irst Search and Depth First Search Traversals ..... 5 - 9
n Spanning Tree ..... 5 - 19
Path Algorithm ..... 5 - 30
gical Sorting ..... 5 - 36
of Symbol Table ..... 5 - 40
..... 5 - 41
ees ..... 5 - 50
Data Structures ..... 5 - 70
ort ..... 5 - 80
tions of Heap ..... 5 - 87

Unit - VI

6 Hashing and File Organization (6 - 1) to (6 - 37)

les and Scattered Tables ..... 6 - 1
ction ..... 6 - 3
istics of Good Hash Function ..... 6 - 5
or Collision ..... 6 - 6
esolution Techniques ..... 6 - 6
File ..... 6 - 18
ions ..... 6 - 19
and File Organization ..... 6 - 29
n of Different File Organizations ..... 6 - 36

el Question Paper (M - 1) to (M - 2)

U Question Paper (S - 1) to (S - 12)

Unit - I

1 Introduction

1.1 : Concept of Data

Q.1 Define the terms : Data object, data type and data structure.

☞[SPPU : May-12, Marks 6, Dec.-16,17, Marks 4, Dec.-18,19, Marks 2]

Ans. : i) Data Object : • Data object is term that refers to a set of elements . This set can be finite or infinite.

- **For example -** Consider a set of students studying in second year of college. This is a finite set where as the set of natural numbers is an infinite set.

ii) Data Types : • Data type means the type of data.

- This type can be numeric, alphanumeric and so on.
- **For example -** In C there are various data types such as int, float,char and so on.

iii) Data Structure : Data structure is a set of domains D, set of functions F, and set of axioms A. This triple (D,F,A) denotes data structure D.

- **For example :** Arrays is a data structure containing the set of numeric elements on which the operations such as addition, multiplication , subtraction can be performed.

1.2 : Types of Data Structure

Q.2 Give classification of data structures with one example of each type. **☞[SPPU : Dec.-13, May-14, Marks 6]**

OR Explain static and dynamic data structures with suitable examples **☞[SPPU : May-16, Marks 3, Dec.-16, Marks 3]**

Ans. :

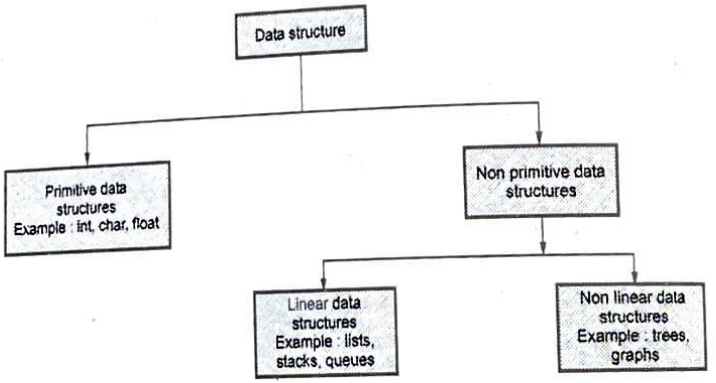


Fig. Q.2.1

1. Primitive and Non Primitive Data Structure

- Primitive data structure is a fundamental data types. For example - int, float, char, double
- Non primitive data structure is built using primitive data types. For example - structure,union, enumerated data types.

2. Linear and Non linear Data Structure

- Linear data structures are the data structures in which data is arranged in a list or in a straight sequence. For example - arrays, linked list.
- Non linear data structures are the data structures in which data may be arranged in hierarchical manner. For example - trees, graphs.

3. Static and Dynamic Data Structure

- Static data structures are data structures having fixed memory size. For example arrays is static data structure.
- Dynamic data structures are data structures kin which the memory can be allocated as per the requirements. For example - Linked list is implementation is dynamic data structure.

4. Persistent and Ephimeral Data Structure

- The persistent data structures are the data structures which can retain their previous state after performing some operations. For example – stack.
- The ephemeral data structures are the data structures in which we cannot retain its previous state. For example - queue.

1.3 : Definition of ADT

Q.3 What is abstract data type? Explain ADT for an array

☞[SPPU : May-06, 07, Marks 6, Dec.-16,17,19, Marks 2]

Ans. : The abstract data type is a triple of D - set of domains, F - set of functions, A - axioms in which only what is to be done is mentioned but how is to be done is not mentioned.

In ADT, all the implementation details are hidden.

AbstractDataType Array

{

Instances : An array A of some size, index i and total number of elements in the array n.

Operations :

Store() - This operation stores the desired elements at each successive location.

display() - This operation displays the elements of the array.

}

1.4 : Introduction to Algorithm

Important Point to Remember

- An algorithm is a finite set of instructions for performing particular tasks. These statements are written in simple English language.

Q.4 Write Characteristics of an algorithm. **☞[SPPU : Dec.-11, Marks 4]**

Ans. :

1. Each algorithm is supplied with **zero or more inputs.**
2. Each algorithm must **produce** at least one **output.**

3. Each algorithm should have **definiteness** i.e. each instruction must be **clear and unambiguous.**
4. Each algorithm should have **finiteness** i.e. if we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after finite number of steps.

Each algorithm should have **effectiveness** i.e. every instruction must be sufficiently basic that it can in principal be carried out by a person using only pencil and paper.

Q.5 Write pseudo code to find union of two sets containing integer elements. **☞[SPPU : May-10, Marks 4]**

Ans. :

```
void Union(int a[],int b[],int term1,int term2)
{
int c[MAX];
int i,j,k,n,flag;
k=0;
flag=0;
for(i=0;i<3;i++)
{
      c[k++]=a[i];
}
for(j=0;j<3;j++)
{
      flag=0;
      for(i=0;i<3;i++)
      {
         if(a[i]==b[j])
            flag=1;
   }
   if(flag==0)
      c[k++]=b[j];
}
n=k;
```



```
      printf("\n The A union B={");
      for(k=0;k<n;k++)
      printf(" %d",c[k]);
      printf(" }");
}
```

1.5 : Analysis of Algorithm

Important Point to Remember

- Algorithmic analysis can be carried by computing time complexity and space complexity.
- The time complexity is not computed in terms of clock time but computed with the help of frequency count.
- The frequency count is a count that denotes how many times particular statement is executed.

Q.6 Obtain the frequency count for the following code

```
for(i=1;i<=n;i++)
{
  for(j=1;j<=n;j++)
  {
     c[i][j]=0;
     for(k=1;k<=n;k++)
        c[i][j]=c[i][j]+a[i][k]*b[k][j];
  }
}
```

☞[SPPU : May-13, Marks 8; Dec.-13, Marks 3]

Ans. : After counting the frequency count, the constant terms can be neglected and only the order of magnitude is considered. The time complexity is denoted in terms of algorithmic notations. The Big oh notation is a

Statement	Frequency Count
for(i=1;i<=n;i++)	n+1
for(j=1;j<=n;j++)	n.(n+1)
c[i][j]=0;	n.(n)
for(k=1;k<=n;k++)	n.n(n+1)
c[i][j]=c[i][j]+a[i][k]*b[k][j];	n.n.n
Total	$2n^3+3n^2+2n+1$

most commonly used algorithmic notation. For the above frequency count all the constant terms are neglected and only the order of magnitude of the polynomial is considered. Hence the time complexity for the above code can be $O(n^3)$. The higher order is the polynomial is always considered.

Q.7 Determine the frequency counts for all the statements in the following program segment

```
i = 10; for (i = 10; i< = n; i++)
for (j=1; j < i; j++) x = x+1;
```

[SPPU : May-13, Marks 6]

Ans. : Assume $(n-10+2) = m$. Then the total frequency count is -

Statement	Frequency Count
i=10;	1
for(i=10;i<=n;i++)	m (we assume the count of execution is m)
for(j=1;j<i;j++)	m((m+1)/2)
x=x+1;	m(m)

The outer loop will execute for m times. The inner loop is dependant upon the outer loop. Hence it will be (m+1)/2. Hence overall frequency count is $(1+m+m^2)(m(m+1)/2)$

Q.8 What do you mean by frequency count and its importance in analysis of an algorithm **[SPPU : May-12, 13, Marks 6]**

Ans. : Frequency count is a count that denotes how many times particular statement gets executed. Computing frequency count is important when we want to compute the time complexity or space complexity of an algorithm. Example of computing frequency count : Refer Q.6

Q.9 What is time complexity? How is time complexity of an algorithm computed? **[SPPU : Dec.-11, Marks 6]**

OR What is time complexity of an algorithm? Explain its importance with suitable examples

Ans. : The amount of time required by an algorithm to execute is called the time complexity of that algorithm.

For determining the time complexity of particular algorithm following steps are carried out -

1. Identify the basic operation of the algorithm
2. Obtain the frequency count for this basic operation.
3. Consider the order of magnitude of the frequency count and express it in terms of big oh notation.

Q.10 Write an algorithm to find smallest element in a array of integers and analyze its time complexity. **[SPPU : May-13, Marks 8]**

Ans. :

```
Algorithm MinValue(int a[n])
{
    min_element=a[0];
    for(i=0;i<n;i++)
    {
        if (a[i]<min_element) then
            min_element=a[i];
    }
    Write(min_element);
}
```

Statement	Frequency Count
min_element=a[0]	1
for(i=0;i<n;i++)	n+1
if (a[i]<min_element) then	n
Write(min_element);	1
Total	2n+3

We will first obtain the frequency count for the above code

By neglecting the constant terms and by considering the order of magnitude we can express the frequency count in terms of Omega notation as O(n). Hence the frequency count of above code is O(n).

Q.11 What is space complexity of an algorithm ? Explain its importance with example.

[SPPU : Dec.-10, Marks 4, Dec.-14, Marks 3, May-17, Marks 3]

Ans. : The space complexity can be defined as amount of memory required by an algorithm to run.

To compute the space complexity we use two factors : **constant and instance characteristics**. The space requirement S(p) can be given as

$$S(p) = C + Sp$$

where C is a constant i.e. fixed part and it denotes the space of inputs and outputs. This space is an amount of space taken by instruction, variables and identifiers. And Sp is a space dependent upon instance characteristics. This is a variable part whose space requirement depends on particular problem instance.

1.6 : Asymptotic Notations

Important Points to Remember

- For representing the time complexity of an algorithm the asymptotic notations are used.
- The asymptotic notations give the time complexity as "fastest possible" , "slowest possible" and "average time"
- The notations such as O, Ω , and Θ are called asymptotic notations.

Q.12 With respect to algorithm analysis, explain the following terms : i) Omega notation ii) Theta notation iii) Big oh notation

[SPPU : May-10, Dec.-11, Marks 6]

Dec.-16, Marks 6, May-18, Marks 6, Dec.-19, Marks 6

OR What are different asymptotic notations

May-17, May-18, Marks 3, Dec.-18, Marks 6

Ans. : Various notations such as Ω, Θ and O used are called **asymptotic notations.**

1. Big oh notaton : The **Big oh** notation is denoted by 'O'.

Let f(n) and g(n) be two non-negative functions.

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that $c > 0$. We can write

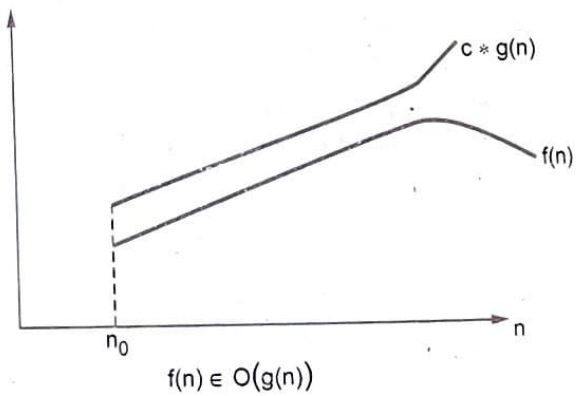


Fig. Q.12.1 Big oh notation

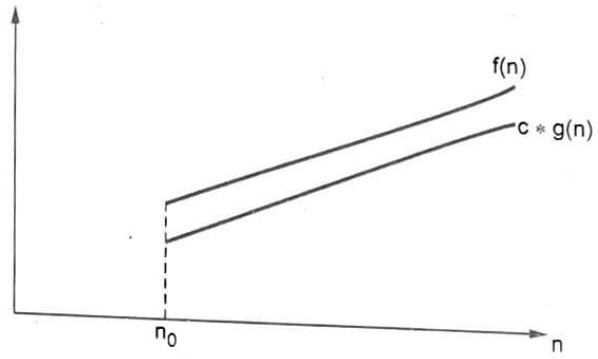


Fig. Q.12.2 Omega notation f(n) ∈ Ω (g(n))

$$f(n) \leq c^*g(n)$$

then f(n) is big oh of g(n). It is also denoted as **f(n) ∈ O (g(n)).**

2. Omega notation : Omega notation is denoted by 'Ω'.

A function f(n) is said to be in Ω (g(n)) if f(n) is bounded below by some positive constant multiple of g(n) such that

$$f(n) \geq c * g(n)$$

For all $n \geq n_0$

It is denoted as **f(n) ∈ Ω (g(n)).**

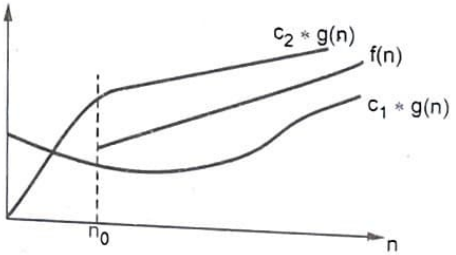


Fig. Q.12.3 Theta notation f(n) ∈ Θ (g(n))

3. Theta notation : The theta notation is denoted by Θ.

Let f(n) and g(n) be two non negative functions. There are two positive constants namely c_1 and c_2 such that

$$c_1 \leq g(n) \leq c_2\ g(n)$$

Then we can say that

$$f(n) \in \Theta\ (g(n))$$

Some Examples of Asymptotic Order

- $\log_2 n \in O(n)$ $\because \log_2 n \leq O(n)$, the order of growth of $\log_2 n$ is slower than n.

