5<6
a[j] < a[small]
a[5] < a[4]
3 < 1 (W)

0	1	2	3	4	5
1	6	7	2	9	3

i=1, 1<5
↓
small = 1
j=2; 2<6
a[j] < a[small]
a[2] < a[1]
7 < 6 (W)

j=3, 3<6
a[3] < a[small]
a[3] < a[1]
2 < 6
small = 3.

j=4, 4<6
a[4] < a[small]
9 < 2 (W)

j=5, 5<6
a[5] < a[3]
3 < 2 (W)

0	1	2	3	4	5
9	2	7	6	9	3

i=2; 2<5
↓
small = 2.
j=3; 3<6
a[j] < a[small]
a[3] < a[2]
6 < 7
small = 3.

j=4, 4<6
a[j] < a[small]
a[4] < a[3]
9 < 6 (W)

j=5, 5<6
a[j] < a[small]
a[5] < a[3]
3 < 6 (T)
small = 5.

1	2	3	6	9	7

i = 3
Small = 3

j = 3+1
3 < 4
a[4] < a[3]
6 < 3 (W)

j = 5
a[5] < a[3]
9 < 3 (W)

i < n-1
i < 5 5 < 5

i = 4
Small = 4

j = 5
a[j] < a[small]
a[5] < a[4]
7 < 9 (T)
Small = 5.

i = 5 5 < 5 (F)

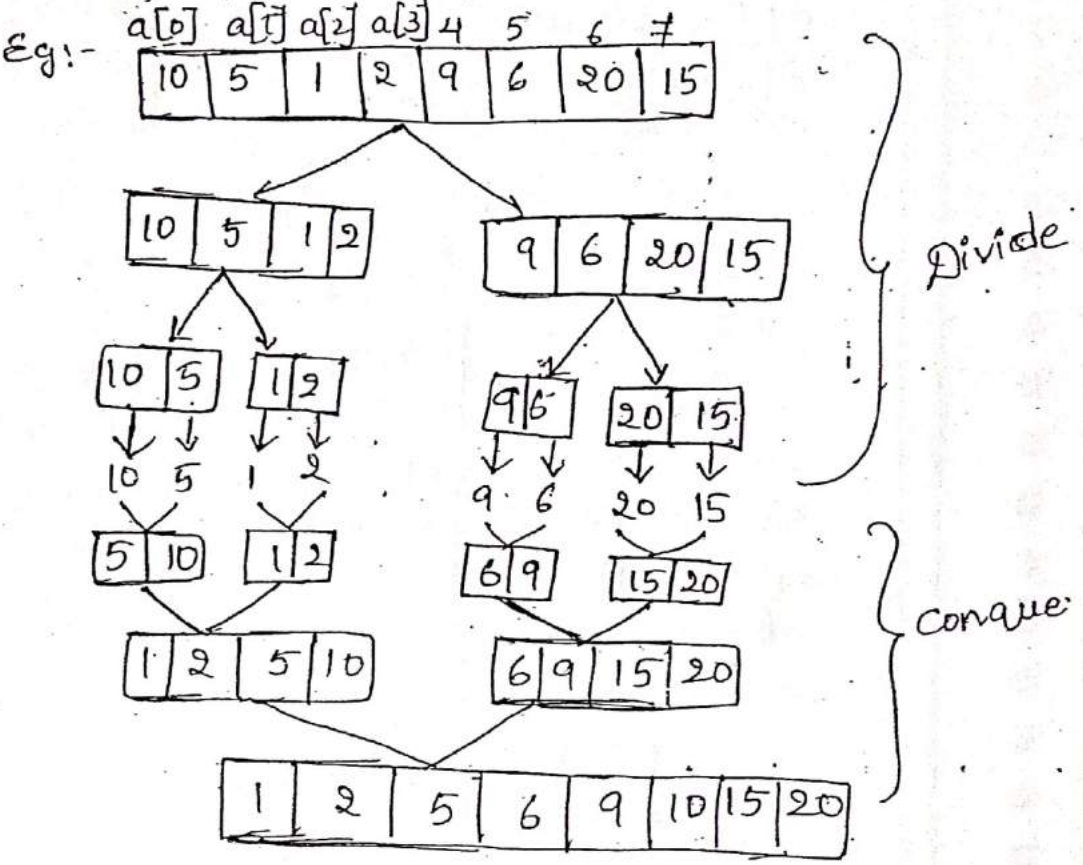
1	2	3	6	7	9

Scanned by CamScanner

Merge sort:-

It is a sorting algorithm that follows Conquer rule the given array is divided into arrays. The two sub arrays are further divid 4 sub arrays. So this process continues. Until i array elements after division the individua are sorted into several sub arrays all sub arr combined and merged to form a single sorted

Eg:-



Program:-

```
#include<stdio.h>
main()
{ int n,i, a[20];
  printf("enter n value");
  scanf("%d",&n);
  printf("enter array elements");
  for(i=0;i<n;i++)
```

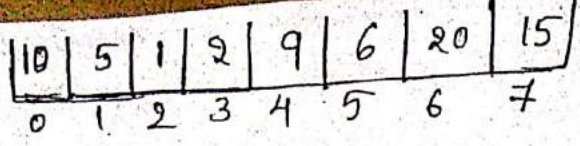
```
void mergesort(int [],int,in
void merge (int[],int,int,in
mergesort(a,0,n-1);
printf("elemer
```


Scanned by CamScanner

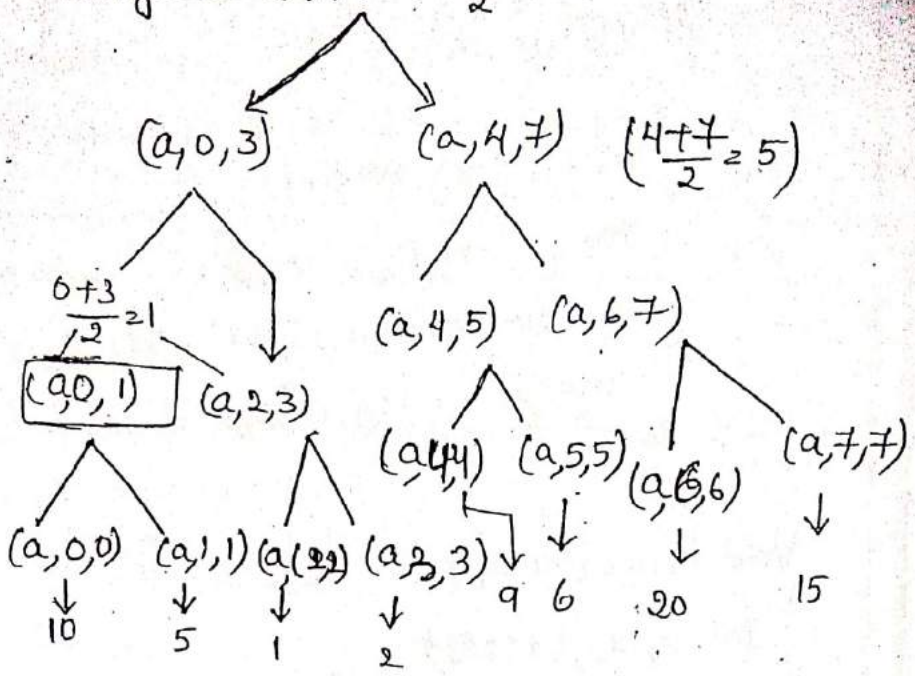
```
Void merge sort (int a[20], int beg, int end)
{
  int mid;
  if (beg < end)
  {
    mid = (beg + end) / 2;
    merge sort (a, 0, mid);
    merge sort (a, mid+1, end);
    merge (a, beg, mid, end);
  }
}
Void merge (int a[20], int beg, int mid, int end)
{ int l1, l2, i, t[20], k;
  l1 = beg, l2 = mid+1, i = beg.
  while (l1 <= mid && l2 <= end)
  { if (a[l1] <= a[l2])
      t[i] = a[l1++];
    else
      t[i] = a[l2++];
    i++;
  }
  while (l1 <= mid)
  { t[i++] = a[l1++];
  }
  while (l2 <= end)
  { t[i++] = a[l2++];
  }
  for (k = beg; k <= end; k++)
  a[k] = t[k];
}
```

Complexity = n log n

Scanned by CamScanner



merge sort (a,0,7) $\Rightarrow \frac{0+7}{2} = 3$.



