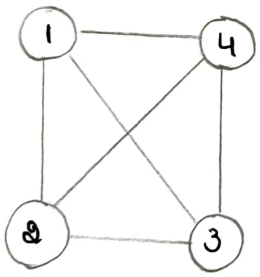
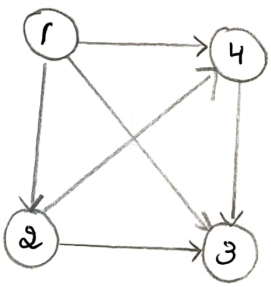
Define the following with respect to graph with an Example

i) Connected graph.

In an undirected graph G, two vertices u and v are said to be connected if there Exists a path from u to v. Since G is undirected, there Exists a path from v to u also. A graph G is said to be connected if and only if there Exists a path between Every pair of vertices

for Example. the graphs shows in figure below are connected graphs.





ii) Directed graph :-

A graph G = (V, E) in which Every Edge is directed is called a directed graph.

Ex :-

here, graph G = (V, E) where

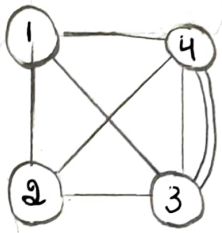
* V = {0, 1, 2} is set of vertices
* E = {<0,1>, <1,0>, <1,2>} is set of Edge

Since all Edge are directed it is a directed graph. In directed graph we use angular brackets < and > to represent an Edge.

iii) multigraph :-

A graph with multiple occurrence of the same edge between any two vertices is called multigraph.

Ex :-

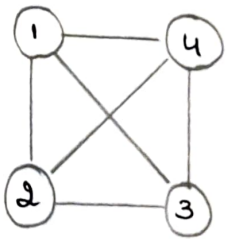


Here, there are two edges between the nodes 1 and 4 and there are three edges between the nodes 4 and 2.

iv. Complete graph :-

A graph $G=(V,E)$ is said to be a complete graph. if there exists an edge between every pair of vertices.

Ex :-

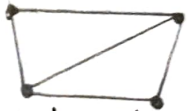


v. Subgraph :-

A subgraph of a graph $G=(V,E)$ is a graph we call (V',E') a subgraph of G if a) $V' \subseteq V$ & $E' \subseteq E$ b) for every edge $e' \in E'$. if e' is incident on v' and w'. then $v', w' \in V'$

Ex :-

K_5

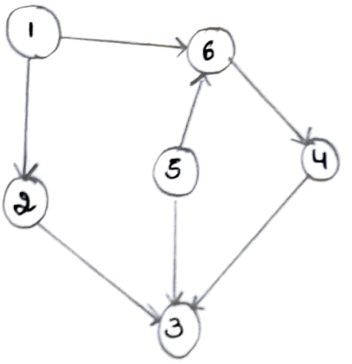


Subgraph of K_5

vii) graph:

A graph Gr is defined as a pair of two sets V and E denoted by Gr = (V, E) where V is set of vertices and E is set of Edges

Ex:



here, graph Gr = (V, E) where

* V = {1, 2, 3, 4, 5, 6} is set of vertices
* E = {<1,6>, <1,2>, <2,3>, <4,3>, <5,3>, <5,6>, <6,4>} is set of directed Edges.

Write a c routine to implement bfs() and dfs() function.

(or)

Write a c function to perform BFS of a graph.

⇒ c function to implement BFS traversal :-

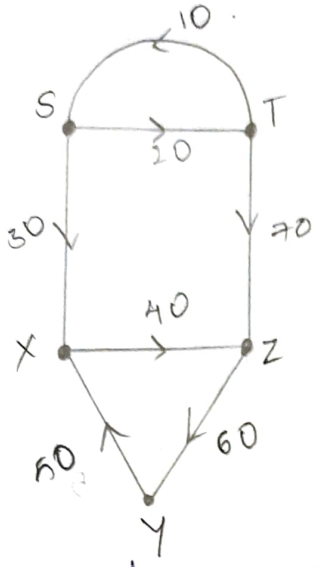
```
void bfs(int a[10][10], int n, int source)
{
    int f, r, q[10], v;
    int visited[10] = {0}; /*Initialize all elements in s to 0 (no nodes visited)*/
    printf("The nodes visited from %d", source);
    f=0, r=-1;                          // queue is empty
    q[++r]=source;                      // insert v into queue
    visited[source]=1;                  // insert v to s
    printf("%d", source);               // print the node visited
    while (f<=r)
    {
        source = q[f++];                // delete an element from q
        for (v=0; v<n; v++)
        {
            if (a[source][v]=1) //if v is adjacent to v
            {
                if (s[v]==0) /*if v is not in s i.e., v has not been visited*/
                {
```

```
                printf("%d", v);      // print the node visited
                visited[v] = 1;       // add v to s, mark as visited
                q[++r] = v;           // insert v to queue
            }
        }
    }
}
printf("\n");
}
```