But
$\log_2 n \notin \Omega(n)$ $\because \log_2 n \leq \Omega(n)$ and if a certain function f(n) is belonging to $\Omega(n)$ it should satisfy the condition $f(n) \geq c * g(n)$

- $n(n-1)/2 \notin O(n)$ $\because$ $f(n) > O(n)$ we get $F(n) = n(n-1)/2 = \frac{n^2 - 1}{2}$

i.e. maximum order is n^2 which is $> O(n)$.
Hence $f(n) \notin O(n)$

1.7 : Best, Worst and Average Case of Algorithm

Q.13 Explain the terms best case, worst case and average case ananlysis.

Ans. : If an algorithm takes minimum amount of time to run to completion for a specific set of input then it is called **best case** time complexity.

For example : While searching a particular element by using sequential search we get the desired element at first place itself then it is called best case time complexity.

If an algorithm takes maximum amount of time to run to completion for a specific set of input then it is called **worst case** time complexity.

For example : While searching an element by using linear searching method if desired element is placed at the end of the list then we get worst time complexity.

The time complexity that we get for certain set of inputs is as a average same. Then for corresponding input such a time complexity is called **average case** time complexity. Consider the following algorithm.

1.8 : Single and Multidimensional Arrays

Q.14 What is sequential memory organization ? List the advantages and disadvantages of sequential memory organization.

[SPPU : Dec.-15, Marks 5]

Ans. : Sequential memory organization means storing the records in contiguous block of memory. In this memory organization, the memory is allocated before storing the records. Array is typically referred as sequential memory organization.

Advantages of sequential organization of data structure

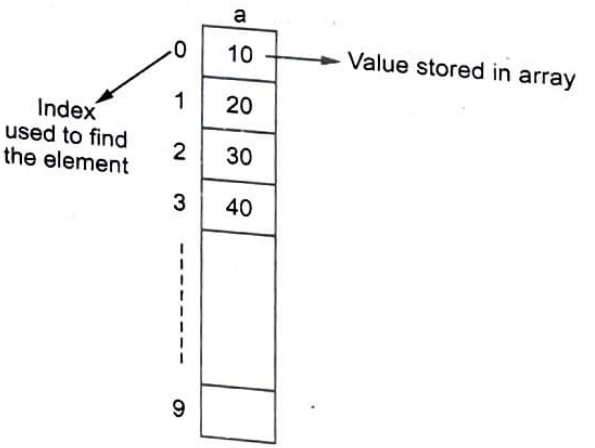
1. Elements can be retrieved or stored very efficiently in sequential organization with the help of index or memory location.
2. All the elements are stored at continuous memory locations. Hence searching of element from sequential organization is easy.

Disadvantages of sequential organization of data structure

1. Insertion and deletion of elements becomes complicated due to sequential nature.
2. For storing the data large continuous free block of memory is required.
3. Memory fragmentation occurs if we remove the elements randomly.

Q.15 What is one and two dimensional arrays ?

Ans. : One dimensional array : The one dimensional array 'a' is declared as int a[10]



Two dimensional array : If we declare a two dimensional array as int a[10] [3];

Then it will look like this -

Column ↓

row →	0	1	2
0	10	20	30
1	40	50	60
2			
.			
.			
.			
9			

The two dimensional array should be in row-column form.

1.9 : Address Calculation

Q.16 Give row major storage representation for two dimensional array. Write address calculation. **☞[SPPU : Dec.-13, Marks 4]**

OR What is row major and column major representation of arrays.

☞[SPPU : Dec.-08, 11, 15, Marks 4]

Ans. : Row Major Representation : If the elements are stored in **rowwise manner** then it is called row major representation.

For example : If we want to store elements

10 20 30 40 50 60 then in a two dimensional array

The ⇒ elements will be stored horizontally

	0	1	2
0	10	20	30
1	40	50	60
.			
.			
9			

To access any element in two dimensional array we must specify both its row number and column number. That is why we need two variables which act as row index and column index.

Column Major Representation : If elements are stored in **column wise manner** then it is called column major representation.

For example : If we want to store elements

10 20 30 40 50 60 then the elements will be filled up by columnwise manner as follows (consider array a[3] [2]). Here 3 represents number of rows and 2 represents number of columns.

	0	1
0	10	40
1	20	50
2	30	60

Each element is occupied at successive locations if the element is of integer type then 2 bytes of memory will be allocated, if it is of floating type then 4 bytes of memory will be allocated and so on.

Address Calculation for Row Major Array :

For calculating the address of any element in the array following formula is used -

address of a[i][j] = base address + i * n +j

Where the array is declared as a[m][n]. Here m,n represents the size of rows and columns respectively.

For example : Consider int a[2][3] = { {10, 20, 30}

{40, 50, 60}};

If we want to find out the address of a[1][2] i.e. location of element 60 then

= 102 + 1* 3 + 2 = 102 + 5

That is, for finding address of element 60 from base address we should move 5 places ahead i.e. at address **112**.

Address calculation for Column Major Array :

For calculating the address of any element in the array following formula is used-

address of a[i][j] = base address + j * m + i

For example : If we want to find out the address of a[1][2] from above given 2D array i.e. location of element 60 then

= 102 + 2 * 2 + 1 = 102 + 5

That is for finding address of element 60 from base address we should move 5 places ahead i.e. at address **112**.

Q.17 Explain the concept of column major address calculation for multidimensional array with suitable example.

[SPPU : May-10, 11, Marks 6, Dec.-14, Marks 4]

OR Explain the two dimensional array in detail with column and row major representation and address calculation in both the cases.

[SPPU : Dec.-16,17, May-17,18, Marks 6, Dec.-19, Marks 3]

Ans. : Let α represents the base address

The n dimensional array can be declared as $A[u_1][u_2] \ldots [u_n]$.

Now the two types of representations are as given below.

Row Major Representation

$$A[i_1][i_2][i_3]\ldots[i_n] = \alpha + (i_1 - 1)u_2 * u_3 * \ldots u_n + (i_2 - 1)u_3 * u_4 \ldots u_n + (i_3 - 1) * u_4 * u_5 \ldots u_n + (i_n - 1)$$

Column Major Representation

For column major representation the formula will be

$$A[i_1][i_2][i_3]\ldots[i_n] = \alpha + (i_n - 1) * u_2 * u_3 * \ldots u_n + (i_{n-1} - 1) * u_3 * u_4 * \ldots u_n + (i_1 - 1)$$

For example

Consider an array A[2] [3] [4] with base address 100. The size of each element is 4 bytes and we have to obtain memory location of A[1] [1] [2].

Given that

$\alpha 1 = 2$, $\alpha 2 = 3$ and $\alpha 3 = 4$

$a = 1$, $b = 2$, $c = 1$, $d = 3$, $e = 1$, $f = 4$

$p = 1$, $q = 1$, $r = 2$, $w = 4$, Base address = 100.

Row Major Order

For calculating address of A [1] [1] [2] -

Base address + w * [(i * $\alpha 2$ + j) * $\alpha 3$ + k]

= 100 + 4 * [((p − a)* $\alpha 2$ + (q − c)) * $\alpha 3$ + (r − e)]

= 100 + 4 * [((1 − 1)* 3 + (1 − 1)) * 4 + (2 − 1)]

= 100 + 4 * [1] = **104**

Column Major Order

For calculating address of A[1] [1] [2] -

Base address + w*[(k * $\alpha 2$ + j) * $\alpha 1$ + i]

= Base address + 4 * [(1 * 3 + 0) * 2 + 0]

= 100 + 4 * 6 = **124**

Q.18 Write address calculation for elements of one dimensional array.

[SPPU : May-18, Marks 2]

Ans. : The address calculation formula for one dimensional array is -

A[i] = BaseAddress + Width * (i - lower bound)

For example -

Consider base address = 100,

Width = 2 bytes (integer)

lower bound = 1

To find address of A[4], we get -

A[4] = 100 + 2 * (4 − 1)

= 100 + 2 * 3

A[4] = 106

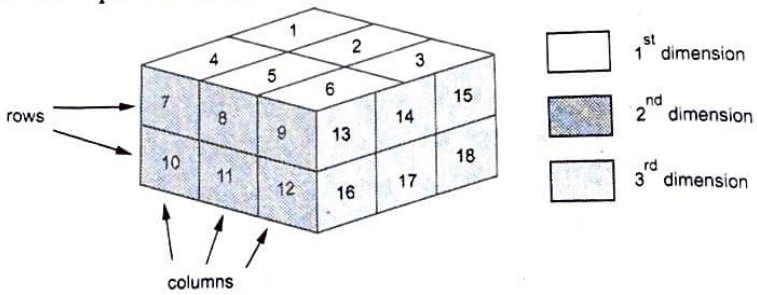
Q.19 Write a note on multidimensional arrays.

Ans. : The multidimensional array is similar to the two dimensional array with multiple dimensions for example Here is 3-D array

a [3] [2] [3] = { { {1,2,3}, {4,5,6} }, { {7,8,9}, {10,11,12} }, { {13,14,15} {16,17,18} } }

dimension ← rows column

We can represent it graphically as



1.10 : Concept of Linked Organization

Important Points to Remember

- **Linked List** is referred as the **Linked organization.**
- In this organization, data is arranged in a linked list.
- The linked organization is also called the dynamic data structure because the nodes in the linked list are allocated or de-allocated as per the requirement.
- The dynamic memory allocation functions supports the creation of linked list.

Q.20 Write advantages of linked memory organization.

[SPPU : Dec.-08, Marks 2, Dec.-13, Marks 3]

OR What is dynamic data structure? List the advantages of linked lists

[SPPU : Dec.-05, May-06,09, Marks 6]

Ans :

Advantages of linked organization

1. In linked organization, insertion and deletion of elements can be efficiently done.
2. There is no wastage of memory. The memory can be allocated and deallocated as per requirement.

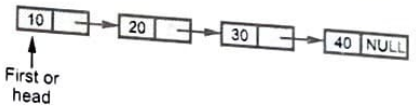
Disadvantages of linked organization

1. Linked organization does not support random or direct access.
2. Each data field should be supported by a link field to point to next node.

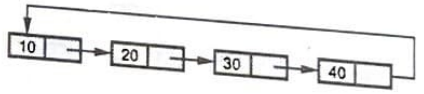
Q.21 Explain the different types of linked lists.

[SPPU : May-06, Dec.-08, Marks 4]

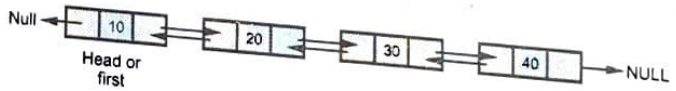
Ans. : • **Singly linear linked list :** It is called singly because this list consists of only one link, to point to next node or element. This is also called linear list because the last element points to nothing it is linear is nature. The last field of last node is NULL which means that there is no further list. The very first node is called head or first.



• **Singly circular linked list :** In this type of linked list only one link is used to point to next element and this list is circular means that the last node's link field points to the first or head node. That means according to example after 40 the next number will be 10. So the list is circular in nature.

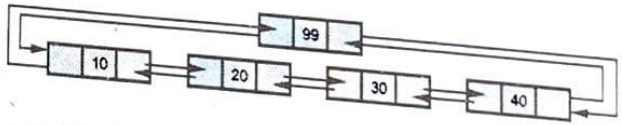


• **Doubly linear linked list :**



This list called doubly because each node has two pointers previous and next pointers. The previous pointer points to previous node and next pointer points to next node. Only in case of head node the previous pointer is obviously NULL and last node's next pointer points to NULL. This list is a linear one.

• **Doubly circular linked list :**



In circular doubly linked list the previous pointer of first node and the next pointer of last node is pointed to head node.

1.11 : Linked List as ADT

Q.22 Explain the concept of Linked List.

Ans. : A linked list is a set of nodes where each node has two fields ' data' and 'link'. The 'data' field stores actual piece of information and 'link' field is used to point to next node.

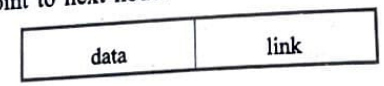


Fig. Q.22.1 Structure of node

Hence link list of integers 10, 20, 30, 40 is

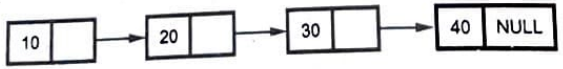


Fig. Q.22.2 Linked list

Q.23 Give the C representation of linked list.

OR Explain - Self-referential structure. ☞**[SPPU : Dec.-18, Marks 3]**

Ans. : 'C' structure

```
typedef struct node
{
   int data; /* data field */
   struct node * next; /* link field */
} SLL;
```

1.12 : Singly Linked List

Q.24 Write a C/C++ program to create singly linked list of integers and display it forward.