Quick Sort :-

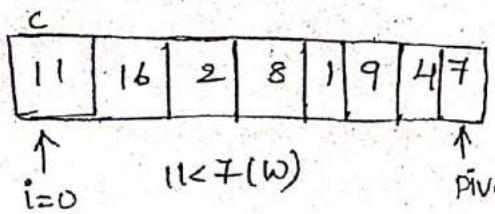
This algorithm follows divide and conquer tec which is here called as partitioning. It partition array into smaller arrays based on pivot eleme

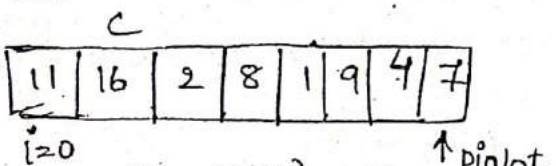
procedure :-

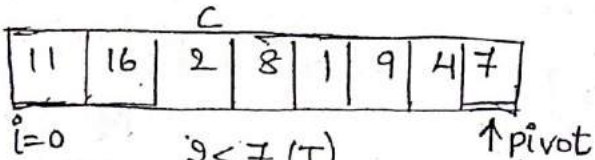
1) Select an element called pivot in the array el it can be last element (or) first element.
2) Rearrange the elements in such a way tha elements before pivot are lesser than pivot elements after pivot are greater than pivot. Such partitioning the pivot is placed in its fi position this is called partition operation.
3) Recursively sort the two sub arrays the conc for checking pivot is if current element is le than pivot then swap index element and cu element and increament the index position by
4) The current element is always incremented

Scanned by CamScanner

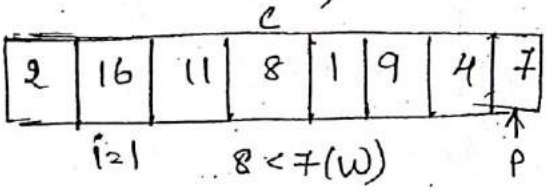
irrespective of condition current element less than pivot is true (or) false.

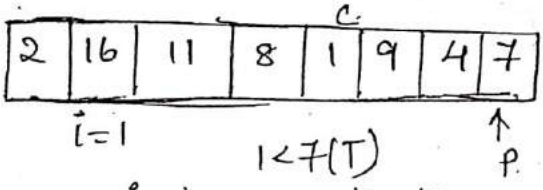




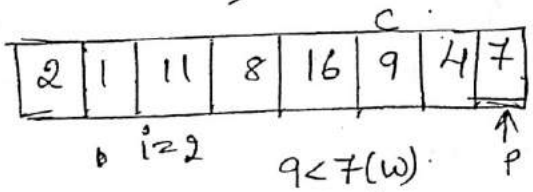


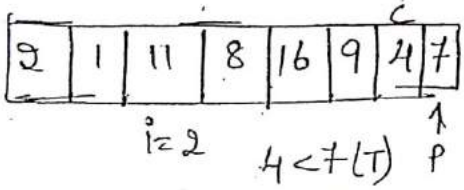
index ≠ current swap
index++, current ++

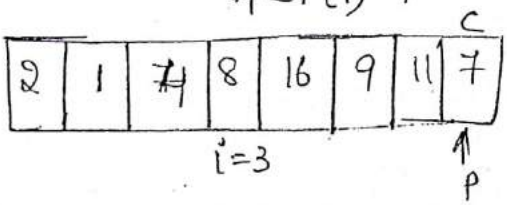




index = current
index++, current++

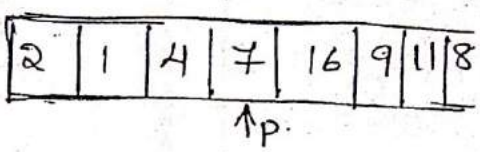




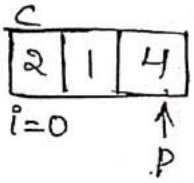


Here current ≠ pivot element are same so we have to check for another condition if index ≠ pivot then ...

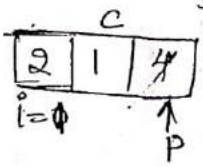
```
if current < pivot
{ swap index = current
  index++
}
current++
```



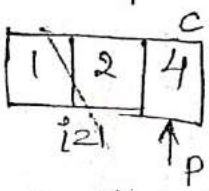
elements before the 'p' are less than pivot & after 'p' are greater than pivot. so divide the array into two sub arrays. the elements before 'p' as one array and another elements after 'p' as one array.



2<4(T)
if c & i are same there is no need to swap.

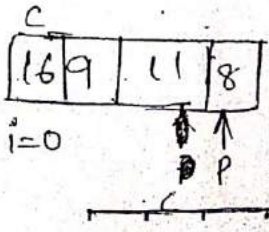


1<4(T)



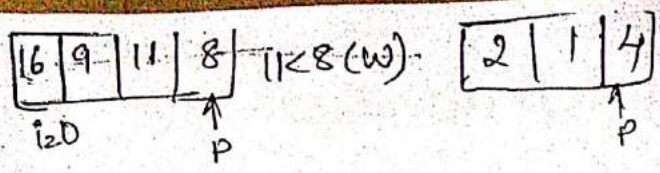
i>p
2>1(W)

So there is no need to swap.

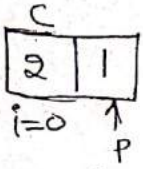


16>9(W)

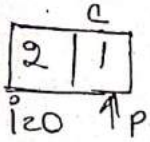
Scanned by CamScanner



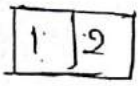
If c, i & p are applied on a same variab
pivot value can be neglected.

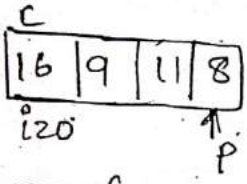


2<1 (W)

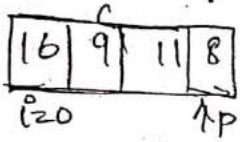


if i>P
2>1

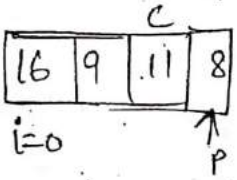




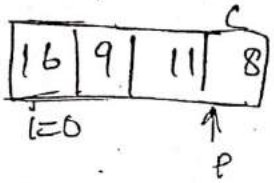
(16<8) (W)



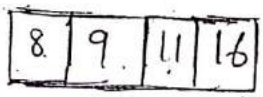
9<8 (W)

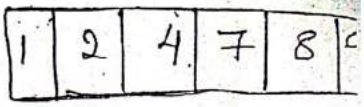


11<8 (W)



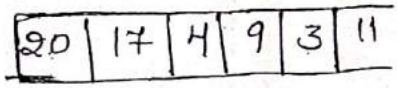
i>P
16>8





If pivot interchang
is not considered.

Eg:-



Algorithm:-

Step-1: start

Step-2 :- Input ← n, a[20]

Step-3 :- Intilize low=0, high=n-1.

Step-4 :- If (low< high)

{

call partition (a, low, high)

call quick sort (

Scanned by CamScanner

Call quicksort (a, p+1, high).

}

Algorithm for partition:

1) Intilize i=0, ~~p=p+1~~, j=0, p=a[high]

2) Repeat step-3 until j<=high.

3) if (a[j]<a[high]) (or) if (a[j]<=p)

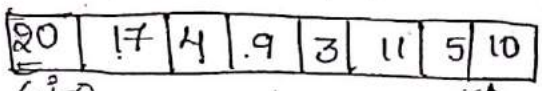
(i) then swap a[i] and a[j]

(ii) Increament i by 1.

4) Swap a[i] & pivot.

5) Return ~~pivot~~ 'Pivot.

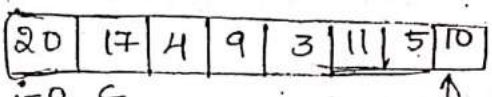
6) Stop.



if current < pivot

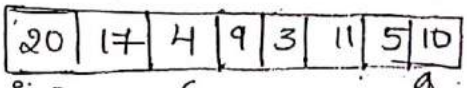
20<10 (W)

current++



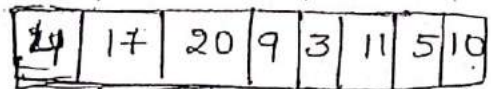
if c<p

17<10 (W)

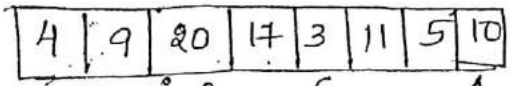


if c<p

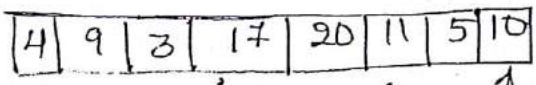
swap i&c 4<10 (T), i++, c++



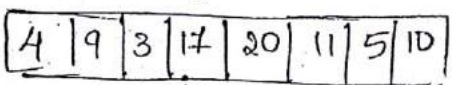
if (9<10) (T)



if c<p ⇒ 3<10 (T)



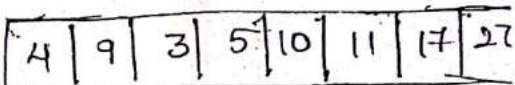
11<10 (W)



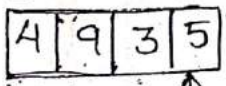
5<10 (T) i=4 c

if index>pivot

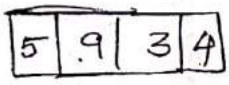
20>10 (T)



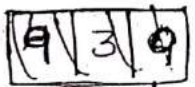
Array-II

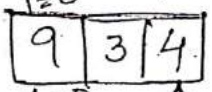


if c<p ⇒ 4<5 (T)

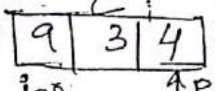


Here pivot changed its posi so it is placed in its place.