C function to implement DFS :-

```
void dfs(int source)
{
    int i;
    visited[source] = 1;
    printf("%d", source);
    for(i=0; i<n; i++)
    {
        if(a[source][i] == 1 && visited[i] == 0)
            dfs(i);
    }
}
```

4. Define graph and represent the graph using adjacency matrix.



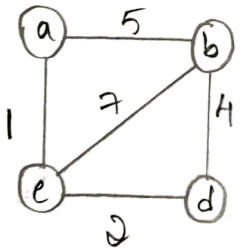
Graph:-

A graph G is defined as a pair of two sets V and E denoted by $G = (V, E)$ where V is set of vertices and E is set of edges.

Adjacency Matrix.

	S	T	X	Y	Z
S	∞	00	30	∞	∞
T	10	∞	∞	∞	70
X	∞	∞	∞	∞	40
Y	∞	∞	50	∞	∞
Z	∞	∞	∞	60	∞

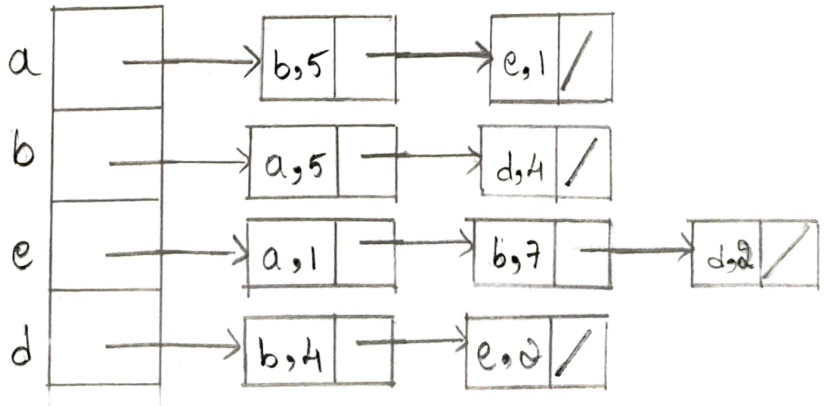
5. Define graph. Give adjacency matrix and adjacency linked list for the given weighted graph, in the fig:



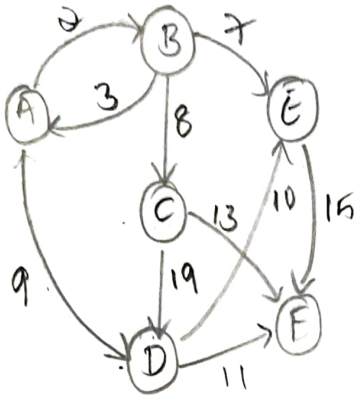
Adjacency Matrix:

	a	b	e	d
a	∞	5	1	∞
b	5	∞	7	4
e	1	7	∞	2
d	∞	4	2	∞

Adjancency Linked List:-



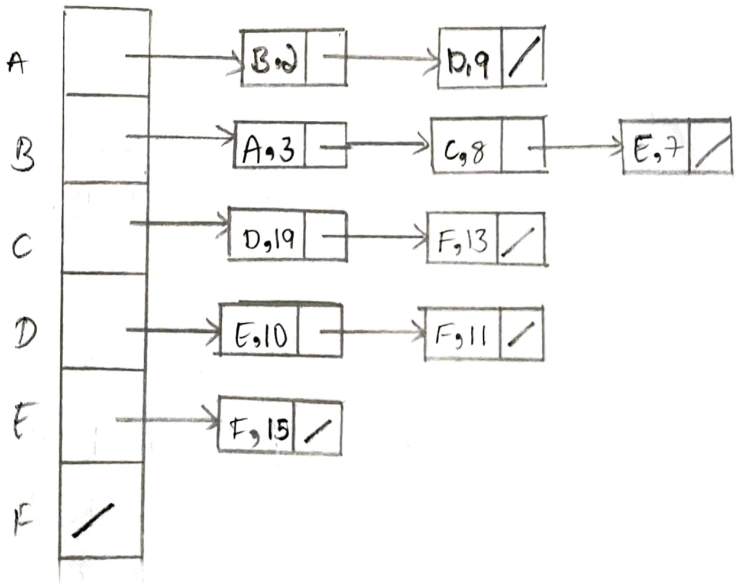
6. Give the adjacency matrix and adjacency list representation for the weighed graph given in the fig.