☞**[SPPU : May-14, Marks 6, June-22, Marks 9]**

Ans. : 1. Creation of Linked List :

```
node*  create()
{
   node *temp, *New, *head;
   int   val, flag;
   char ans='y';
```



```
   node *get_node();
   temp = NULL ;
   flag = TRUE; /* flag to indicate whether a new node
         is created for the first time or not */
   do
   {
      printf("\nEnter the Element :");
      scanf("%d", &val);
      /* allocate new node */
      New =get_node();
      if ( New == NULL )
         printf("\nMemory is not allocated");
      New -> data  = val ;
      if (flag==TRUE) /* Executed only for the first time */

      {
         head = New;
         temp=head; /*head is a first node in the  SLL*/
         flag = FALSE;
      }
      else
      {
         /* temp keeps track of the most recently created node */
         temp ->next = New;
         temp = New;
      }
   printf("\n Do you Want to enter more elements?(y/n)");
   ans=getche();
   }while(ans=='y');
   printf("\nThe Singly Linked List is created\n");
   getch();
   clrscr();
   return head;
}
node *get_node()
{
 node *temp;
 temp=( node *) malloc( sizeof(node) );
```

```
 temp->next=NULL;
 return temp;
}
```

2. Display of Linked List

```
void  display(node *head)
{
   node *temp ;
   temp = head;
   if ( temp == NULL )
   {
      printf("\nThe list is empty\n");
      getch();
      clrscr();
      return;
   }
while ( temp != NULL )
   {
      printf(" %d -> ", temp->data );
      temp = temp -> next;
   }
   printf("NULL");
   getch();
   clrscr();
}
```

Q.25 Write C function for insertion a node in a linked list.

Ans. : There are three cases for insertion of a node

Case 1 : Insertion of head node

```
node *insert_head(node *head)
{
 node *New,*temp;
 New=get_node();
 printf("\n Enter The element which you want to insert");
 scanf("%d",&New->data);
```



```
 if(head==NULL)
    head=New;
 else
 {
  temp=head;
  New->next=temp;
  head=New;
 }
 return head;
}
```

No node in the linked list. i.e. when linked list is empty

Case 2 : Insertion intermediate node

```
void insert_after(node *head)
{
 int key;
 node *New,*temp;
 New= get_node();
 printf("\n Enter The element which you want to insert");
 scanf("%d",&New->data);
 if(head==NULL)
 {
 head=New;
 }
 else
 {
  printf("\n Enter The element after which you want to insert the
node");
 scanf("%d",&key);
 temp=head;
 do
 {
   if(temp->data==key)
   {
```

```
        New->next=temp->next;
        temp->next=New;
        return;
      }
  else
     temp=temp->next;
  }while(temp!=NULL);
}
}
```

Case 3 : Insertion of last node

```
void insert_last(node *head)
{
 node *New,*temp;
 New=get_node();
 printf("\n Enter The element which you want to insert");
 scanf("%d",&New->data);
 if(head==NULL)
   head=New;
 else
 {
    temp=head;
    while(temp->next!=NULL)
      temp=temp->next;
    temp->next=New;
    New->next=NULL;
   }
}
```

Finding end of the linked list. Then **temp** will be a last node

Q.26 Write a C code for inserting a node at start and at end in SLL

[SPPU : May-10, Marks 10, May-16, Marks
May-17, Marks 6, May-18, Marks

Ans. : Refer Q.25.

Q.27 Write pseudo code to delete a node from singly linked list

[SPPU : Dec.-15, Marks

Ans. :

```
void dele(node **head)
{
 node *temp,*prev ;
 int key;
 temp = *head;
 if ( temp == NULL )
 {
   printf("\nThe list is empty\n");
   getch();
   clrscr();
   return ;
 }
clrscr();
printf("\nEnter the Element you want to delete: ");
scanf("%d", &key);
temp = search(*head,key);
if ( temp != NULL )
{
   prev = get_prev(*head,key);
   if ( prev != NULL )
   {
       prev -> next = temp->next;
       free (temp);
   }

   else
   {
       *head = temp ->next;
       free(temp);
   }
   printf("\nThe Element is deleted\n");
   getch();
   clrscr();
 }
}
```

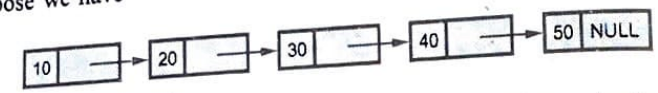
Always check before deletion whether the linked list is empty or not. Because if th e list is empty there is no point in performing deletion

Firstly, Node to be deleted is searched in the linked list

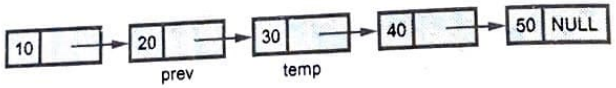
Once the node to be deleted is found then get the previous node of that node in variable **prev**

If we want to delete head node then set adjacent node as a new head node and then deallocate the previous head node

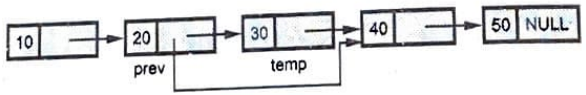
Suppose we have



Suppose we want to delete node 30. Then we will search the node containing 30, using search(* head, key) routine. Mark the node to be deleted as **temp**. Then we will obtain previous node of **temp** using get_prev( ) function. Mark previous node as **prev**

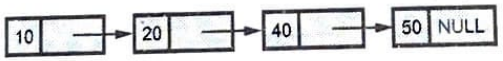


Then

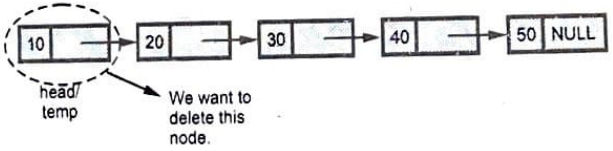


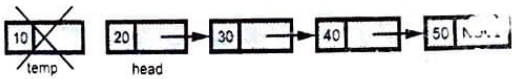
prev → next = temp → next

Now we will **free** the **temp** node using free function. Then the linked list will be -



Another case is, if we want to delete a head node then -





This can be done using following statements

```
*head = temp → next;
free (temp);
```


Q.28 Write a C function for searching of a node from singly linked list.

OR Write an algorithm to search an element from a linked list.

Ans. : Algorithm

1. Start reading the linked list from the first node or head node.
2. Compare data field of each node with the key value.
3. If the data field value is not matching with the key value then move on to the next node using next pointer.
4. If the data value is equal to the key value then display the message "Node is found" and return from the search routine.

C Function

The search function is for searching the node containing desired value.

```
node *search(node *head, int key)
//The head node and key
//is passed to this function
{
   node *temp;
   int found;
   temp = head;
   if ( temp == NULL )
   {
      printf("The Linked List is empty\n");
      getch();
      clrscr();
      return NULL;
   }
   found = FALSE;
   while ( temp != NULL && found==FALSE)
   {
      if ( temp->data != key)
         temp = temp -> next;
      else
```

Compare data at each node with the key value. If not matching then move to next node

```
            found = TRUE;
        }
        if ( found==TRUE )
        {
         printf("\nThe Element is present in the list\n");
         getch();
         return temp;
        }
        else
        {
         printf("The Element is not present in the list\n");
         getch();
         return NULL;
        }
}
```

If the node containing desired data is obtained in the linked list then found variable is set to TRUE.

Q.29 Suppose a linked list consists of numerical values. Write a function for finding the maximum element of the list and the product of all the numbers in the list. **☞[SPPU : Dec.-14, Marks 7]**

Ans. : i) To find the maximum element in the linked list :

```
void maxElement(node *root)
{
    node *temp;
    int Maxval;
    temp=root;
    Maxval=temp->data;
    while(temp!=NULL)
    {
        if(temp->data>Maxval)
        {
            Maxval=temp->data;
        }
        temp=temp->next;
    }
    printf("\n Maximum Value is: %d",Maxval);
}
```


ii) To find the product of all numbers in the linked list :

```
void Product(node *root)
{
    node *temp;
    int p=1;
    temp=root;
    while(temp!=NULL)
    {
        p=p*temp->data;
        temp=temp->next;
    }
    printf("\n Product of all the elements: %d",p);
    }
```

Q.30 Write a C function to reverse a linear singly linked list by changing link pointers. Write its time complexity.

☞[SPPU : Dec.-13, Marks 6, May-16, Marks 7]

OR Write a C function to reverse a singly linked list using three pointer.

Ans. : **☞ [SPPU : Dec.-17, Marks 4]**

```
void reverse ()
    {
    node *temp1,*temp2,*temp3;
    temp1=head;
    if(temp1==NULL)
    {
     printf("\n The List is empty");
     getch();
    }
    else
    {
     temp2=NULL;
      while(temp1!=NULL)
      {
       temp3=temp2;
       temp2=temp1;
```

```
        temp1=temp1->next;
        temp2->next=temp3;
      }
      head=temp2;
    }
    printf("\n The List is REVERSED");
  }
```

The time complexity is O(n) where n stands for number of nodes in a linked list.

Q.31 Assume a singly linked list where each node contains student details like name, rollno and percentage of marks. Write a "C" function COUNT() to traverse the linked list and count how many students have obtained more than 60% marks.

☞[SPPU : Dec.-05, Marks 6]

Ans. : The data structure can be -

```
struct student
{
   char *name;
   int roll;
   float marks;
   struct student *next;
};
```

The COUNT function can be written as -

```
void COUNT(struct student *temp)
{
   int count=0;
   while(temp!=NULL)
   {
      if(temp->marks>60)
         count++;
      temp=temp->next;
   }
   printf("\n\n %d students have obtained more than 60 Percentage",
count);
}
```


Q.32 Write a C program to create a singly linked list and split it at the middle. And make the second half as the first and vice versa, display the final list.

☞[SPPU : May-07, Marks 8]

Ans. :

```
typedef struct node
{
      int data;
      struct node *next;
}node;
void main()
{
      node*head=NULL,*p,*q;
      int n,i,x;
      printf("\nEnter number of nodes:");
      scanf("%d",&n);
      for(i=0;i<n;i++)
      {
         printf("\nEnter element:");
         scanf("%d",&x);
         p=(node*)malloc(size of(node));
         p→data=x;
         p→next=NULL;
         if(head==NULL)
            head=q=p;
         else
         {
            q→next=p;
            q=p;
         }
      }
      p=q=head;
      while(q→next!=NULL)
      {
```

```
        p=p→next;
        q=q→next;
        if(q→next!=NULL)
            q=q→next;
    }
    q→next=head;
    head=p→next;
    p→next=NULL;
    for(p=head;p!=NULL;p=p→next)
        printf("\n%d",p→data);
}
```

Q.33 Write a function that removes all duplicate elements from a linear singly linked list. **☞[SPPU : May-07, Marks 6]**

Ans. :

```
void dupdelete(node*head)
{
    node *p,*q,*r;
    p=head;
    q=p→next;
    while(q!=NULL)
    {
        if(p→data!=q→data)
        {
            p=p→next;
            q=q→next;
        }
        else
        if(p→data==q→data)
        {
            p→next=q→next;
            r=q;
            q=q→next;
        free(r);
    }
    return(head);
    }
}
```


Q.34 Write a pseudo C algorithm to merge two sorted linked lists into the third. **☞[SPPU : Dec.-16, Marks 7**

Ans. : 1. Set temp1 as head node of first linked list and temp2 as hea node of second linked list.

2. temp3 = new_node(). The new_node function will create a new node for temp3 variable.

3.

```
while (temp1 != NULL && temp2 != NULL)
{
    if (temp1 → data < temp2 → data)
    {
        temp3 = temp1;
        temp1 = temp1 → next;
    }
    else
    {
        temp3 = temp2;
        temp2 = temp2 → next;
    }
    temp3 = temp3 → next;
}
```

4. If one of the list ends then copy remaining list to temp3.

```
while (temp1 != NULL)
    {
        temp3 = temp1 ;
        temp1 = temp1 → next;
        temp3 = temp3 → next;
    }
while (temp2!= NULL)
{
    temp3 = temp2;
    temp2 = temp2 → next;
    temp3 = temp3 → next;
}
```

5. Finally return the head node of temp3 list.

1.13 : Doubly Linked List

Q.35 What is doubly linked list ? Give its C structure.

☞ **[SPPU : Dec.-18, May-19, Marks 3]**

Ans. : Doubly linked list is a linked list in which each node contains two pointers previous and next along with data field.