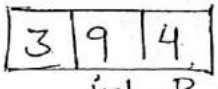




9<4 (W)

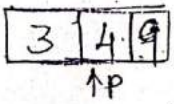


3<4 (T)

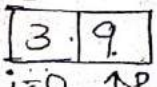


if (i>p)

9>4 (T)



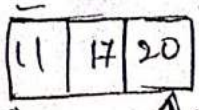
pivot changed its position so it is removed & placed in its position.



3<9 (T)

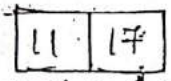
Here c, p are applie

Scanned by CamScanner

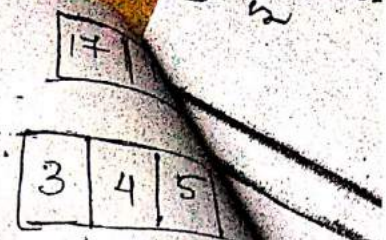


if 11<20(T)

Here pivot changed its position



11<17(T)



program:-

```
#include<stdio.h>
Void main()
{
  int a[10], i,n
  printf("enter n value");
  scanf("%d",&n);
  printf("enter array elements");
  scanf("%d", &a[i]);
  quick sort(a,0,n-1);
  printf("elements after sorting");
  for(i=0; i<n; i++)
}
```

Void quicksort (int[], int, int
int partition (int[], int, in

```
Void partition sort(a, low, high)
{ int i=low, j=low, p=a[high];
  for(i=0; i<n; i++)
  {for(j=0; j<=high; j++)
   {if (a[j]<p)
     {
      a[j]=t;
      a[i]=a[j];
      t=a[i];
     }
```

```
  printf("%d", a[i]);
  getche();
}
```


Scanned by CamScanner

```
Void quick sort (int a[20], int low, int high)
{
  if (low < high)
  { int piz
    call partition (a, low, high);
    call quick sort (a, low, pi-1);
    call quick sort (a, pi+1, high);
  }
}
int partition (int a[20], int low, int high)
{
  int pivot = a[high];
  int i = low, c, t;
  for (c = low, c <= high-1; c++)
  {
    if (a[c] <= pivot)
    {
      t = a[i];
      a[i] = a[c];
      a[c] = t;
      i++;
    }
  }
  t = a[i];
  a[i] = a[high];
  a[high] = t;
  return i;
}
```


Scanned by CamScanner

Radix sort:-

It is a sorting that can be used to so numbers by its base. To sort the array numbers it needs 10 temparory locations ca Because the base numbers for decimals is 10. If sort english words we need 26 buckets because alphabets are 26. This algorithm is also called as sort.

Procedure:-

1) Identify the biggest number in the array and of digits.

2) place 0s before every number in the array lik largest number in the array.

3) If no. of digits are '3' then their will be 3 pl sorting. If no. of digits are 'n' then their will k of sorting.

Phase-1:- Take buckets from 0 to 9 and sear the units place for the number according to store the numbers in a temporary array. sto 0th bucket to 9th bucket.

Phase-2:- Search for 10th place and place accordi numbers in the buckets. Now take the results and perform phase-3.

phase-3:- Search for 100's place and place the according to buckets. If no. of digits are 3 in number then the array will be sorted comp this phase otherwise places will be increase place and so on.

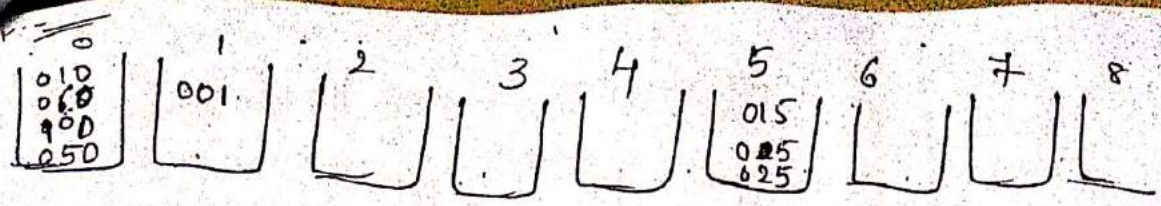
Eg:- 10 15 1 60 5 100 25 50

first select largest element = 100

No. of digits in the largest element = 3.

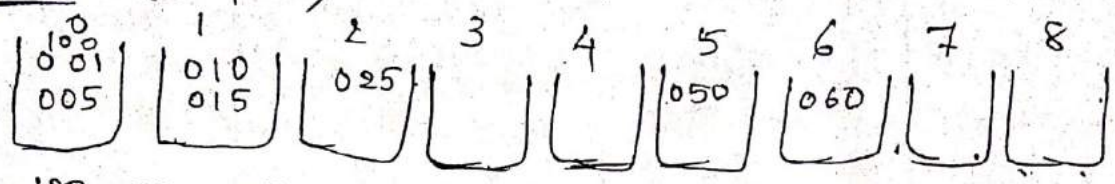
010 015 001 060 005 100 025 050

Scanned by CamScanner



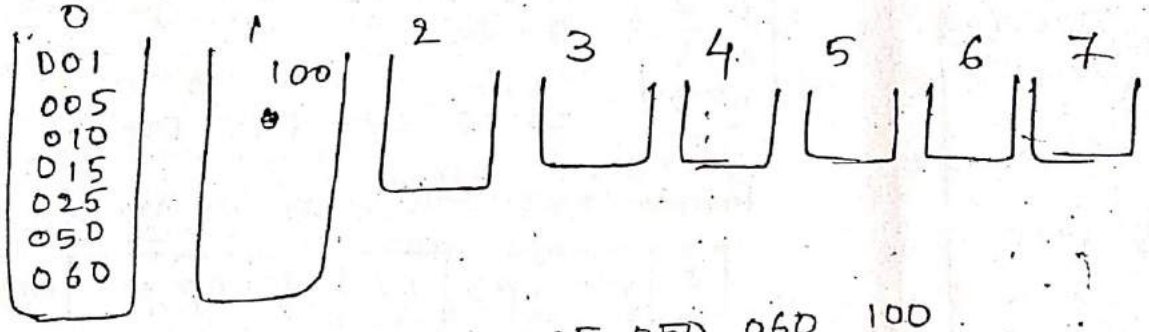
010 060 100 050 001 015 005 025

phase-2:- (10th place)



100 001 005 010 015 025 050 060

Phase-3:-(100th place)



001 005 010 015 025 050 060 100

→ Remove zeros from the above array

1	5	10	15	25	50	60	100

Implement Radix sort in the given list of numbers

170, 45, 75, 90, 802, 24, 2, 66.

1) first select the largest element in the given array.

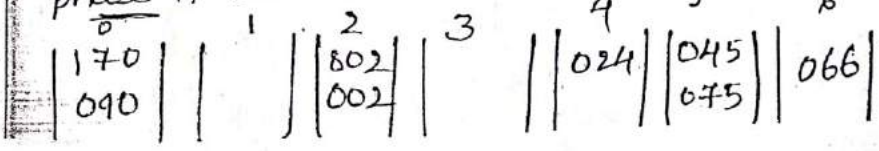
largest element = 802.

2) Next count no. of digits in the largest element. And we have to place 0s before every number in the array like the largest number in the array.

170 045 075 090 802 024 002 066

3) After placing 0's in the array phases will start. The no. of phases in the array depends on the no. of digits of the largest element.

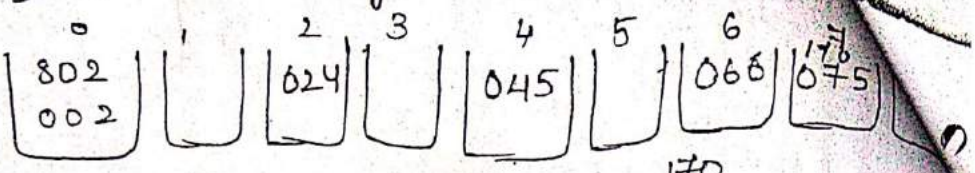
phase-1:- search for units place.