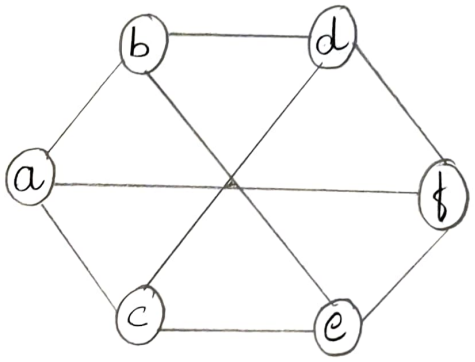
Adjacency Matrix:

	A	B	C	D	E	F
A	∞	3	∞	9	∞	∞
B	3	∞	8	∞	7	∞
C	∞	∞	∞	19	∞	13
D	∞	∞	∞	∞	10	11
E	∞	∞	∞	∞	∞	15
F	∞	∞	∞	∞	∞	∞

Adjacency Linked List:



Define a graph and its traversal methods. List and explain the different graph traversal methods Find resultant of the types of graph traversal method, on the following graph : (consider 'a' as the starting vertex).



(or)

what are the methods used for traversing the graph? Explain one with an example

⇒ There are 2 important graph traversal methods.

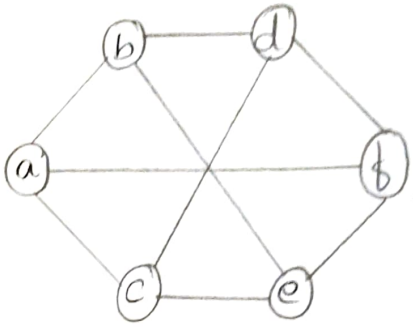
i) Breadth First Search (BFS)

ii) Depth First Search (DFS)

i) Breadth First Search (BFS) :-

It is also known as level order traversal. The Queue data structure is used for BFS traversal. When we use the BFS algorithm for the traversal in a graph we can consider any node as source node.

let us consider, an example



Step1 :- Initially select the source node i.e., a, then insert it into queue

q → | a | | | | | |

Step2 :- check the adjacent neighbour node of a i.e., b and c and f, now insert all the 3 into queue.

a	b	c	f		

Step3 :- Now check the adjacent neighbour node of b which are not visited i.e., d and e, insert it into queue.

a	b	c	f	d	e

Step4 :- Now check the adjacent neighbour node of c, i.e., d and e but they are already visited.

Similarly check adjacent node of f i.e.,

a, d and e which are already visited.

Similarly check for 'd' and 'e', if they are visited then leave it, if there are any adjacent nodes that are not visited, then insert them into the queue.

Queue → | a | b | c | f | d | e |

Result → a b c f d e

ii) Depth First Search (DFS) :-

DFS is a type of graph traversal technique which uses the idea of backtracking. The stack data structure is used for the DFS traversal.

Now, let us consider the same example,

Step 1 :- Select a source node and push it into stack, here the source node is 'a'.

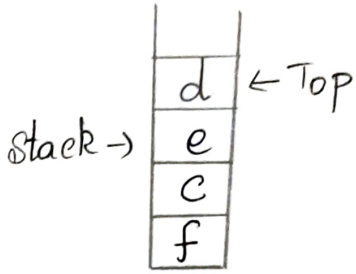
Stack→ | a |

Step 2 :- Pop the node from the stack and push all its adjacent nodes into stack, here the adjacent node for a are b, c and f.

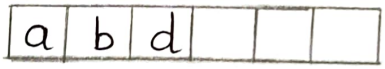
| a | | | | | |

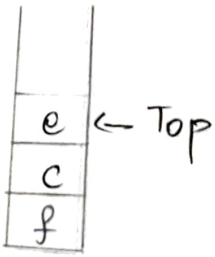
b	← Top
c	
f	

Step 3 :- Pop the node 'b' and push all the adjacent nodes of 'b' i.e., 'd' and 'e' into the stack.

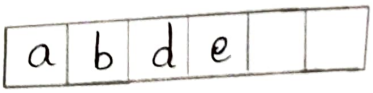


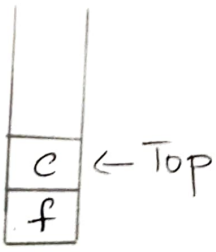
Step 4 :- Pop node 'd' and push all the adjacent nodes of 'd', here the adjacent nodes are 'c' and 'f' which are already in the stack, so just pop 'd'



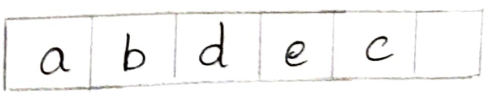


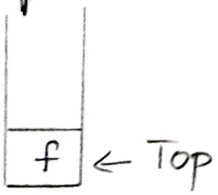
Step 5 :- Pop node 'e' and push its adjacent nodes i.e., c, b and f, but as they are already in the stack, just pop e





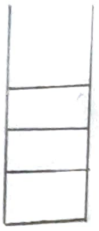
Step 6 :- Pop node 'c' and its adjacents nodes are 'a', 'd' and 'e', which are already visited.