Fig. Q.35.1 Node in DLL

'C' structure of doubly linked list :

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

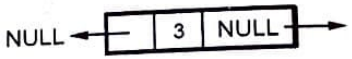
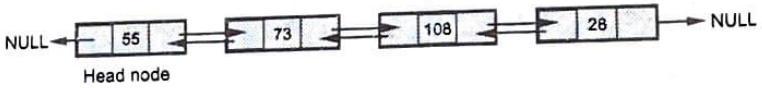


Fig. Q.35.2 Example of node in DLL

The linked representation of a doubly linked list is

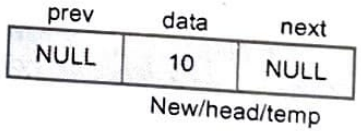


Q.36 Explain the creation of linked list with C code

Ans. :

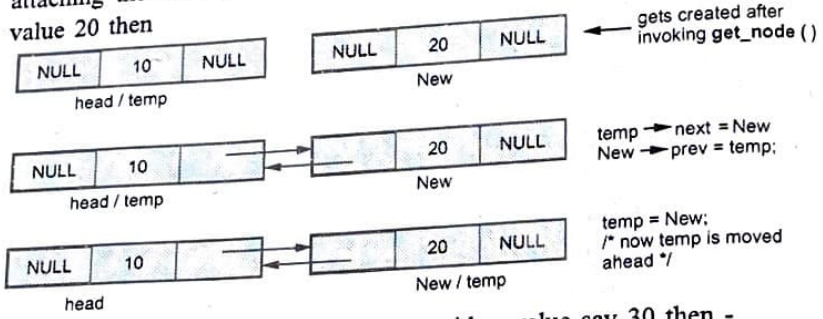
```
node*  create()
{
   node *temp, *New, *head;
   int  val, flag;
   char ans='y';
   node *get_node();
   temp = NULL ;
   flag = TRUE; /* flag to indicate whether a new node
          is created for the first time or not */
   do
   {
      printf("\nEnter the Element :");
      scanf("%d", &val);
      /* allocate new node */
      New =get_node();
      if ( New == NULL )
         printf("\nMemory is not allocated");
      New -> data  = val ;
      if (flag==TRUE) /* Executed only for the first time */
      {
         head = New;
         temp=head; /*head is a first node in the DLL*/
         flag = FALSE;
      }
      else
      {
         /* temp keeps track of the most recently created node */
         temp ->next = New;
         New->prev=temp;
         temp = New;
      }
   printf("\n Do you Want to enter more elements?(y/n)");
   ans=getche();
   }while(ans=='y');
   printf("\nThe Doubly Linked List is created\n");
   getch();
   clrscr();
   return head;
}
```

Explanation : Step 1 : Initially one variable **flag** is taken whose value is initialized to **TRUE**. The purpose of this **flag** is for making a check on creation of first node. After creating first node reset **flag** (i.e. assign **FALSE** to **flag**)

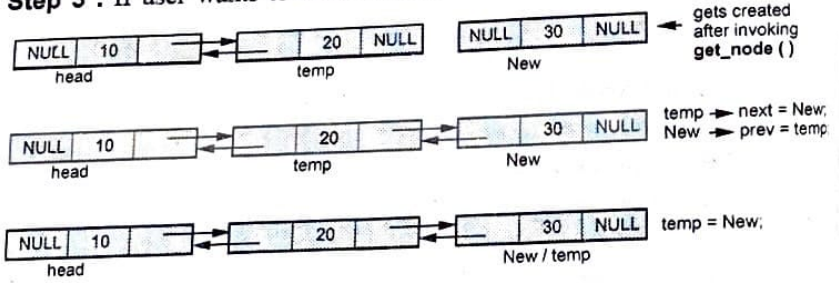


Now, set flag = FALSE;

Step 2 : If **head** node is created, we can further create a linked list by attaching the subsequent nodes. Suppose, we want to insert a node with value 20 then



Step 3 : If user wants to insert a node with a value say 30 then -



Continuing in this fashion doubly linked can be created.

Q.37 Explain the insertion of node in doubly linked list at :

i) The start of the list; ii) The end of the list;

iii) After the position. ☞[**SPPU : May-06, Dec.-08,16, Marks 6, Dec.-16, Marks 4, Dec.-19, Marks 6**

Ans. :

i) The start of list

```
node *insert_head(node *head)
  {
   node *New,*temp;
   New=get_node();
   printf("\n Enter The element which you want to insert");
   scanf("%d",&New->data);
   if(head==NULL)
      head=New;
```



```
   else
   {
      temp=head;
      New->next=temp;
      temp->prev=New;
      head=New;
   }
  return head;
  }
```

ii) The end of the list

```
void insert_last(node *head)
{
      node *New,*temp;
      New=get_node();
      printf("\n Enter The element which you want to insert");
      scanf("%d",&New->data);
      if(head==NULL)
      head=New;
      else
      {
      temp=head;
      while(temp->next!=NULL)
         temp=temp->next;
      temp->next=New;
      New->prev=temp;
      New->next=NULL;
      }
}
```

iii) After the position

```
void insert_after(node *head)
   {
   int key;
   node *New,*temp;
   New= get_node();
   printf("\n Enter The element which you want to insert");
   scanf("%d",&New->data);
   if(head==NULL)
```

```
    {
      head=New;
    }
    else
    {
      printf("\n Enter The element after which you want to insert the
node");
      scanf("%d",&key);
      temp=head;
      do
      {
      if(temp->data==key)
      {
        New->next=temp->next;
        (temp->next)->prev=New;
        temp->next=New;
        New->prev=temp;
        return;
      }
      else
        temp=temp->next;
      }while(temp!=NULL);
    }
  }
```

Q.38 Write a pseudo C code for deletion of node form DLL from first and last position. **[SPPU : May-10, Marks 10]**

OR Write a pseudo C code to delete a node form a doubly linked list **[SPPU : May-13, Marks 6]**

[SPPU : Dec.-16, Marks 3, May-17, Marks 6, May-18, Marks 7

Ans. :

```
void dele(node **head)
{
    node *temp,*prev_node ;
    int key;
    temp = *head;
    if ( temp == NULL )
```



```
    {
        printf("\nThe list is empty\n");
        getch();
        clrscr();
        return ;
    }
    clrscr();
    printf("\nEnter the Element you want to delete: ");
    scanf("%d", &key);
    temp = search(*head,key);
    if ( temp != NULL )
    {
        prev_node =temp->prev;
        if (prev_node != NULL )
        {
            prev_node -> next = temp->next;
            (temp->next)->prev = prev_node;
            free (temp);
        }
        else
        {
            *head = temp ->next;
            (*head)->prev=NULL;
            free(temp);
        }
        printf("\nThe Element is deleted\n");
        getch();
        clrscr();
    }
}
```

Q.39 Write pseudo C function to insert a node before and after any node in doubly linked list and for deletion of specified node.

[SPPU : May-07, Marks 10]

Ans. : Insertion of a node : Refer Q.37

Deletion of specified node : Refer Q.38

Q.40 Write a C function to display the doubly linked list in forward and backward direction.

Ans. :

```
void display(node *head)
{
   node *temp ;
   temp = head;
   if ( temp == NULL )
   {
      printf("\nThe list is empty\n");
      getch();
      clrscr();
      return;
   }
   printf("\n List in Forward direction is ...\n");
   while ( temp != NULL )
   {
      printf(" %d -> ", temp->data );
      temp = temp -> next;
   }
   printf("NULL");
   temp=head;
   while(temp->next!=NULL)
      temp=temp->next;//reaching at the last node
   printf("\n List in Reverse direction is ...\n");
   while ( temp != NULL )
   {
      printf(" %d-> ", temp->data );
      temp = temp -> prev;
   }
   printf("NULL");
   getch();
   clrscr();
}
```


Q.41 Give the structure definition to represent doubly linked list to store numbers. Compare doubly linked list with singly linked list.

[SPPU : Dec-06, Marks 4]

OR Compare singly linked list with doubly linked list.

[SPPU : May-10, Marks 6, Dec.-10, Marks 2, May-13, Marks 6]

Ans. : Structure definition of doubly linked list: Refer Q.35

Sr. No.	Singly Linked List	Doubly Linked List
1.	Singly linked list is a collection of nodes and each node is having one data field and next link field. For example : Data | Next link	Doubly linked list is a collection of nodes and each node is having one data field, one previous link field and one next link field. For example : Previous link field | Data | Next link field
2.	The elements can be accessed using next link.	The elements can be accessed using both previous link as well as next link.
3.	No extra field is required; hence node takes less memory in SLL.	One field is required to store previous link hence node takes more memory in DLL.
4.	Less efficient access to elements.	More efficient access to elements.

1.14 : Circular Linked List

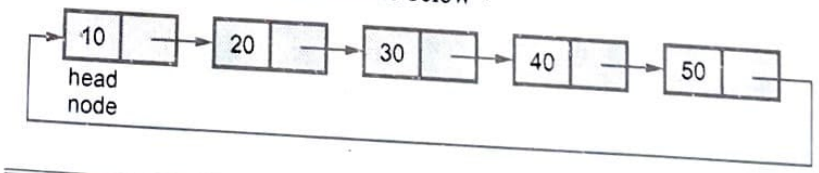
Q.42 What is circular linked list ?

OR Explain with suitable example circular linked list

[SPPU : Dec.-16, Marks 3, May-19, Marks 3]

Ans. : Circular linked list is a linked list in which the next pointer of the last node points to its head node or start node.

The circular linked list is as shown below -



Q.43 Write a C function for creation and display of circular linked list.

Ans. : 1. Creation of CLL

```
struct node *Create()
{
 char ans;
 int flag=1;
 struct node *head,*New,*temp;
 struct node *get_node();
 clrscr();
 do
New = get_node();
   printf("\n\n\n\tEnter The Element\n");
   scanf("%d",&New->data);
   if(flag==1)/*flag for setting the starting node*/
   {
      head = New;
      New->next=head;
      flag=0;/*reset flag*/
   }
   else          /* find last node in list */
   {
    temp=head;
    while (temp->next != head)/*finding the last node*/
       temp=temp->next; /*temp is a last node*/
    temp->next=New;/*attaching the  node*/
    New->next=head; /*each time making the list circular*/
   }
   printf("\n Do you want to enter more nodes?");
   ans=getch();
 }while(ans=='y'||ans=='Y');
 return head;
}
/* function used while allocating memory for a node*/
struct node *get_node()
```

Creating a single node in linked list


```
{
 struct node *New;
 New =(node *) malloc(sizeof(struct node));
 New->next = NULL;
 return(New);/*created node is returned to calling function*/
}
```

2. Display of CLL

```
void Display(struct node *head)
{
    struct node *temp;
    temp = head;
    if(temp == NULL)
     printf("\n Sorry ,The List Is Empty\n");
    else
    {
     do
     {
        printf("%d\t",temp->data);
        temp = temp->next;
     }while(temp != head);
    }
    getch();
}
```

The next node of last node is head node

Q.44 Write advantages of circular singly linked list over a linear linked list.

[SPPU : May-11, Marks 4, May-14, Marks 2 May-17, Marks 3, May-18, Marks 2]

Ans. : In circular linked list the next pointer of the last node points to the head node. Hence we can move from last node to head node of the list efficiently. Thus accessing any node in circular linked list is faster than that of singly linked list.

Q.45 Explain the insertion of node in circular linked list : 1) Start of the list 2) The end of the list 3) After the position

[SPPU : May-09, Marks 8, Dec.-18, Marks 4]

Ans. : 1) Start of the List

```
struct node *insert_head(struct node *head)
```

```
{
 struct node *get_node();
 struct node *New,*temp;
 New=get_node();
 printf("\n Enter The element which you want to insert ");
 scanf("%d",&New->data);
 if(head==NULL)
    head=New;
 else
 {
       temp=head;
       while(temp->next!=head)
          temp=temp->next;
       temp->next=New;
       New->next=head;
       head=New;
       printf("\n The node is inserted!");
 }
 return head;
}
```

2) The end of the List

```
void insert_last(struct node *head)
{
   struct node *New,*temp;
   New=get_node();
   printf("\n Enter The element which you want to insert ");
   scanf("%d",&New->data);
   if(head==NULL)
      head=New;
   else
   {
    temp=head;
    while(temp->next!=head)
          temp=temp->next;
```



```
    temp->next=New;
    New->next=head;
    printf("\n The node is inserted!");
   }
}
```

3) After the position

```
void insert_after(struct node *head)
{
 int key;
 struct node *New,*temp;
 New= get_node();
 printf("\n Enter The element which you want to insert ");
 scanf("%d",&New->data);
 if(head==NULL)
 {
   head=New;
 }
 else
 {
   printf("\n Enter The element after which you want to
   insert the node ");
   scanf("%d",&key);
   temp=head;
   do
   {
     if(temp->data==key)
     {
        New->next=temp->next;
        temp->next=New;
        printf("\n The node is inserted");
        return;
     }
     else
        temp=temp->next;
  }while(temp!=head);
```

```
    }
}
```

Q.46 List the applications of circular lists

☞[SPPU : May-16, Marks 5, May-17, Marks 4]

Ans. : 1) The circular list is used to create the linked lists in which the last node points to the first node.

2) In round robin method the circular linked list is used.

3) The circular linked list is used in solving Josephus problem.

4) For representing the polynomial the circular linked list.

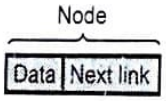
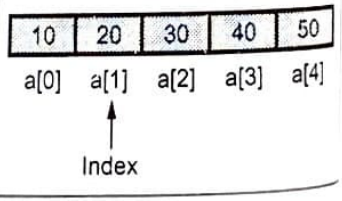
5) For representing the generalized linked list, the circular linked list is used.

1.15 : Linked List Vs. Arrays

Q.47 Compare array with linked list

☞[SPPU : Dec.-10, Marks 4, May-13, Marks 6 May-17, Marks 6, Dec.-17, Marks 4, Dec.-19, Marks 3]

Ans. :

Sr. No.	Linked list	Array
1.	The linked list is a collection of nodes and each node is having one data field and next link field. For example : 	The array is a collection of similar types of data elements. In arrays the data is always stored at some index of the array. For example : 
2.	Any element can be accessed by sequential access only.	Any element can be accessed randomly i.e. with the help of index of the array.

3.	Physically the data can be deleted.	Only logical deletion of the data is possible.
4.	Insertions and deletion of data is easy	Insertions and deletion of data is difficult.
5.	Memory allocation is dynamic. Hence developer can allocate as well as deallocate the memory. And so no wastage of memory is there.	The memory allocation is static. Hence once the fixed amount of size is declared then that much memory is allocated. Therefore there is a chance of either memory wastage or memory shortage.

Q.48 Compare linked list with arrays with reference to following aspects :

i) Accessing any element randomly

ii) Insertion and deletion of an element

iii) Utilization of computer memory **☞[May-12, Dec.-15, Marks 6, May-18, Marks 6, Dec.-19, Marks 6]**

Ans. : i) Accessing any element randomly : Using array, any element can be accessed easily.

a) It takes constant time.

b) In linked list, we can access elements in sequence. So it is very difficult and time consuming to access a element randomly.

ii) Insertion and deletion of an element :

a) Insertion and deletion of elements is time consuming.

b) In linked list, we can insert and delete elements easily and it is less time consuming.

iii) Utiiization of computer memory :

a) Array requires contiguous memory. First we have to declare array and compiler allocates memory at declaration.

b) If we utilize less memory than allocated, then unutilized memory is unnecessarily reserved for array. It can't be used by other programs.

c) If we want to add more elements than the actual size of array, it is not possible.

d) In linked list, memory is utilized efficiently. Memory is not pre-allocated like static data structure. Memory is allocated as per the need. Memory is deallocated when it is no longer needed.