Scanned by CamScanner

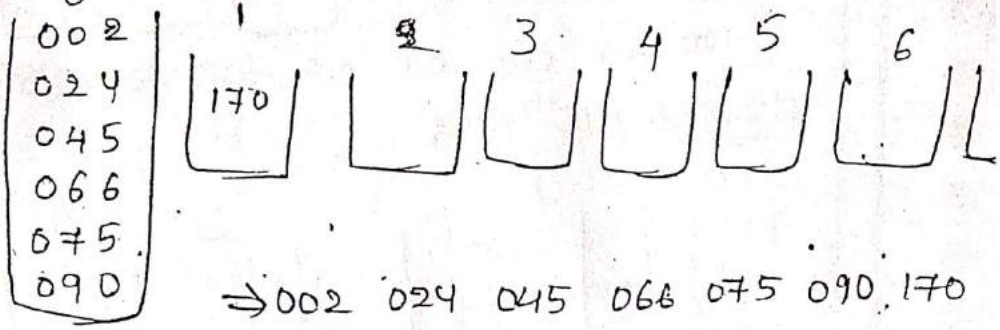
→ 170 090 802 002 024 045 0

Phase-2:- Searching for 10's place.



802 002 024 045 066 075 090 170

Phase-3:- Searching for 100's place.



⇒ 002 024 045 066 075 090 170

Remove zero's added in the above array

2	24	45	66	75	90	170	802

Algorithm:-

Step-1:- Start

Step-2:- ~~Read~~ Input n no. of elements in an

Step-3:- Accept array elements

Step-4:- find the largest element in the arra

Step-5:- pat each element with no. of zero's that

Step-6:- Initilize bucket j=0 and repeat the ne until j<n.

Step-7:- Compare ith position of each element array and place it in corresponding buck

7.1 sort the number by Units value

7.2 sort the number by 10's value.

7.3 sort the number by 100's value.

Step-8:- Read the elements of bucket

Scanned by CamScanner

step-9:- Display the sorted array by removing zero before every element.

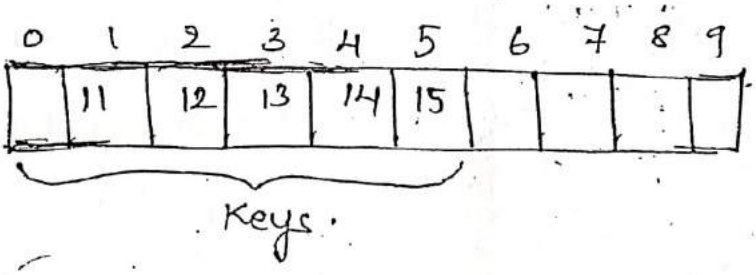
Hashing

Hashing is an important data structure technic that is used to uniqly identify a specific object from a group of ~~sing~~ similar objects. This hashing ~~technique~~ technic arranges data in key value pairs ~~based~~ where key are unique elements. this keys are used as integers in an array and values are used as elements in the array. this hashing is performed in a data structure called "Hash Table" which stores data in an associative manner(key value pairs). In hashing large keys are converted to small keys by using Hash functions. Hash function is defined as H(x) = [x%. Array size]

Eg:- Let us assume an array list containing 10 element as list of 10 11 12 13 14 15 16 17 18 19 20

we can generate keys by using function as

H(11) = 11%.10 ⇒ 1
H(12) = 12%.10 ⇒ 2
H(13) = 13%.10 ⇒ 3
H(14) = 14%.10 ⇒ 4
H(15) = 15%.10 ⇒ 5



Collision in hashing:-

when two (or) more distinct key elements are maping to same hash value of index then it is called as collision

To prevent collisions we are having some collision resolution techniques. Among them (i) linear probing.

linear probing:-

In this methodology it moves to the next location

Scanned by CamScanner

free slot.

The function for calculating index v

probing $(H(x,i) = (H(x)+i) \bmod n$.

where $H(x)$ is the keyelement.

n is the size of the array

Develop a c program to create a hash tab perform operations insert, search using linear technique

```
#define n
int a[n];
void insert()
  {
   int key,hash,i,index,n;
printf("enter the key");
scanf("%d",&key);
   hash = key %.n;
 for(i=0;i<n;i++)
{
index=(hash+i)%.n;
 if(a[index]==0)
   {a[index]=key;
    break;
 } }
 if(i==n)
 printf("element is not inserted");
}
void search()
  { int key,hash,i,index,n;
    printf("enter the
```


Scanned by CamScanner

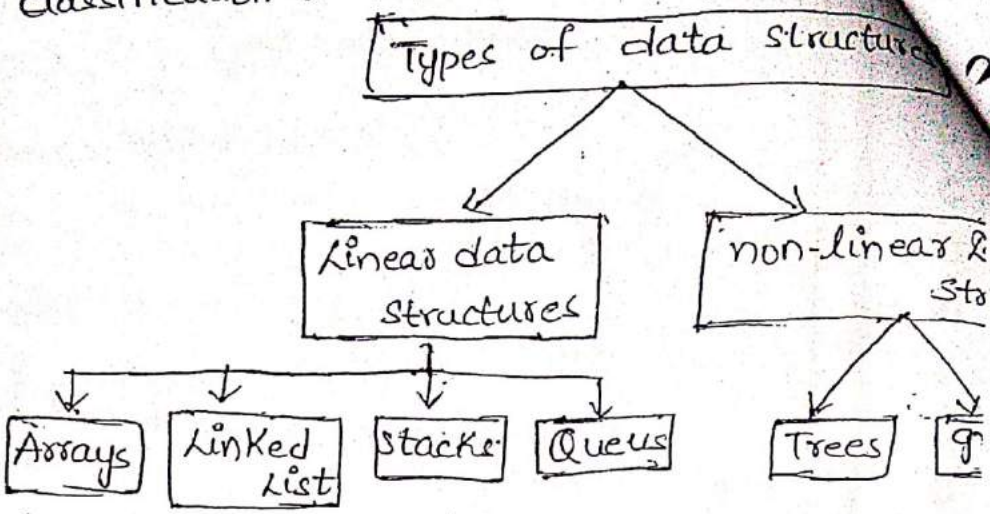
```
for(i=0;i<n;i++)
{ index = (hash+i)%n;
  if (a[index]==key)
  {printf("element found");
   break;
  }
}
if (i==n)
printf("element not found");
}
Void disp()
{
  int i;
for(i=0;i<n;i++)
  printf("%d",a[i]);
}
Void main()
{ int option;
  while(1)
    {
    printf("press 1.Insert \n 2.Display \n 3 disp \n search
                                                    4.exit)

    scanf("%d",&option);
    switch(option)
    {
     case 1: Insert();
             break;
     case 2: Disp();
             break;
     case 3 : search();
              break;
     case 4: exit(0);
    }
```


Scanned by CamScanner

8/7/19

Unit - II

Classification of Data structures.



linear data	Non linear dat
1) In linear data structures elements are accessed in sequential order.	1) Elements are a through sorted c there will be ge between each and element.
2) these are simple to Implement	2) these are com Implement
3) These are operated at single level.	3) These are oper Multi level.
4) These data structures memory utilization is ineffective.	4) Here memory is effective.

Linked List:- A linked list is a data struct containing group of elements con nodes whic connected with eachother. Each node contains

(i) 1st part contains data(values)

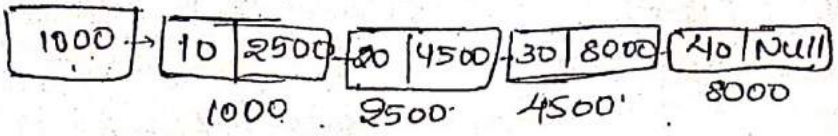
(ii) 2nd part contains link, which holds the of next node

The ...

Scanned by CamScanner

The link part of node last node is null.

Eg:- root N_1 N_2 N_3 N_4



There are different types of linked list.

(i) single linked list

(ii) Double Linked List

(iii) circular Linked List

(i) single linked list

It contains only one link in each node.

(ii) Double linked list

It contains two links previous link pointing to previous node and next link pointing to next node.

(iii) Circular linked list:

It contains same as the node part Double linked list but is operates in circular manner.

```
Struct node
    {
      int data;
      struct node * Link;
    }
```

Link is a pointer that points to address of another node. so the link pointer is of type struct node.

Operation of single linked list

Insertion:- we can Insert an element in the begining of the linked list (or) we can insert an element at the end of the list (or) we can also insert at a specified place in linked list

Deleting:-

we can delete an element at the begining of the list (or) ending of the list (or) we can also delete a specified place.

Reversing:-

We can

Scanned by CamScanner

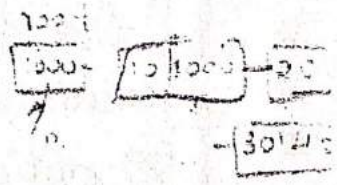
Count:-
we can count no. of elements (or) no

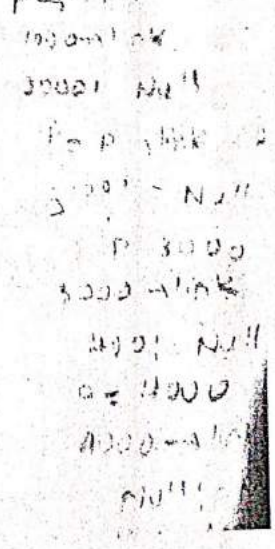
list.

Search:-
we can also search for an element in lis

write a C program to perform operations on linked list.

```
struct node
  {
   int data;
   struct node * link;
  }
  struct node * root=Null;
 Void main()insert ()
  {
    struct node *temp;
      temp = (struct node *) malloc (sizeof (stru
      printf ("enter data");
      scanf (" %d", &temp -> data);
      temp -> link = Null;
       if (root==Null)
       {
          root = temp;
       }
      else
       {
        struct node *p;
         p = root;
         while (p->link! =Null)
         {
           p=p->link;
         }
          p -> link = temp;
       }
```

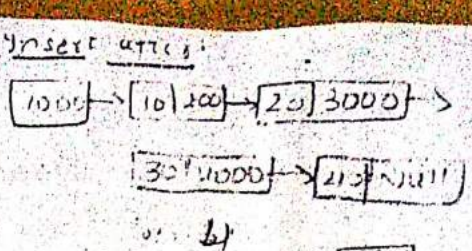




Scanned by CamScanner

```
{ int count=0;
struct node *P;
  P=root;
  while (p->link != Null)
    { count++;
      P= P->link;
    }
   return count;
}
void insert after ()
{ struct node * temp,*P;
int loc, length, i=1, node;
printf("enter location");
scanf("%d", &location);
 len = counting ();
 if (len< loc)
 {
   printf("Invalid loc");
 }
 else
 { p=root;
  while (i<loc)
   {
     p=p->link;
     i++;
   }
  temp= (struct node *) malloc (sizeof(insert node));
  printf("enter data");
  scanf("%d", &temp->data);
  temp->link = p->link;
    p->link =temp;
 }
```

insert after:



Node temp 50 5000

if (5<4) (W)

P=1000

while (1<4)

1000=1000 → 2000

i=2

if (2<4)

2000=2000 → 300

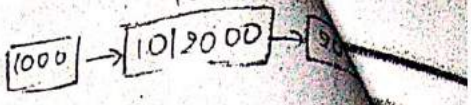
if (3<4)

3000=3000 → 4xxx

5000=4000 → 5c

Scanned by CamScanner

```
Void display ()
{
  struct node *P;
  P = root;
  while (P != NULL)
  {
  printf("%d", P->data);
    P = P->link
  }
}
```



P: root = 1000
1000 != Null (T)
P -> data => 1000
1000 -> 10
P = 1000
P = 2000 != Null
2000 -> data => 200
P = 2000 -> link
P = 3000 => 3000
3000 -> data =>
P = 3000 -> link
P = 4000
4000 -> data =>

Deletion of a node:-

point the node as temp.
Remove the left connection of the node by filling the previous nodes link part with current nodes link part. Remove the right connection by placing null in link part of the desired deleted.
Then the node is ready to delete we can release the memory of the desired node by using function free.

syntax :- free(ptr);

```
Void delete()
{
  struct node *P;
  int loc, len, j=1;
  printf("enter location");
  scanf("%d", &loc);
  len = counting();
  if (len < loc)
  printf("Invalid location");
  else if (loc == 1)
  {
    P = root;
```


Scanned by CamScanner

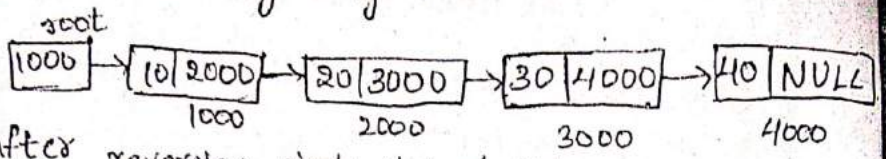
```
    free(p);
  }
  else
  {
    struct node *p,*q;
      p=root;
    while (i<loc-1)
      {
        p=p->link;
          i++;
      }
      q=p->link
      p->link=q->link;
       q->link=Null;
       free(q);
  }
}
void main()
{
    int option,x;
    while(1)
      {
       printf("\n1.insert \n2.counting\n3.Insertafter \n4. display
                                        \n5.delete \n6.exit \n");

       scanf("%d",&option);
       switch(option)
       {
          case 1: insert();
                  break;
          case 2:x=counting();
                  break;          printf("No of nodes are %d", x);
          case 3; Insertafter();
                  break;
          case 4; display();
                  break;
          case 5; delete();
                  break;
          case 6:. exit(0);
       }
    }
```


Scanned by CamScanner

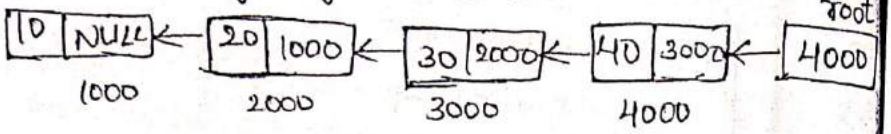
Reversing a single linked list

In reversed linked list the root will
list and every node points to the previous
node holds the address of the previous node
part. The first node contains null in the link part

Before reversing single linked list



After reversing single linked list



Algorithm :-

Step-1 :- Start

Step-2 :- Declare 3 pointers *current, *next, *prev

Step-3 :- Initilize current = root, prev = NULL.

Step-4 :- Repeat step 5, 6, 7, 8 until current != NULL

Step-5 :- set Next = current → link

Step-6 :- set current → link = pre

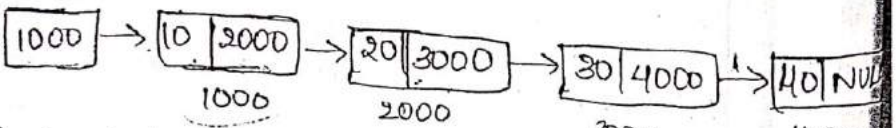
Step-7 :- prev = current

Step-8 :- current = next

Step-9 :- set prev = root

Step-10 :- stop.

root

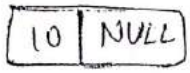


(1) current = 1000, prev = NULL

current != NULL

1000 != NULL (T)

Next = current → link



Next = 2000.

4000 → link = pre.

1000 → link = NULL.

prev = 1000

Scanned by CamScanner

② current = 2000, prev = NULL

2000 != NULL

Next = current → link.

Next = 3000.

2000 → link = 1000

prev = 1000

Next = 3000 = current

20	1000

③ current = 3000, prev = NULL

3000 != NULL

Next = current → link

Next = 4000

prev = 4000

30	2000

Current = 4000

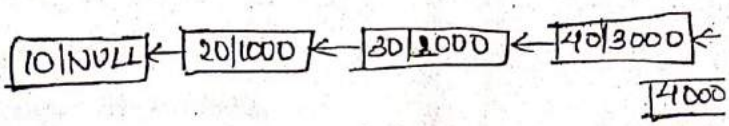
prev = 3000

Next = 4000 → NULL

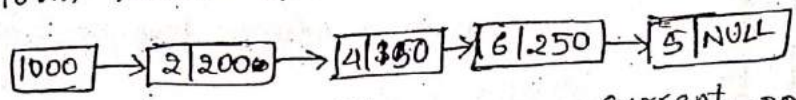
4000 → NULL = 3000

prev = 3000

40 | 3000



Perform reverse operation for linked list



Current = 100, prev = NULL

current != NULL

100 != NULL(T)

Next = current → link

Next = 200

100 → link = Null

prev = 100

current = 200

2 | NULL

Current = 150, prev = NULL

150 != NULL

Next = 250

250 → link = 150.

prev = 150

Current = 250

6 | 150

current = 200, prev = NULL

200 != NULL(T)

Next = 300

300 → link = 200

prev = 200

current = 300

4 | 200

current = 250, prev = NULL

250 != NULL

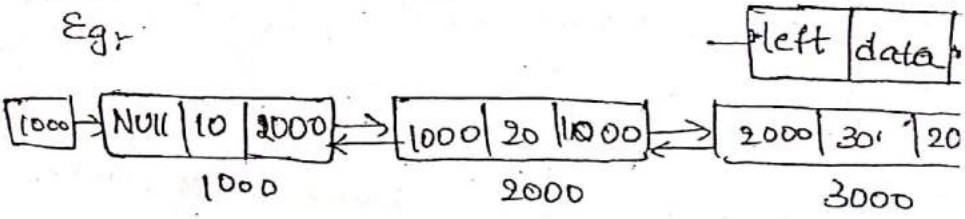
Next = 9

Scanned by CamScanner

Double linked list

It is a two way linked list which to the previous node as well as a poin node, in the sequence. In single linked list the list in forward direction only. whereas list we can move in forward and backwar creation of a node in double linked list contains 3 parts. middle part contains data, ains pointer to the previous node and right p pointer to next node.