1.16 : Applications of Linked List

Q.49 Enlist the applications of linked list.

Ans. : Various applications of linked list are -

1. Representation of polynomial and performig various operations such as addition, multiplication and evaluation on it.
2. Performing addition of long positive integers.
3. Representing non-integer and non-homogeneous list.

1.17 : Set Operation

Q.50 What is set ?

Ans. : Set is defined as **collection of objects**. These objects are called **elements of the set**. All the elements are enclosed within curly brackets '{' and '}' and every element is separated by commas. If 'a' is an element of set A then we say that $a \in A$ and if 'a' is not an element of A then we say that $a \notin A$.

Typically set is denoted by a **capital letter**.

Q.51 Explain various operations of set.

Ans. : Various operations that can be carried out on set are -

i) Union

ii) Intersection

iii) Difference

iv) Complement

i) $A \cup B$ is union operation - If

$A = \{1, 2, 3\}$ $B = \{1, 2, 4\}$ then

$A \cup B = \{1, 2, 3, 4\}$ i.e. combination of both the sets.

ii) $A \cap B$ is intersection operation - If

$A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then

$A \cap B = \{2, 3\}$ i.e. collection of common elements from both the sets.

iii) $A - B$ is the difference operation - If

$A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$ then

$A - B = \{1\}$ i.e. elements which are there in set A but not in set B.

iv) $\overline{A}$ is a complement operation - If

$\overline{A} = U - A$ where U is a universal set.

For example :

if $U = \{10, 20, 30, 40, 50\}$

$A = \{10, 20\}$

then $\overline{A} = U - A$

$= \{30, 40, 50\}$

C++ Functions for Set operations

1. Union Operation

```
void Union(int a[],int b[],int term1,int term2)
{
int c[MAX];
int i,j,k,n,flag;
k=0;
flag=0;
for(i=0;i<3;i++)
{
   c[k++]=a[i];
}
for(j=0;j<3;j++)
{
   flag=0;
   for(i=0;i<3;i++)
   {
      if(a[i]==b[j])
```

```
            flag=1;
        }
        if(flag==0)
            c[k++]=b[j];
    }
    n=k;
    cout<<"\n The A union B={";
    for(k=0;k<n;k++)
        cout<<" "<<c[k];
    cout<<" }";
}
```

2. Intersection Operation

```
void Intersection(int a[MAX],int b[MAX],int term1,int term2)
{
int c[MAX];
int i,j,k,n,flag;
k=0;
flag=0;
for(i=0;i<3;i++)
{
   flag=1;
   for(j=0;j<3;j++)
   {
      if(a[i]==b[j])
         flag=0;
   }
   if(flag==0)
      c[k++]=a[i];
}
n=k;
cout<<"\n The A intersection B={";
for(k=0;k<n;k++)
   cout<<" "<<c[k];
cout<<" }";
}
```


3. Difference

```
void difference(int a[],int b[],int term1,int term2)
{
 int c[MAX];
 int i,j,k,n,flag;
 k=0;
 flag=0;
 for(i=0;i<3;i++)
 {
    flag=1;
    for(j=0;j<3;j++)
    {
       if(a[i]==b[j])
       {
          flag=0;
          break;
       }
    }
    if(flag==1)
       c[k++]=a[i];
 }
 n=k;
 cout<<"\n The A difference B={";
 for(k=0;k<n;k++)
   cout<<c[k];
 cout<<" }";
}
```

4. Symmetric Difference

```
void symm_difference(int a[MAX],int b[MAX],int term1,int term2)
{
int c[MAX],d[MAX],e[MAX];
int i,j,k,n,m,p,flag;
```

```
int term3,term4;
++ k=0;  //finding union
     //of A and B
flag=0;
for(i=0;i<3;i++)
{
  c[k++]=a[i];
}
for(j=0;j<3;j++)
{
  flag=0;

  for(i=0;i<3;i++)
  {
     if(a[i]==b[j])
        flag=1;
  }
  if(flag==0)
     c[k++]=b[j];
}
term3=k;
m=0;          //finding intersection of A and B
flag=0;
for(i=0;i<3;i++)
{
  flag=1;
  for(j=0;j<3;j++)
  {
     if(a[i]==b[j])
           flag=0;
  }
  if(flag==0)
```

Symmetric difference is a set elements which are present in one set and is not present in another set.


```
     d[m++]=a[i];
}
term4=m;
p=0;//finding difference of A and B
flag=0;
for(i=0;i<term3;i++)
{
  flag=1;
  for(j=0;j<term4;j++)
  {
     if(c[i]==d[j])
     {
        flag=0;
        break;
     }
  }
  if(flag==1)
     e[p++]=c[i];
}
n=p;
cout<<"\n The Symmetric difference is={";
for(i=0;i<n;i++)
  cout<<e[i];
 printf(" }");
}
```

1.18 : String Operation

Q.52 What is string? Explain various built in functionalities of string.

Ans. : String is nothing but collection of characters. That means an array of characters can hold a string and we can perform various operations on such a string.

- For example : we can find the length of a string, we can compare or concatenate two strings, we can reverse a given string and so on.
- In ANSI C++ use of strings is possible not simply by array of characters but there is a special class called **string** is available using which string can be manipulated.

Various functionalities provided by string class are as given below -

Functions	Purpose
append	Appending one string to another.
at	For getting the character at specific location.
begin	Getting the starting location of string.
compare	Comparing two strings.
empty	If string is empty then it returns true otherwise false.
end	Returning the ending location of string.
erase	For removing the specified character.
find	For searching the occurrence of specific substring.
insert	For inserting the character at specific location.
length	Returns the length of a string.
replace	Replaces the specific character.
size	Gives the size of the string.
swap	For swapping the two strings.

Q.53 Write a C++ program to sort the given strings alphabetically.

Ans. :

```
#include<iostream>
#include<string>
using namespace std;
class STRING
{
    string str[10];
```



```
    int n;
public:
    void getdata()
    {

        cout<<"\n How many strings do you want?";
        cin>>n;
        cout<<"\n Enter the strings ";
        for(int i=0;i<n;i++)
        {
            cin>>str[i];
        }
    }
    void display()
    {
        cout<<"\n The strings are:";
        for(int i=0;i<n;i++)
        {
            cout<<"\n"<<str[i];
        }
    }
    void sort()
    {
        string temp;
        for(int i=0;i<n;i++)
        {
            for(int j=i+1;j<n;j++)
            {
             if(str[i]>str[j])
             {
              temp=str[i];
              str[i]=str[j];
              str[j]=temp;
             }
            }
```

```
        }
    }
};
void main()
{
    STRING obj;
    obj.getdata();
    obj.display();
    obj.sort();
    obj.display();
}
```

Output

```
How many strings do you want?3
Enter the strings BBB
CCC
AAA

The strings are:
BBB
CCC
AAA

The strings are:
AAA
BBB
CCC
```

END..

Unit - II

2 Searching and Sorting

2.1 : Need of Searching and Sorting

Important Points to Remember

- Searching is a technique used for locating the position of the required element from heap of data.
- Sorting is useful for arranging the data in desired order.

Q.1 Discuss the importance of sorting and searching in computer application

☞[SPPU : Dec.-06, Marks 4]

OR Write short note on - "Application of sorting"

☞[SPPU : Dec.-08, Marks 4, Dec.-09, Marks 6]

Ans. : Various applications of sorting are : 1) The sorting is useful for arranging data in desired order. After sorting the required element can be located very easily.

2) The sorting is useful in database applications for arranging the data in desired order.
3) In the applications like dictionary, the data is arranged in sorted order.
4) For searching the element from list of elements, the sorting is required.
5) For checking the uniqueness of the element the sorting is required.
6) For finding the closest pair from the list of elements the sorting is required

2.2 : Concept of Internal and External Sorting

Q.2 Explain the following terms :

i) Internal sorting ii) External sorting ☞[SPPU : Dec.-14, Marks 4,

☞May-17, Dec.-17, Marks 6, Dec.-18, Marks 3]

Ans. : Internal Sorting : ∘ The internal sorting is a sorting in which the data resides in the main memory of the computer.

- ∘ Various methods that make use of internal sorting are -
  1. Bubble Sort
  2. Insertion Sort
  3. Selection Sort
  4. Quick Sort
  5. Radix Sort and so on.

External Sorting :

- ∘ For many applications it is not possible to store the entire data on the main memory for two reasons, i) amount of **main memory available is smaller** than amount of data. ii)Secondly the **main memory is a volatile device** and thus will lost the data when the power is shut down. To overcome these problems the data is stored on the secondary storage devices. The technique which is used to sort the data which resides on the secondary storage devices are called **external sorting.**
- ∘ The data stored on secondary memory is part by part loaded into main memory, sorting can be done over there. The sorted data can be then stored in the intermediate files. Finally these intermediate files can be merged repeatedly to get sorted data. Thus **huge amount of data** can be sorted using this technique.

For example : Consider that there are 10,000 records that has to be sorted. Clearly we need to apply external sorting method. Suppose main memory has a capacity to store 500 records in blocks, with each block size of 100 records.

The sorted 5 blocks (i.e. 500 records) are stored in intermediate file. This process will be repeated 20 times to get all the records sorted in chunks.

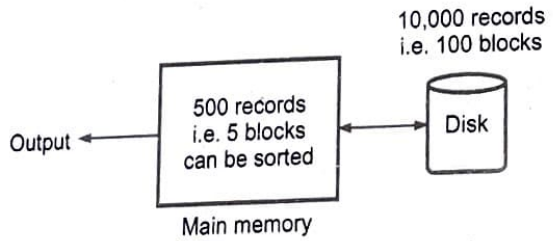


Fig. Q.2.1

In the second step, we start merging a pair of intermediate files in the main memory to get output file.

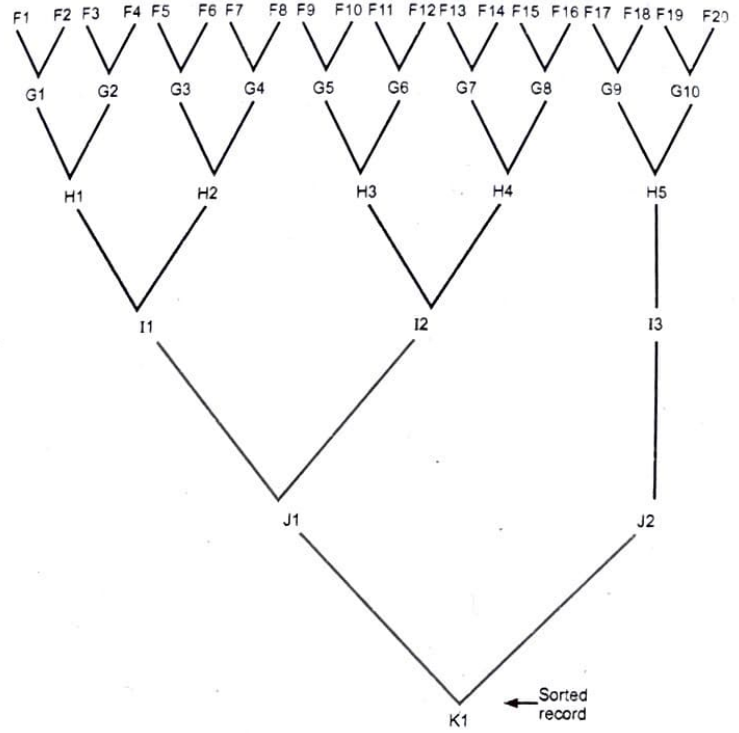


Fig. Q.2.2

2.3 : Sort Stability

Q.3 Explain the following terms with respect to sorting
i) Sorting stability ii) Efficiency iii) Passes

☞[SPPU : May-09, Marks 6]

Ans. : i) Sort stability : The sorting stability means comparing the records of same value and expecting them in the same order even after sorting them.

ii) Efficiency : Efficiency means how much time is required by the particular sorting algorithm to sort the given list of elements. It is usually denoted in terms of time complexity. The time complexities are given in terms of big-on notations.

Commonly there are $O(n^2)$ and O(nlogn) time complexities for various algorithms. The sorting techniques such as bubble sort, insertion sort, selection sort, shell sort has the time complexity $O(n^2)$ and the techniques such as merge sort, quick sort has the time complexity as O(nlogn).

iii) Passes : While sorting the elements in some specific order there is lot of arrangement of elements. The phases in which the elements are moving to acquire their proper position is called **passes.**

For example : 10, 30, 20, 50, 40

Pass 1 : 10, 20, 30, 50, 40

Pass 2 : 10, 20, 30, 40, 50

In the above method we can see that data is getting sorted in two passes distinctly. By applying a logic as comparison of each element with its adjacent element gives us the result in two passes.

Q.4 Explain the following terms : Sort stability.

☞[SPPU : Dec.-14, Marks 2]

Ans. : Refer Q.3(i)

Q.5 What is sort stability and efficiency ?

☞[SPPU : May-16, Marks 2]

Ans. : Refer Q.3(i) and (ii)

2.4 : Searching Methods

Important Points to Remember

- When we want to find out particular record efficiently from a given list of elements then there are various methods of searching the elements. These methods are called **searching methods.**
- Two commonly used algorithms for searching methods are – linear or sequential search and binary search.
- The basic characteristics of any searching algorithm is
  1. It should be **efficient.**
  2. **Less number of computations** must be involved in it.
  3. The **space** occupied by searching algorithm must be **less.**

2.5 : Linear Search

Q.6 Explain sequential searching with suitable example.