Eg:



Program to perform the operations on Dou

```
Struct node
  {
    int data;
    Struct node * left;
    Struct node * right;
  }
  Struct node *root=NULL;
  Void insert()
  {
    Struct node *temp;
    temp=(struct node*)malloc(sizeof(str
    printf("enter data");
    scanf("%d",&temp->data);
    temp->right = NULL;
    temp->left = NULL;
    if (root == NULL)
      {
        root=temp;
      }
```


Scanned by CamScanner

```
    struct node *p;
      P=root;
    while( P->right != NULL)
      {
        P = P->right;
      }
      P->right= temp.
      temp->left =P;
  }
}
int counting c)
 {
   int count=0;
   struct node *P;
    P=root
    while( P-> != NULL)
    {
      count++;
      P=P->right;
    }
    return count;
}
Void display c)
{
  struct node *P;
    P= root;
  while(P!= NULL)
   {printf("%d ", P->data);
     P = P->right;
   }
 }
Void insert after( )
{
  struct node *P, *temp;
   int loc, len, i=1, node;
   printf("enter location");
```


Scanned by CamScanner

```
len=counting();
if(len< loc)
 {
   printf("invalid location");
 }
else
 {
   p=root;
   while(i<loc)
   {
     p=p->right;
     i++;
   }
   temp=(struct node *) malloc(size of (insert-n
   printf("enter data");
   scanf("%d",&temp->data);
   temp->right = p-> right;
   p->right->left = temp;
   p->right =temp;
   temp-> left =p;
 }
}
Void delete()
{
   struct node *p;
   int loc, len, i=1;
   printf("enter loc");
   scanf("%d",&loc);
   len= counting();
   if(len< loc)
   printf("Invalid location");
}
else if (loc==1)
{
   p= root;
   root= p->right
```

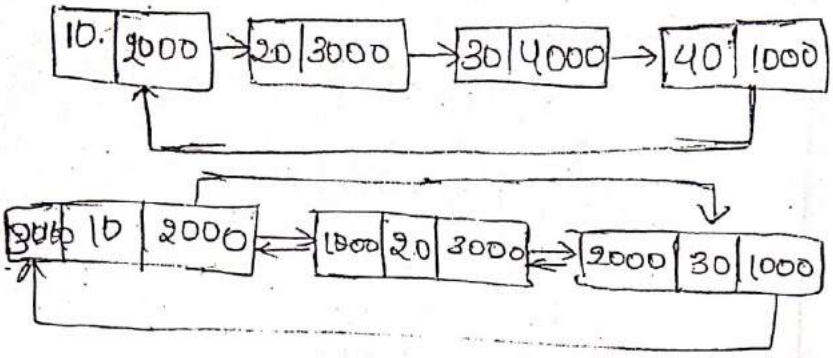

Scanned by CamScanner

```
p->right = NULL;
p->left = NULL;
free(p);
}
else
{
struct node *p, *q;
p=root;
while (i<loc-1)
{
p=p->right;
i++;
}
q=p->right;
p->right = q->right;
q->right = NULL;
p->left = q->left;
q->left = NULL;
free(q);
}
}
void main()
{
int x, option;
while(1)
{switch(option)
{
case 1: insert();
break;
case 2: d = counting;
printf("No of nodes are '/.d", x);
break;
case 3: insertafter();
break;
case 4: display();
break;
case 5: exit(0)
}
}
```


Scanned by CamScanner

.rcular Linked List:-

It is a linked list where all the node to form a circle. There is no NULL value of the list. A circular linked list can be a (or) double linked list.



Algorithm:-

The operations on circular single linked l

1) Insertion of a node at begining
2) Insertion of a node at ending.
3) Deletion of a node at begining
4) Deletion of a node at ending

Procedure:- Allogate memory for new node initilize its data

2) Take a pointer variable p which will poir

3) move p so that it points to the last no list. add the new node after the node

Note:- If availability space is NULL then u insert a node in the list.

Step-1:- If avail = NULL (No available memory then print error: over flow.

step-2:- goto step 10.

Allocate memory for temp

Step-3:- Initilize temp → data

Step-4:- set p=root.

Scanned by CamScanner

step-6:- p=p→link.

step-7:- set p→link = temp

step-8: Set temp→link = root

step-9 set root = temp.

step 10:- stop.

Algorithim to insert node at end.

step-1:- start

step-2:- If avail = NULL (no available memory space)
then print error: over flow.

step 3:- Allocate memory for temp.

step-4:- Initilize temp→data.

step 5:- set p=root

step-6:- Repeat step 7 until p→link != root.

step-7:- p= p→link

step-8:- p→link = temp.

step-9:- temp→ link= root

step-10: Stop

Algorithm to delete node at begining.

step-1:- start

step-2:- if root == NULL
print "Underflow"
goto step 10.

step-3:- stp = root

step-4:- Repeat step 5 until p→link != root.

step-5:- set p=p→link

step-6:- Set p→link = root → link

step-7:- root → link = NULL.

step-8:- free(root)

step-9:- root = p→link

step-10:- stop.

Algorithm to delete node at ending.

Step-1:- start.

Scanned by CamScanner

Step 2:- if root == NULL
print underflow
goto step 10

Step 3:- Set p = root

Step 4:- Repeat step 5,6 until p->link != root

Step-6:- p = p->link.

Step-5:- set preptr = p.

Step-7:- preptr->link = root

Step-8:- p->link = NULL

Step-9:- free(p)

Step-10:- stop.

Applications of linked list

1) Implementation of stacks and Queues.

2) Implementation of graphs. (adjacency represe graphs is most popular which uses linked lis adjacent vertices)

3) Dynamic memory allogation.

4) Maintaing directory of names.

5) performing arthimetic operations on long i

6) Manipulation of polynomials by storing cons the node of linked list.

7) Representing sparse matrix.

Applications in real world

1) image viewer. (previous & next images are linke be accessed by using previous & next b

2) Music player.

3) previous & next pages in web browser.

4) circular linked list is used in task men application of operating system.

Scanned by CamScanner

Consider the Polynomials $x_1 = 7x^4 + 5x^2 + 3x$, $x_2 = 5x^3 + 3x - 8$

x_1:

	0	1	2	3
Coeff	7	5	3	
Exp	4	2	1	

x_2:

	0	1	2	3
Coeff	5	3	-8	
Exp	3	1	0	

Develop a C Program to perform addition of 2 polynomial using array.

```
Struct Poly
{
    int Coeff, exp;
};
Struct Poly P1[10], P2[10], P3[10];
Void readPoly (Struct Poly P[], int *);
Int addPoly (Struct Poly P[], Struct Poly P[], int, int);
Void display (Struct Poly P[], int);
Void main()
{
    int t1, t2, t3;
    Printf("\nEnter no of terms in Polynomial 1");
    Scanf ("%d", &t1);
    readPoly (P1, t1);
    Printf ("\n Enter no of terms in polynomial 2");
    Scanf ("%d", &t2);
    readPoly (P2, t2);
    Printf ("\n Polynomial 1 representation is");
    display (P1, t1);
    Printf ("\n Polynomial 2 representation is");
    display (P2, t2);
    P3 = addPoly (P1, P2, t1, t2);
    Printf ("\n addition of 2 polynomials is");
} display (P3, t3);
```

```
Void readpoly (struct poly P[10], int t)
{
    int i;
    for (i=0; i<t; i++)
    {
        Printf ("\n Enter Coefficient %d", i+1);
        Scanf ("%d", &P[i].Coeff);
        Printf ("\n Enter Exponent %d", i+1);
    } Scanf ("%d", &P[i].Expn);
}
int addpoly (struct poly P1[10], struct poly P2[10], int t1, int
{
    int i=0, j=0, k=0;
    while (i<t1 && j<t2)
    {
        if (P1[i].Expn > P2[j].expn)
        {
            P3[k].Coeff = P1[i].Coeff;
            P3[k].Expn = P1[i].Expn;
            i++; k++;
        }
        else if (P1[i].Expn < P2[j].Expn)
        {
            P3[k].Coeff = P2[j].Coeff;
            P3[k].Expn = P2[j].Expn;
            j++; k++;
        }
        else
        {
            P3[k].Coeff = P1[i].Coeff + P2[j].Coeff;
            P3[k].Expn = P1[i].Expn + P2[j].Expn;
            i++; j++; k++;
        }
    }
}
```

```
while (i<t1)
{
    P3[k].Coeff = P1[i].Coeff;
    P3[k].Expr = P1[i].Expr;
    i++; k++;
}
while (j < t2)
{
    P3[k].Coeff = P2[j].Coeff;
    P3[k].Expr = P2[j].Expr;
    j++; k++;
}
return k;
}
void display (struct poly P[], int t)
{
    int i;
    for (i=0; i<t; i++)
    {
        Printf ("%dx^%d", P[i].Coeff, P[i].Expr);
        if (i<t-1)
            PrintF ("+");
    }
}
```

Sparse Matrix :- A sparse matrix is a matrix which contains more no of 0's & very few non-zero elements

Eg :-

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	0	2	0	0	0

When a sparse matrix is represented in a 2D array we waste a lot of space to represent that matrix because the matrix will contain majority of 0's

For Example :- If a matrix contains 5 rows & 6 columns total 30 elements then compiler allocate $30 \times 4 = 120$ Bytes But the actual no of non zero elements are only 6 that means the size of 24 Bytes is enough is enough for matrix & we are wasting 96 Bytes also the compiler will scan all 30 elements. To reduce this drawbacks we can use a Triplet Representation of Sparse matrix.