Ans. : Sequential search is technique in which the given list of elem is scanned from the beginning. The key element is compared with ev.y element of the list. If the match is found the searching is stopped otherwise it will be continued to the end of the list.

Example

Array

	Roll no	Name	Marks
0	15	Parth	96
1	2	Anand	40
2	13	Lalita	81
3	1	Madhav	50
4	12	Arun	78
5	3	Jaya	94

Fig. Q.6.1 Represents students Database for sequential search

From the above Fig. Q.6.1 the array is maintained to store the students record. The record is not sorted at all. If we want to search the student's record whose roll number is 12 then with the key-roll number we will see the every record whether it is of roll number = 12. We can obtain such a record at Array [4] location.

Q.7 Write a C function for sequential search.

Ans. :

```
search(int k)
{
 for(i=0;i<n;i++) // Comparing each element of array
 {
  if(a[i]==k) // with key value
  return 1;
 }
return 0;
}
```

2.6 : Binary Search

Q.8 Search a given number using binary search, show all passes. Number to be searched is 6.

[SPPU : May-10, Marks 6]

1,4,9,13,23,34

Ans. :

Pass 1 : Consider that the elements are stored in an array

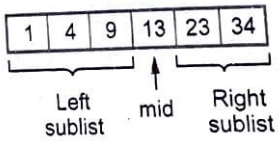
0	1	2	3	4	5
1	4	9	13	23	34

Divide the list in two equal parts by separating middle element.

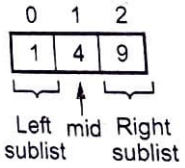
A [3] is mid element. The key = 6.

Compare A [3] i.e. 13 and key i.e. 6.

As 6 < A [3]. Search in left sublist



Pass 2 :

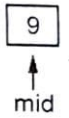


Key = 6, A[mid] = 4

Key > A[mid] ∴ Search Right sublist

Pass 3 :

Key = 6, A[mid] = 4



There is single element and ≠ key.

Hence **"Element is not present in the given list".**

Q.9 Write a pseudo C routine (recursive) binary search(A,N,X) to search for a given number X in an array A with N sorted numbers. Compare it with iterative counterpart.

[SPPU : Dec.-06, 08, 18, Marks 6]

Ans. : i) Recursive Routine :

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

ii) Non Recursive Routine :

```
int BinSearch(int A[SIZE],int KEY)
{
  int low,high,m;
  low=0;
  high=n-1;
  while(low<=high)
  {
        m=(low+high)/2;   //mid of the array is obtained
        if(KEY==A[m])
            return m;
```

```
    else if(KEY<A[m])
        high=m-1;        //search the left sub list
    else
        low=m+1;         //search the right sub list
  }
  return-1;              //if element is not present in the list
}
```

Q.10 Write a C function for iterative binary search to search a given number in an array. **☞[SPPU : May-09, Marks 6]**

Ans. : Refer Q.9(ii)

Q.11 Compare linear and binary search.

☞[SPPU : May-14, Marks 2, Dec.-17, 19, Marks 3]

Ans. :

Sr. No.	Linear search method	Binary search method
1.	The linear search is a searching method in which the element is searched by scanning the entire list from first element to the last.	The binary search is a searching method in which the list is sub divided into two sub-lists. The middle element is then compared with the key element and then accordingly either left or right sub-list is searched.
2.	Many times entire list is searched.	Only sub-list is searched for searching the key element.
3.	It is simple to implement.	It involves computation for finding the middle element.
4.	It is less efficient searching method.	It is an efficient searching method.

Q.12 Explain linear and binary search techniques with examples

☞[SPPU : May-16, Marks]

Ans. : Refer Q.6 and Q.8

2.7 : Fibonacci Search

Q.13 Explain Finonacci Search with suitable example.

Ans. :

In Fibonacci search rather than considering the mid element, we consider the indices as the numbers from fibonacci series. As we know, the Fibonacci series is -

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

To understand how Fibonacci search works, we will consider one example, suppose, following is the list of elements.

arr [ ]

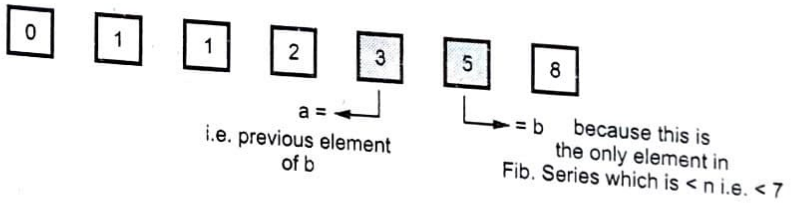
10	20	30	40	50	60	70
1	2	3	4	5	6	7

Here n = total number of elements = 7

We will always compute 3 variables i.e. a, b and f.

Initially f = n = 7.

For setting a and b variables we will consider elements from Fibonacci series.



Now we have

f = 7

b = 5

a = 3

With these initial values we will start searching the key element from the list. Each time we will compare key element with arr [f]. That means

If (Key < arr [f])

f = f – a

b = a

a = b – a

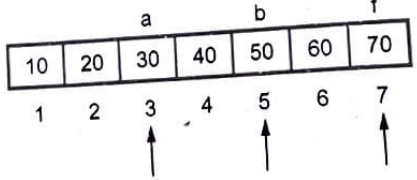
If (Key > arr [f])

f = f + a

b = b – a

a = a – b

Suppose we have **f = 7, b = 5, a = 3**



If **Key = 20**

i.e. Key < arr[f]

i.e. 20 < 70

∴f = f – a = 7 – 3 = 4

b = a = 3

a = b – a = 2

Again we compare

if (Key < arr [f])

i.e. 20 < arr [4]

i.e. if (20 < 40) → Yes

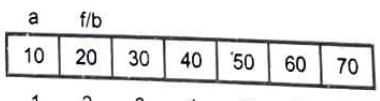
At Present f = 4, b = 3, a = 2

∴f = f – a = 4 – 2 = 2

b = a = 2

a = b – a = 3 – 2 = 1

Now we get f = 2, b = 2, a = 1



If **Key = 60**

i.e. Key < arr [f]

60 < 70

∴f = f – a = 7 – 3 = 4

b = a = 3

a = b – a = 2

Again we compare

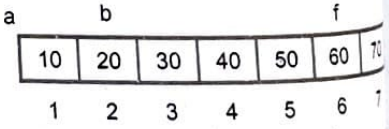
if (Key > arr [f])

∴i.e. 60 > arr [f] i.e. 40

∴f = f + a = 4 + 2 = 6

b = b – a = 3 – 2 = 1

a = a – b = 2 – 3 = – 1



If (Key < arr [f])

i.e. if (60 < 60) → No

If (Key < arr [f]) i.e.

if (20 < 20) → No

If Key > arr [f]) i.e.

if (20 > 20) → No

That means

"Element is present at f = 2 location"

If (key > arr [f])

i.e. if (60 > 60) → No

That means

"Element is present at f = 6 location."

Analysis : The time complexity of fibonacci search is **O(logn).**

Algorithm

Let the length of given array be **n [0...n-1]** and the element to be searched be **key**

Then we use the following steps to find the element with minimum steps :

1. Find the **smallest Fibonacci number greater than or equal to n.** Let this number be **f** (m^{th} element).
2. Let the two Fibonacci numbers preceding it be **a** ($(m-1)^{th}$ element) and
   b ($(m-2)^{th}$ element)

While the array has elements to be checked :

Compare key with the last element of the range covered by **b**

(a) If **key** matches, return index value.

(b) Else if **key is less** than the element, move the third Fibonacci variable two Fibonacci down, indicating removal of approximately two-third of the unsearched array.

(c) Else key is greater than the element, move the third Fibonacci variable one Fibonacci down. Reset offset to index. Together this results into removal of approximately front one-third of the unsearched array.

3. Since there might be a single element remaining for comparison, check if **a** is '1'. If Yes, compare key with that remaining element. If match, return index value.

2.8 : Bubble Sort

Q.14 Write algorithm and C function for bubble sort method.

Ans. : Algorithm : 1. Read the total number of elements say n

2. Store the elements in the array
3. Set the i = 0 .
4. Compare the adjacent elements.
5. Repeat step 4 for all n elements.
6. Increment the value of i by 1 and repeat step 4, 5 for i< n
7. Print the sorted list of elements.
8. Stop.

'C' Function

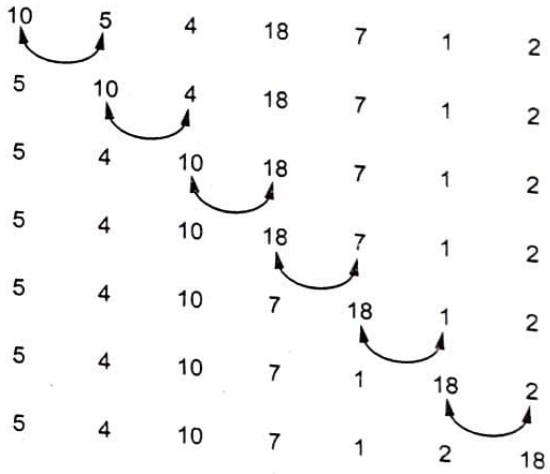
```
void bubblesort (int a[20], int n)
{
    int i, j, m,temp;
    for (i = 0; i < n – 1; i ++)
    {
        for (j= 0; j < n; j ++)
        {
            if (a[j] > a [j + 1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j + 1] = temp;
            }
        }
    }
```

Q.15 Write a pseudo C code to sort a list of integers using bubble sort. Show the output of each pass for the following list 10, 5, 4, 18, 17, 1, 2. **[SPPU : Dec.-11, Marks 8, Dec.-17, Marks**

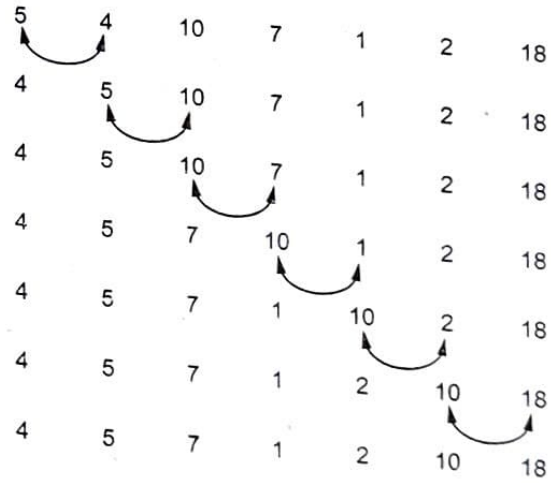
Ans. : For pseudo C code : Refer Q.14.

Let, 10, 5, 4, 18, 17, 1, 2 be the given list of elements. We will compare adjacent elements say A[i] and A[j]. If A[i] > A[j] then swap the elements.

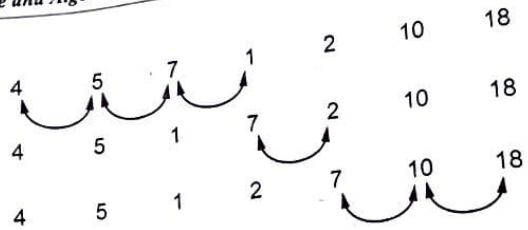
Pass I



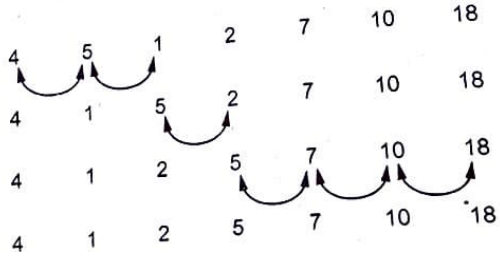
Pass II



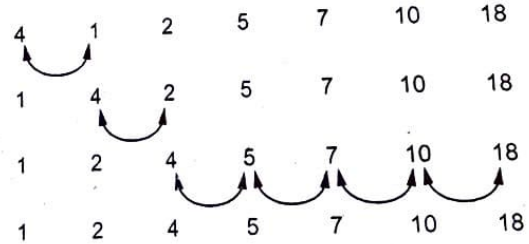
Pass III



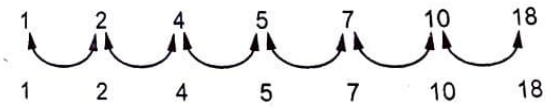
Pass IV



Pass V



Pass VI



This is the sorted list of elements.

Q.16 What is bubble sort ? Explain with example.

[SPPU : Dec.-09, May-12, Marks

Ans. : Refer Q.15

2.9 : Insertion Sort

Q.17 What are advantages of sorting data? Explain insertion sort with example

[SPPU : May-11, Marks 6]

Ans. : Due to sorting the elements are arranged either in ascending or descending order.

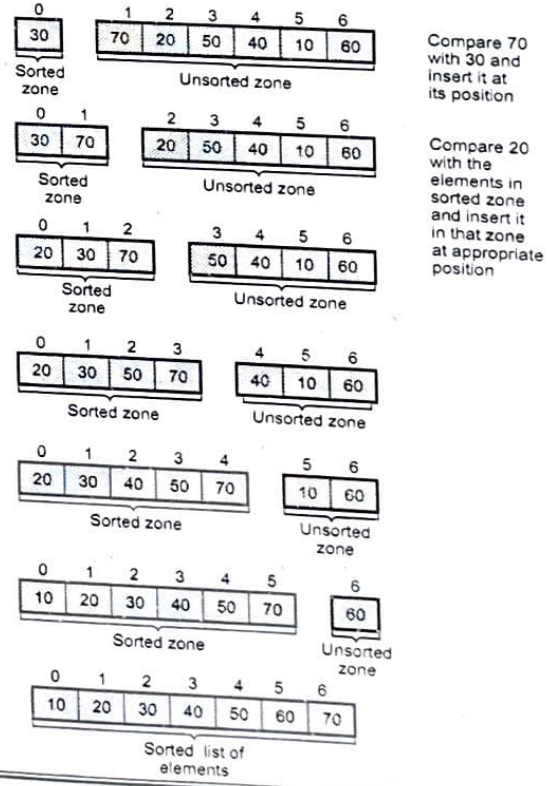
In this method the elements are inserted at their appropriate place. Hence is the name **insertion sort**. Let us understand this method with the help of some example -

For Example

Consider a list of elements as,

0	1	2	3	4	5	6
30	70	20	50	40	10	60

The process starts with first element



Q.18 Write a C function to sort N input numbers using insertion sort

[SPPU : Dec.-05, Marks 8]

Ans. : C Function : void Insert_sort(int A[10],int n)

```
{
 int i,j,temp;
 for(i=1;i<=n-1;i++)
 {
        temp=A[i];
        j=i-1;
        while((j>=0)&&(A[j]>temp))
        {
         A[j+1]=A[j];
         j=j-1;
        }
        A[j+1]=temp;
 }
 printf("\n The sorted list of elements is...\n");
 for(i=0;i<n;i++)
        printf("\n%d",A[i]);
}
```

2.10 : Quick Sort

Q.19 What is the importance of pivot element in the quick sort method ?

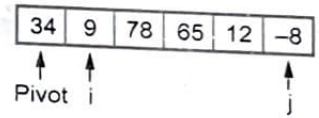
[SPPU : Dec.-15, May-18, Marks 2]

Ans. : In quick sort method the sorting procedure is carried out with reference to pivot element. The elements less than pivot form a left sublist and the elements greater than pivot form a right sublist. Thus quick sort performs partitioning of the lists using pivot element.

Q.20 Sort using quick sort the following and show all passes.
34, 9, 78, 65, 12, –8

[SPPU : May-10, Marks 10]

Ans. : We will store the elements in array A. We will use following rules for sorting the elements using quick sort.



Rule 1 : Select the first element of list or sublist as **Pivot element.** Mark the adjacent element to pivot as i and last element of the list as j. elements. It is as shown below.

Rule 2 : If A[i] < A [Pivot] then increment i

Rule 3 : If A[j] > A [Pivot] then decrement j

Rule 4 : If i and j can not move, then swap A[i] and A [j]

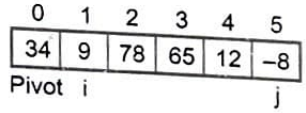
Rule 5 : When j ≤ i just swap A [j] and A [Pivot]. Due to this the pivot element gets placed at its proper position and we get two sublists –

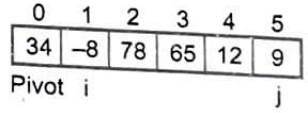
i) the left sublist with all the elements less than pivot element.

ii) the right sublist with all the elements greater than pivot element.

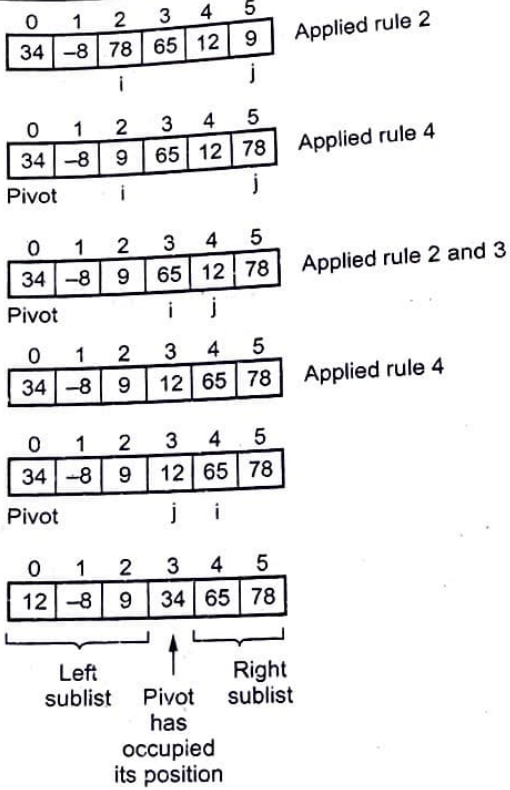
Repeat above rules until we get the sorted list.

Step 1 :



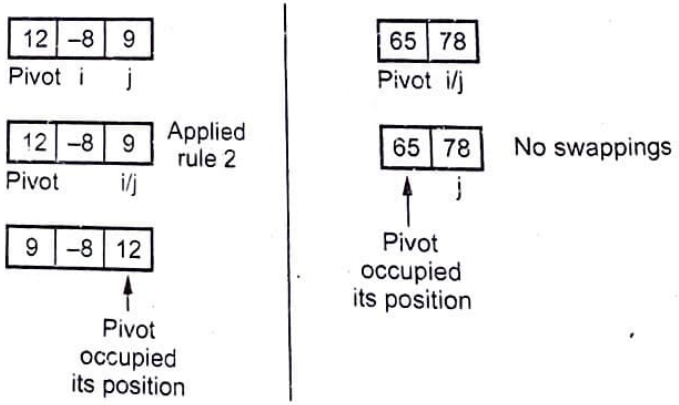


Applied rule 4

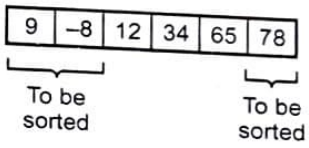


After Pass 1 we get above list.

Step 2 : We will sort two sublists independently.

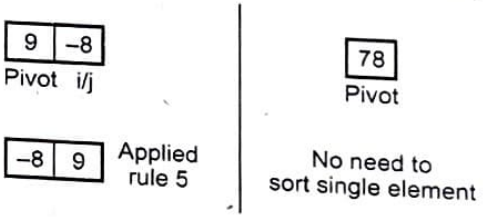


After combining above lists we get

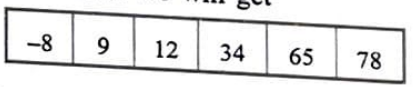


After pass 2 we will get above list.

Step 3 : Now we will sort unsorted sublists independently.

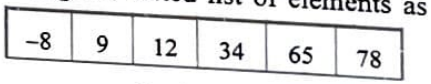


After combining above sublists we will get



After Pass 3 we will get above list

Step 4 : Finally we will get a sorted list of elements as



Sorted List

2.11 : Merge Sort

Q.21 Write a non recursive pseudo C routine to sort the numbers using merge sort. Show all passes to sort the values using merge sort in ascending order :
21, 5,7,11,35,76,0,9,27,45

Ans. : **☞[SPPU : May-12, Marks 8]**

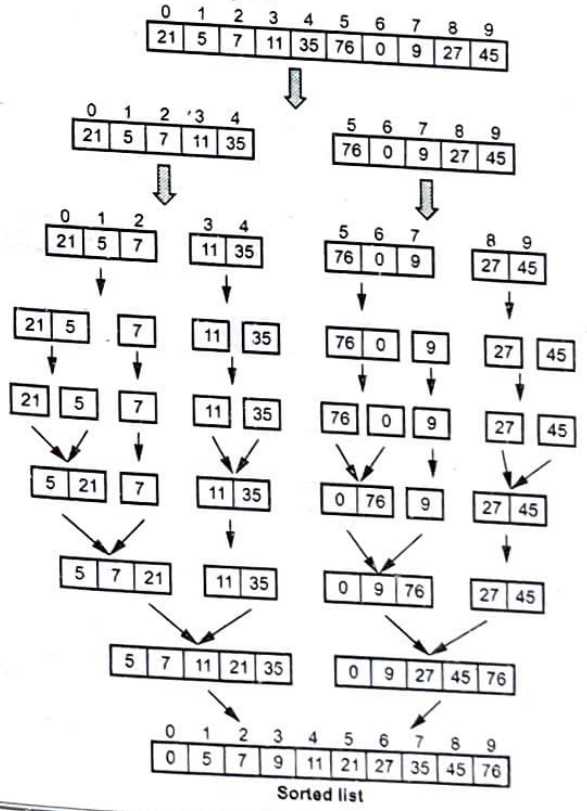
```
/* This non recursive function is to split the list into sublists */
void MergeSort(int A[10],int n)
{
   int mid,high;
```

```
    int low,size;
    void Combine(int A[10],int low,int mid,int high);
    for(size=1;size<n;size++)
    {
        for(low=0;low<n-1;low+=2*size)
        {
            mid=low+size-1;
            high=min(low+2*size-1,n-1);
            Combine(A,low,mid,high);//merging of two sublists
        }
    }
}
/* This function is for merging the two sublists */
void Combine(int A[10],int low,int mid,int high)
{
    int i,j,k;
    int temp[10];
    k=low;
    i=low;
    j=mid+1;
    while(i <= mid && j <= high)
    {
        if(A[i]<=A[j])
        {
            temp[k]=A[i];
            i++;
            k++;
        }
        else
        {
            temp[k]=A[j];
            j++;
            k++;
        }
    }
    while(i<=mid)
    {
```



```
temp[k]=A[i];
        i++;
        k++;
    }
    while(j<=high)
    {
        temp[k]=A[j];
        j++;
        k++;
    }
    //copy the elements from temp array to A
    for(k=low;k<=high;k++)
        A[k]=temp[k];
}
```

Consider the elements in an array as



Q.22 Sort the following elements using merge sort.
10, 5, 7, 6, 1, 4, 8, 3, 2, 9.

Ans. :

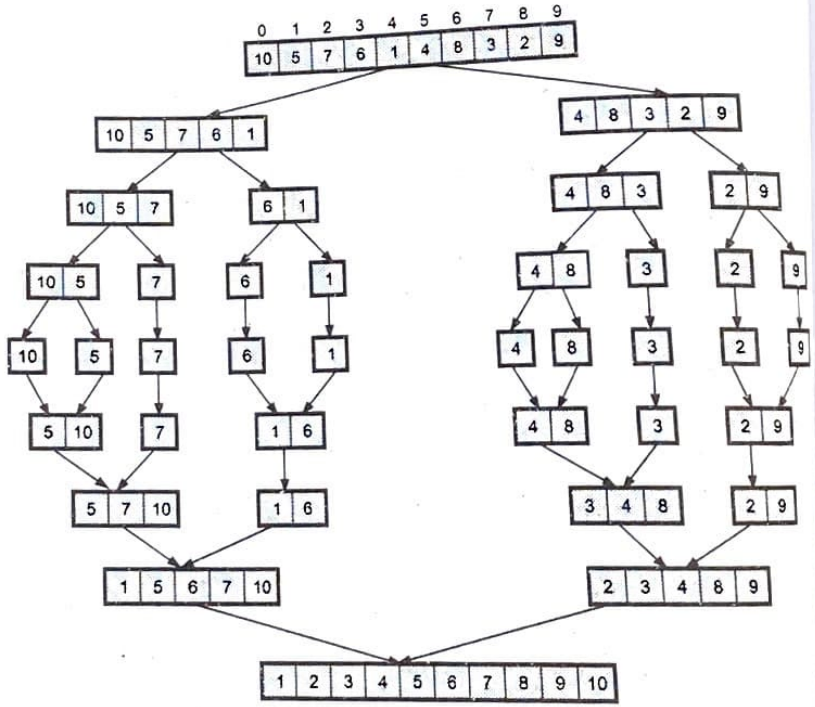


Fig. Q.22.1

2.12 : Shell Sort

Q.23 Explain Shell sort technique with suitable example.

Ans. : This method is a improvement over the simple insertion sort. this method the elements at fixed distance are compared. The distan will then be decremented by some fixed amount and again the compariso will be made. Finally, individual elements will be compared. Let us t some example.

Example : If the original file is

	0	1	2	3	4	5	6	7
X array	25	57	48	37	12	92	86	33

Step 1 : Let us take the distance k = 5

So in the first iteration compare

(x [0], x[5])

(x [1], x [(6])

(x [2], x[7])

(x[3])

(x[4])

i.e. first iteration

After first iteration,

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
25	57	33	37	12	92	86	48

Step 2 : Initially k was 5. Take some d and decrement k by d. Let us take d = 2

$\therefore$ k = k – d i.e. k = 5 – 2 = 3

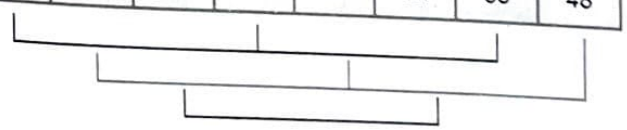
So now compare

(x[0], x[3], x[6]), (x[1], x[4], x[7])

(x[2], x[5])

Second iteration

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
25	57	33	37	12	92	86	48



After second iteration

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
25	12	33	37	48	92	86	57

Step 3 : Now k = k – d ∴k = 3 – 2 = 1

So now compare

(x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7])

This sorting is then done by simple insertion sort. Because simple insertion sort is highly efficient on sorted file. So we get

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
12	25	33	37	48	57	86	92

2.13 : Comparison of All Sorting Methods

Q.24 Give the best case, worst case and average case analysis of any five sorting methods.

Ans. : Comparison of all Sorting Methods

Sorting technique	Best case	Average case	Worst case
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$
Quick sort	O(nlogn)	O(nlogn)	$O(n^2)$
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)
Shell sort	O(nlogn)	O(n)	O(n)

2.14 : Analysis of Insertion Sort, Quick sort, Binary search,Hashing

Q.25 Write pseudo C code for quick sort and write average and worst case time complexity **☞[SPPU : Dec.-10, Marks 10]**

OR Write pseudo C algorithm for quick sort.