Triplet Representation :- In this we consider only non zero values along with their row & column index values. So the resultant matrix contains 3 columns in 0th row (i) Total no of rows (ii) Total no of columns (iii) Total no of non zero elements

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	0	2	0	0	0

5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

Transpose of Sparse Matrix: The first row of the triplet matrix the row value is changed to column value column value to row value. From 2nd row check the column value for index 0 & interchange row to column column to row value. Non zero value remains. like this repeat the column index checking upto end of matrix

Triplet

5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

Transpose

6	5	6
0	2	4
1	1	8
2	4	2
3	2	2
4	0	9
5	3	5

Write an algorithm to read the values in a matrix of order m×n & then print its triplet representation & also find transpose of matrix.

Step 1: Start

Step 2: Declare m, n, S[10][10], j, i, t, k, Element

Step 3: Read i=0, j=0, m, n, t=0

Step 4: Repeat Step 5 until i<m

Step 5: Repeat Step 6,7 until j<n

Step 6: Read Element

Step 7: If Element != 0

Step 7.1: Set t=t+1

Step 7.2: Set S[t][0]=i

Step 7.3: Set S[t][1]=j

Step 7.4: Set S[t][2]=Element

Step 8: S[0][0]=m

Step 9: S[0][1]=n

Step 10: S[0][2]=t

Step 11: Display S[0][0]

Step 12: ~~Step~~

* Stacks *

A Stack is an important data structure which stores its elements in an ordered manner. It uses a principle called "LIFO" that means the element that was inserted last was the first one to be taken out.

A Stack is used to store function calls.

Ex:- If there are

A Stack is a linear data structure in which elements are added or removed from Stack from one end called TOP.

Stack representation:-

A Stack is represented using i) Array ii) linked list

(i) Array:- Every Stack has a variable called top which is used to store the address of the top most element of Stack. There is another variable called MAX which is used to store the max number of elements the Stack can hold. If TOP = MAX - 1 then we can say that Stack is in overflow condition

Similarly if TOP == -1 then Stack will be in underflow condition.

Algorithm for push operation in Stack :-

Step 1 : Start
Step 2 : If TOP == MAX-1 then print "overflow" goto step 6
3 : Read val
4 : Set TOP = TOP+1
5 : Set Stack [TOP] = val
6 : Stop

Algorithm for pop operation in Stack :-

Step 1 : Start
2 : If TOP == -1 then print "underflow" Goto step 5
3 : Print Stack [TOP]
4 : Set TOP = TOP-1
5 : Stop

Algorithm for peak operation :

Step 1 : Start
2 : If TOP == -1 then print "underflow" go to step 4
3 : Print Stack [TOP]
4 : Stop.

```
#define max 10
int stack [max], top = -1;
void push(); void pop(); void peek(); void display();
void main()
{
    int val, opt;
    while(1)
    {
        Printf("\n Enter option \n1. Push \n2. Pop \n3. Peek
            \n4. display \n5. Exit ");
        Scanf("%d", &opt);
        switch(opt)
        {
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: peek();
                    break;
            case 4: display();
                    break;
            case 5: Exit(0);
        }
        default: Printf("overflow");
}
void push()
{
    int val;
    if (top == max-1)
        Printf("Over flow");
    else
    {
        Printf("Enter Element");
        Scanf("%d", &val);
        top = top+1;
```

```
    Stack [top] = val;
    Printf ("Element is inserted");
  }
}
void pop()
{
  if (top == -1)
    Printf ("Under Flow");
  else
  {
    Printf ("The element deleted is %d", Stack [top]);
    top = top - 1;
  }
}
void Peek()
{
  if [top == -1)
    Printf ("Under Flow");
  else
    Printf ("top most Element is %d", Stack [top]);
}
void display()
{
  int i;
  if (top == -1)
    Printf ("No Element found");
  else
  {
    for (i=top; i>=0; i--)
    {
      Printf ("%d", Stack [i]);
    }
  }
}
```

Stack representation using linked list:

Stack is represented as collection of nodes in linked list. Each node contains two components data & link. Data contains elements & link contains address to next node. The elements are added or removed at one end of linked list. i.e at the beginning of link list. The disadvantage of array representation of stack is the array contains fixed memory so we cant expand the memory during execution

If the list is Empty then "top" contains NULL.

Algorithm for Push operation:-

Step 1: Allocate memory for node & name it as top

Step 2: Read temp → data

Step 3: If top == NULL then

3.1: Set temp → link = NULL

3.2: Set top = temp

3.3: Else Set temp → link = top

3.4: Set top = temp

Step 4: STOP

Algorithm for pop operation:-

Step 1: If top == NULL the print "No elements in list"

Step 2.0: Else Set P = top

2.1: Set top = P → link

2.2: Set P → link = NULL

Step 2: free(p)

Step 3: Stop

Algorithm for Peek Operation:-

Step 1: If top == NULL print "No Elements in the list"

Step 2: Else Print top → data

Step 3: Stop.

Algorithm for traversing linked list:-

Step 1: If top == NULL print "No Elements in the list"

Step 2: Else

2.1: Set P = top

2.2: while P != NULL

2.3: Set P = P → link

2.4: Print P → data

Step 3: Stop

Applications of Stacks:-

* Stacks are used for Conversion of infix to Post fix Expression
* Stack is used to implement Towers of Hanoi

Types of Notations:-

(i) Infix: In this notation operator is placed in b/w operands Ex: A + B

(ii) Post fix: It is called as reverse Polish notation The idea of forming Post fix notation / Prefix notation is to develop a parenthesis free Expressions

In Post Fix Notation operator is placed after the operands

Ex:- $(A+B)*C = (AB+)*C = AB+C*$

$(A-B)*(C-D) = (AB-)*(CD-) = AB-CD-*$

(iii) Pre Fix:- It is evaluated from left to right like Post Fix but difference is operator is placed before operands

Ex:- $(A+B)*C = (+AB)*C = *+ABC$

$(A-B)*(C+D) = (-AB)*(+CD) = *-AB+CD$

Infix to Post Fix :- $A-(B/C+(D\%E*F)/G)*H$

The algorithm uses a Stack to temporarily hold operators & Post Fix Expression holds operands & operators that are popped from Stack. For Conversion

Step1: Add ")" at end of Expression

A-(B/C+(D%E*F)/G)*H)

Input Character to Stack	Stack	Post Fix Expression
	(	
A	(	A
-	(-	A
(	(-(	A
B	(-(	AB
/	(-(/	AB
C	(-(/	ABC
+	(-(+	ABC/
(	(-(+(	ABC/
D	(-(+(	ABC/D
%	(-(+(%	ABC/D

E	(-(+(%	ABC/DE
*	(-(+(%*	ABC/DE
F	(-(+(%*	ABC/DEF
)	(-(+	ABC/DEF*%
/	(-(+/	ABC/DEF*%
G	(-(+/	ABC/DEF*%G
)	(-(	ABC/DEF*%G/+
*	(-*	ABC/DEF*%G/+
H	(-*	ABC/DEF*%G/+H
)	Empty	ABC/DEF*%G/+H*-

/ * % . + -

∴ A-(B/C+(D%E*F)/G)*H = ABC/DEF*%G/+H*-

Algorithm :

Step 1: Add ")" at the End of infix Expression

Step 2: Push left Bracket "(" on to the Stack

Step 3: Repeat until Each Character in infix Char is Scanned

3.1 : If "(" is Encounted push it on to Stack.

3.2 : If an operand (whether a digit or alphabet) is encounted then add into post fix Expression

3.3 : If an round Bracket is Encounted

3.3.1 : Repeatedly pop from Stack & add it to post fix Expression until a ")" is encounted

3.3.2 : Discard "(" & ")" & don't add them to post fix Expression

3.4 : If an operator "O" is Encounted then repeatedly pop from Stack & add each operator to post fix Exp

Which has highest precedence than "("

Step4: Repeatedly pop from stack & add it to P.E. Sum until the stack is empty

H.P * / %

L.P + −

Step5: Stop.