☞[SPPU : May-16, Marks 6]

Ans. :

```
Algorithm Quick(A[0...n-1],low,high)
//Problem Description : This algorithm performs sorting of
//the elements given in Array A[0...n-1]
//Input: An array A[0...n-1] in which unsorted elements are
//given. The low indicates the leftmost element in the list
//and high indicates the rightmost element in the list
//Output: Creates a sub array which is sorted in ascending
//order
if(low<high)then
//split the array into two sub arrays
m ← partition(A[low...high])// m is mid of the array
Quick(A[low...m-1])
Quick(A[mid+1...high] )
```

In above algorithm call to partition algorithm is given. The *partition* performs arrangement of the elements in ascending order. The recursive *quick* routine is for dividing the list in two sub lists. The pseudo code for *Partition* is as given below -

```
Algorithm Partition (A[low...high])
//Problem Description: This algorithm partitions the
//subarray using the first element as pivot element
//Input: A subarray A with low as left most index of the
//array and high as the rightmost index of the array.
//Output: The partitioning of array A is done and pivot
//occupies its proper position. And the rightmost index of
//the list is returned
pivot ← A[low]
i ← low
j ← high+1
while(i<=j) do
{
 while(A[i]<=pivot) do
  i ← i+1
 while(A[j]>=pivot) do
  j ← j-1;
 if(i<=j) then
```

```
    swap(A[i],A[j])//swaps A[i] and A[j]
}
swap(A[low],A[j])//when i crosses j swap A[low] and A[j]
return j//rightmost index of the list
```

The partition function is called to arrange the elements such that all the elements that are less than pivot are at the left side of pivot and all the elements that are greater than pivot are all at the right of pivot.

When pivot is chosen such that the array gets divided at the mid then it gives the best case time complexity. The best case time complexity of quick sort is $O(n\log_2 n)$.

The worst case for quick sort occurs when the pivot is minimum or maximum of all the elements in the list. This can be graphically represented as -

This ultimately results in $O(n^2)$ time complexity. When array elements are randomly distributed then it results in average case time complexity, and it is $O(n\log_2 n)$.

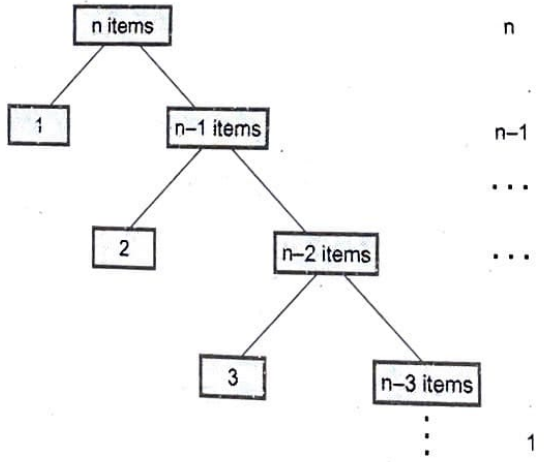


Fig. Q.25.1

Q.26 Write pseudo C code for binary search and analyze its time complexity.

[SPPU : Dec.-13, May-18, Marks 6]

Ans. : Refer Q.9.

Time complexity : In this algorithm, the list is divided into two sublists with respect to middle element. The middle element is compared with key element and then left or right sublist is searched.

The recurrence relation can be written as

$$T(n) = \underbrace{T\left(\frac{n}{2}\right)}_{\text{Time required to compare left sublist or right sublist}} + \underbrace{1}_{\text{One comparison made with mid element}} \quad \ldots (Q.26.1)$$

$$T(n) = 1 \quad \ldots (Q.26.2)$$

We will assume $n = 2^k$, equation (Q.26.1) becomes

$$T(n) = T\left(\frac{2^k}{2}\right) + 1$$

$$\text{i.e. } T(n) = T(2^{k-1}) + 1 \quad \ldots (Q.26.3)$$

By backward substitution method,'

$$T(2^k) = [T(2^{k-1}) + 1] + 1$$

$$\text{i.e. } T(2^k) = T(2^{k-2}) + 2$$

$$T(2^k) = [T(2^{k-3}) + 1] + 2$$

$$\text{i.e. } T(2^k) = T(2^{k-3}) + 3$$

$$T(2^k) = T(2^{k-k}) + k$$

$$T(2^k) = T(1) + k \quad \ldots (Q.26.4)$$

By putting value of equation (Q.26.2) in equation (Q.26.4),

$$T(2^k) = 1 + k \quad \ldots (Q.26.5)$$

But $n = 2^k \therefore k = \log_2 n$. Substituting this value in equation (Q.26.5) we get,

$$T(n) = \log_2 n + 1$$

Thus the time complexity in terms of theta notation is

$$\mathbf{T(n) = \Theta(\log_2 n)}$$

Q.27 Give the time complexity of various operations of various hashing techniques.

Ans. : The time complexity of various operations of various hashing techniques are -

1) Linear Probing

	Worst Case	Average Case	Best Case
Search	O(n)	O(1)	O(1)
Insert	O(n)	O(1)	O(1)
Delete	O(n)	O(1)	O(1)

2) Separate Chaining

	Worst Case	Average Case	Best Case
Search	O(n)	O(1)	O(1)
Insert	O(n)	O(1)	O(1)
Delete	O(n)	O(1)	O(1)

2.15 : Analysis of Selection Sort, Bucket Sort, Radix Sort

Q.28 Write Pseudo C code for selection sort. Analyse its time complexity. Compare selection sort and bubble sort.

[SPPU : May-11, Marks 10]

Ans. :

```
void selection (int A[10]
{
    int i,j,Min,temp;
    for (i=0;i<=n-2;i++)
    {
       Min=i;
       for(j=i+1;j<=n-1;j++)
       {
          if(A[j]<A[Min])
          Min=j;
```



```
       }
       temp=A[i];
       A[i]=A[Min];
       A[Min]=temp;
    }
    printf("\n The sorted List is ...\n");
    for(i=0;i<n;i++)
        printf(" %d,A[i]);
}
```

Analysis for Time complexity : The pseudo code for selection sort shows that sorting can be performed with the help of two for loops.

Hence time complexity can be specified using following equation.

T(n) = Outer for loop × Inner for loop × Comparison of A[i] and A[j]

$$\therefore \quad T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \qquad \ldots (Q.28.1)$$

$$\text{If } = \sum_{i=0}^{n} 1 \text{ then } = n - 0 + 1 = n + 1$$

Using this formula, in equation (Q.28.1) we get,

$$T(n) = \sum_{i=0}^{n-2} (n-1) + (i+1) + 1 = \sum_{i=0}^{n-2} (n-1-i)$$

Simplify above equation and we will get

$$T(n) = \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \text{ Using this formula}$$

$$\sum_{i=0}^{n-2} = \frac{(n-2)(n-2+1)}{2}$$

$$= \frac{(n-2)(n-1)}{2}$$

$$\therefore \quad T(n) = \sum_{i=0}^{n-2} (n-1) - \frac{(n-2)(n-1)}{2} \qquad \ldots (Q.28.2)$$

We will take (n – 1) from equation (Q.28.2) as common factor,

$$T(n) = (n-1)\left(\sum_{i=0}^{n-2} 1\right) - \frac{(n-2)}{2}$$

$$\sum_{i=0}^{n-2} 1 = (n-2) - 0 + 1 = (n-1)$$

$$\therefore \quad T(n) = (n-1)(n-1) - \frac{(n-2)}{2} = (n-1)^2 - \frac{(n-2)}{2}$$

$$= \frac{2(n-1)^2 - (n-2)}{2} = \frac{2(n^2 - 2n + 1) - (n-2)}{2}$$

$$= \frac{2n^2 - 4n + 2 - n + 2}{2} = \frac{2n^2 - 5n + 4}{2} \approx \Theta(n^2)$$

Hence time complexity of selection sort is $\Theta(n^2)$

Comparison between Selection Sort and Bubble Sort

Sr.No.	Selection Sort	Bubble Sort
1.	The average case time complexity is $\Theta(n^2)$.	The average case time complexity is $\Theta(n^2)$.
2.	Selection sort needs less swaps for sorting the list of elements.	Bubble sort needs more swaps for sorting the list of elements.
3.	It is complex to perform.	It is simple to perform.
	It is not a stable sorting algorithm	It is a stable sorting algorithm.

Q.29 Compare the insertion sort and selection sort with i) Efficiency ii) Sort stability iii) Passes

[SPPU : Dec.-06, 07, 09, May-07, 12, Marks 8]

Ans. : i) Efficiency : The code for selection sort is as follows

```
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
{
   if(ai>aj)
      swap ai and aj
}
```


In the i^{th} pass, n-i comparisons will be needed to select the smallest element. n-1 passes are required to sort the array. Thus, the number of comparisons needed to sort an array having n elements

$$= (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}(n^2 - n) \approx O(n^2)$$

Insertion sort :

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

Inner loop is data sensitive. If the input list is presented for sorting is presorted then the test a[i] > y in the inner loop will fail immediately. Thus, only one comparison will be made in each pass.

Thus the total number of comparisons (Best case)

$= n - 1 \approx n$ for large n.

If the numbers to be sorted are initially in descending order then the inner loop will make i iterations in i^{th} pass.

∴ Total number of comparisons.

$$= 1 + 2 + 3 + \ldots + (n-1) = \frac{n(n-1)}{2} = \frac{1}{2}(n^2 - n) \approx n^2 \text{ for large n.}$$

ii) Sort stability : Both the techniques are stable. Both of them have same time complexity.

iii) Passes : Both selection and insertion sort requires n-1 passes.

Insertion sort is more stable than selection sort.

Insertion sort requires less overhead while sorting data. Selection sort requires more number of comparisons. In both the algorithms, elements are read in linear fashion and adjacent elements are compared. Hence identical numbers will maintain their relative sequence even in the sort list.

Q.30 Sort the elements using bucket sort. 56, 12, 84, 56, 28, 0, -13, 47, 94, 31, 12, -2. **[SPPU : May-08]**

Ans. : We will set up an array as follows

0	1	2	3	4	5	6	7	8	9	10

Range	–20 to –1	0 to 10	10 to 20	20 to 30	30 to 40	40 to 50	50 to 60	60 to 70	70 to 80	80 to 90	90 to 100

Now we will fill up each bucket by corresponding elements

0	1	2	3	4	5	6	7	8	9	10
–13	0	12	28	31	47	56			84	94
–2		12				56				

Now sort each bucket

0	1	2	3	4	5	6	7	8	9	10
–13	0	12	28	31	47	56			84	94
–2		12				56				

Print the array by visiting each bucket sequentially.

-13, -2, 0, 12, 12, 28, 31, 47, 56, 56, 84, 94.

This is the sorted list.

Q.31 Write C routine to sort the elements using bucket sort

Ans. :

```
void bucketsort( int a[],int n,int max )
{
 int i,j=0;
 //inialize each bucket 0 and thus indicates bucket to be empty
 int *buckets=calloc(max+1,sizeof(int));
//place Each element from unsorted array in each corresponding
bucket
 for(inti=0;i<n;i++)
   buckets[a[i]]++;
//sort each bucket individually
// Sequentially empty each bucket in some array
for(i=0;i<max;i++)
```



```
while(buckets[i]--)
  b[j++]=i;
//display the array b as sorted list of elements
```

Q.32 Sort the following data in ascending order using Radix Sort : 25, 06, 45, 60, 140, 50,

Ans. :

Step 1 :

Sort the elements according to last digit and sort them.

Last digit	Element
0	50, 60, 140
1	
2	
3	
4	
5	25, 45
6	06
7	
8	
9	

Step 2 :

Sort the elements according to second last digit and sort them.

Second last digit	Element
0	06
1	
2	25
3	

4	45, 140
5	50
6	60
7	
8	

Step 3 :

Sort the elements according to 100^{th} position of the element and sort them.

100^{th} position	Element
0	06, 25, 45, 50, 60
1	140
2	
3	
4	
5	
6	
7	
8	
9	

Thus the sorted list of elements is

06, 25, 45, 50, 60, 140

Algorithm :

1. Read the total number of elements in the array.
2. Store the unsorted elements in the array.
3. Now the simple procedure is to sort the elements by digit by digit.
4. Sort the elements according to the last digit then second last digit so on.

5. Thus the elements should be sorted for up to the most significant bit.
6. Store the sorted element in the array and print them.
7. Stop.

C++ Program

```
#include<iostream>
using namespace std;
int main()
{
   int a[100][100], r = 0, c = 0, i, sz, b[50];
   cout<<"Size of array: ";
   cin>>sz;
   cout<<"\n";
   for (r = 0; r<100; r++)
   {
      for (c = 0; c<100; c++)
         a[r][c] = 1000;
   }
   for (i = 0; i<sz; i++)
   {
      cout<<"Enter element "<<i + 1;
      cin>>b[i];
      r = b[i] / 100;
      c = b[i] % 100;
      a[r][c] = b[i];
   }
   for (r = 0; r<100; r++)
   {
      for (c = 0; c<100; c++)
      {
         for (i = 0; i<sz; i++)
         {
            if (a[r][c] == b[i])
            {
```

```
                cout<<"\n\t";
                cout<<" "<<a[r][c];
            }
        }
    }
}
```

Output

Size of array: 5

Enter element 1 20

Enter element 2 40

Enter element 3 10

Enter element 4 30

Enter element 5 50

10

20

30

40

50

Analysis

Each key is looked at once for each digit (or letter if the keys are alphabetic) of the longest key. Hence, if the longest key has **m** digits and there are **n** keys, radix sort has order O(m.n).

END...