Expression Evalution using Stacks:-

Convert the following Exp into Post fix & Evaluate it.

Expression 9-((3*4)+8)/4

9-((3*4)+8)/4

9-(34* +8)/4
A B

9-(34*8+ /4)
A B

9- 34*8+4/
A B

934*8+4/-

Input Character to Scan	Stack
9	9
3	9,3
4	9,3,4
*	9,18 (9,3*4)
8	9,12,8
+	9,20 (9,12+8)
4	9,20,4
/	9,5 (9,20/4)
-	4 (9-5)

Evaluate the Expression 2*(4+6)

2*(46+)

246+*

Input Character Scan	Stack.
2	2
4	2,4
6	2,4,6
+	2,10
*	20

Algorithm for Evalution of Post fix Expression:-

Step 1: If an operand is encountered push it on to Stack

Step 2: If an operator "O" is encountered then

2.1: Pop top 2 elements from Stack & name them as A & B A is top most element, B is element below A.

2.2: Evaluate BOA

2.3: Push the result into Stack

Step 3: Stop.

Queue:- A Queue is a FIFO data Structure containing two ends the left end is called front & right end is called rear. Insertion of element is done form rear & deletion of element done from front A Queue can be represented using (i) array (ii) linked list.

Queue Representation Using Array:

Algorithm For Insertion:

Step 1: If Rear = MAX-1 then
Print "Queue is Overflow"
Go to Step 5

Step 2: Else If rear == -1 and front == -1
then Set front = rear = 0

Step 3: else
Set rear = rear + 1

Step 3: Read val

Step 4: Set queue [rear] = val

Step 5: STOP.

Algorithm For deletion:

Step 1: If (front == -1 || front > rear) then
(q is empty) (All elements are deleted)
Print ("Queue is empty");
Go to Step 4.

Step 2: Print queue [front]

Step 3: Set front = front + 1.

Step 4: STOP

Algorithm For display Of Queue:

Step 1: If (front == -1 || front > rear) then
Print "Queue is Empty"
go to Step 6

Step 2: Set i = front

Step 3: Repeat the Steps until i <= rear

Step 4: Print queue [i]

Step 5: Increment i By 1 Step 6: Stop

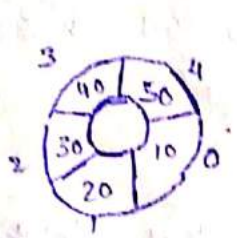
Queue representation using linked list :

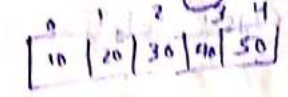
```
Struct node
{
    int data;
    Struct node *link;
};
Struct node *front=NULL, *rear=NULL;
Void insert()
{
    Struct node *temp;
    temp = (Struct node *) malloc (Sizeof (Struct node));
    Printf("Enter data");
    Scanf("%d", &temp->data);
    temp -> link = NULL;
    if(rear == NULL)
    { Printf("First
        rear = temp; } front = rear = temp;
        front = temp;
    }
    Else
    {
        rear -> link = temp;
        rear = temp;
}
Void delete()
{
    if (front == NULL)
        Printf("First is Empty");
    Else
    {
        Printf("Deleted Element is %d", front -> data);
```

```
        struct node *p;
        p = front;
        front = p->link;
        p->link = NULL;
        free(p);
    }
}
void display()
{
    struct node *temp = front;
    if (front == NULL)
        printf("list is empty");
    else
    {
        while (temp != NULL)
        {
            printf("%d", temp->data);
            temp = temp->link;
        }
    }
}
void main()
{
    int ope;
    while(1)
    {
        printf("Enter Choice 1n1.Insert 1n2.delete 1n3.display");
        scanf("%d", &ope);
        switch(ope)
        {
            case 1: insert();
                    break;
            case 2: delete()
                    break;
            case 3: display()
                    break;
            case 4: exit(0);
        }
    }
}
```

Circular Queue :- A queue is represented in circular fashion that mean we can traverse from first to last & again fr last to first

Circular Queue representation using array will be.





Write a program to implement operations on Circular queue using array

```
#define Size 5
int front = -1, rear = -1, q[Size];
void insert()
{
   int val;
   Printf("Enter value"); Scanf("%d", &val);
   if(Size = rear+1 || front = rear+1) // when we crossed array boundary & front == 0)
      Printf("Queue is Full");
   if(front == -1 && rear == -1)
   {
      front = rear = 0;
      q[rear] = val;
   }
   else if(rear == Size-1)
   {
      rear = 0;
      q[rear] = val;
   }
   else
   {
      rear += 1;
      q[rear] = val;
   }
```

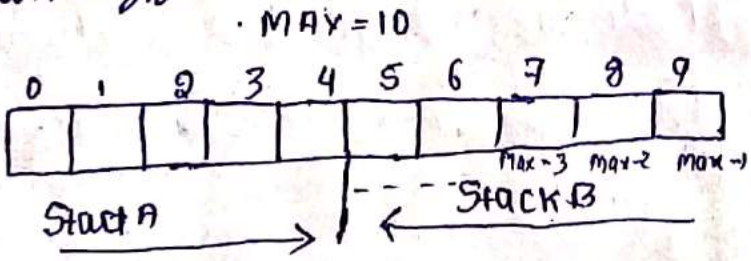
```
Void delete()
{
    if (front == -1 && rear == -1)
        Printf ("Queue is Empty");
    if (front = Size-1)
    {
        Printf ("Element deleted is %d", q[front]);
    } front = 0;
    else if (front == rear)
    {
        Printf ("Element deleted is %d", q[front]);
        front = rear = -1;
    }
    else
    {
        Printf ("Element deleted is %d", q[front]);
        front += 1;
    }
}
Void display()
{
    int i;
    if (front <= rear)
    for (i = front; i <= rear; i++)
        Printf ("%d", q[i]);
    }
    else
    {
        for (i=0; i<= rear; i++)
            Printf ("%d", q[i],
        for (i = front; i< size; i++)
            Printf ("%d", q[i]);
    }
}
```

if (front == -1 && rear == -1)
Printf ("Queue is Empty");

void main()

Multi Stack:- It is multiple implementation of Stacks in a single array. While implementing a stack using array the size of array must be known in advance. If the stack is allocated less space than frequent overflow condition will be encountered. So inorder to increase the efficiency of program & save the memory space than we have a solution called multiple Stacks where more than one stack is used in same array.

If 2 Stacks are represented as Stack A & Stack B Stack A will grow from left → right & Stack B will increase from right → left.



Algorithm for Push A

Step1: If TOPA == TOPB - 1 then
Print "Stack is full"
Go to Step5

Step2: Read value

Step 3: Set TOPA = TOPA + 1

Step 4: Stack[TOPA] = val

Step5: Stop

Algorithm for Push 'B'

Step 1: if (top == top B - 1)
printf ("Stack is full");
go to step 5;

Step 2: Read value;

Step 3: set top B: top B - 1

Step 4: set Stack [Top B] = val;

Step 5: Stop

Algorithm for Pop 'A':

Step 1: if TopA == -1
Printf → "Stack is empty"
goto step 4

Step 2: Print → Stack [Top A]

Step 3: TopA - -

Step 4: Stop

Algorithm for Pop 'B':

Step 1: if TopB == MAX
Print → "Stack is empty"
go to step 4

Step 2: Print → Stack [Top]

Step 3: TopB ++

Step 4: Stop

Algorithm for display 'A':

Step 1: If (top A == -1)
Print -> "stack is empty"
go to step 6

~~Step 2: Read the value~~

Step 2: set i := top A

Step 3: Repeat the following steps 4,5. until i >= 0

Step 4: Print -> stack[i]

Step 5: Decrement i by 1

Step 6: Stop

Algorithm for display 'B':

Step 1: If (top B == MAX)
Print -> "stack is empty"
go to step 6

Step 2: set i := top B

Step 3: Repeat following step until i == max

Step 4: Print -> Stack[i]

Step 5: Increment i by 1

Step 6: Stop

Towers of hanoi:

It is mathamathematical problem which contains 3 poles. A is source, C is destination, B is spare pole using B we have to shift all right on A to C. This problem is solved by using recursion. All the recursive calls will be placed inside stack. the condit for problem is larger rings will be at bottom of and smaller rings at top of pole. We can move o one ring at a time

```
main()
{
    int n;
    Printf("enter no of rings");
    Scanf("%d", &n);
    move(n,'A','C','B');
}
Void move(int n, char source, char dest, char spare)
{
    if(n==1)
        Printf("Move from %c to %c", source, dest);
    else
    {
        move(n-1, source, spare, dest);
        move(1, source, dest, spare);
        move(n-1, spare, dest, source);
    }
}
```