Similarly pop node f. and its adjacent nodes are a, d, e which are already visited. So just pop f.

a	b	d	e	c	f



Write an algorithm for,

i) Breadth First Search

ii) Depth First Search

⇒ i) Breadth First Search :-

No node is visited to start with

Insert source v to q

Print v

Mark v as visited, i.e., add v to s

while queue is not empty

delete a vertex v from q

for every v adjacent to v

if v is not visited

print v

mark v as visited

insert v into queue

end if

end while

// int s[10]={0};

// f=0, r=-1, q[++r]=v

// printf v

// s[v] = 1

// while f<=r

// v=q[f++]

/* for each v, if a[v][v] ==1*/

// if s[v]==0

// print v

// s[v] = 1

// q[++r]= v

// end if

// end while

ii) Depth First Search :-

Step1 :- Select node v as the start vertex, push v onto stack and mark it as visited.

Step2 :- while stack is not empty
for vertex v on top of the stack, find the next immediate adjacent vertex.
if v is adjacent
if a vertex v is not visited, then
push it onto stack and number it in the order it is pushed.
Mark it as visited by adding v to S.

else
ignore the vertex

end if

end while

Step3 :- Repeat step 1 and step 2 until all the vertices in the graph are considered.

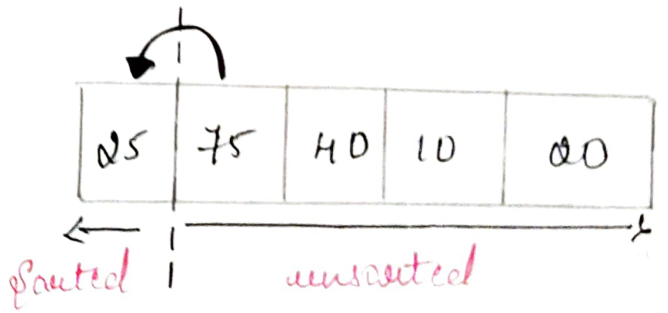
Write a insertion sort algorithm Explain with an example

→ • Step 1 : If it is the first element, it is already sorted. return 1 ;

- Step 2: Pick next element
- Step 3: Compare with all elements in the sorted sub-list
- Step 4: Shift all the elements in the sorted sub-list that is greater than the value to be sorted
- Step 5 : Insert the value
- Step 6 : Repeat until list is sorted

Sort the elements 25 75 40 10 20 using Insertion sort

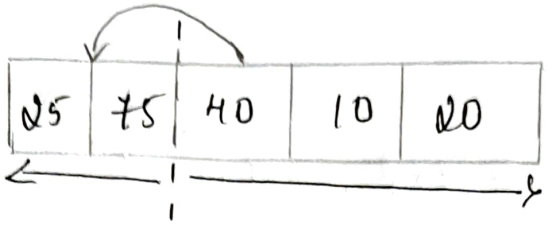
Step 1 : Item to be inserted is 75 i.e, Item = a [1]



75 is inserted after 25

Step 2: Item to be Inserted is 40 i.e,

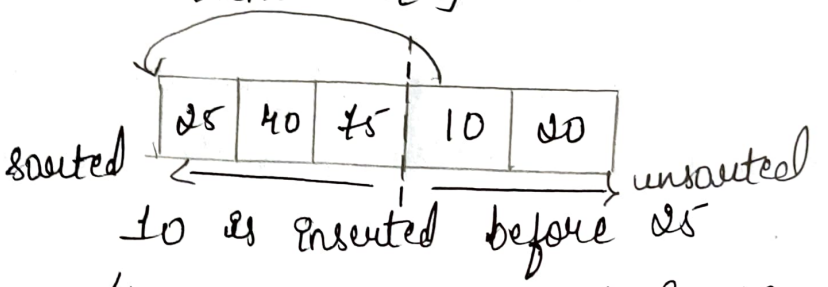
Item = a[2]



40 is Inserted between 25 and 75

Step 3: Item to be Inserted is 10 i.e

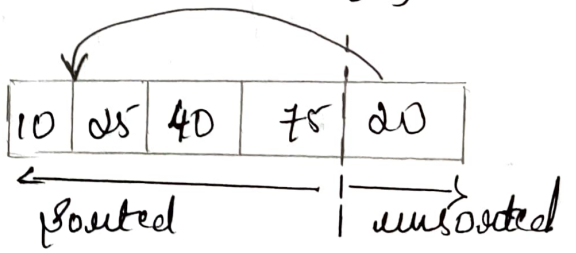
Item = a[3]



10 is Inserted before 25

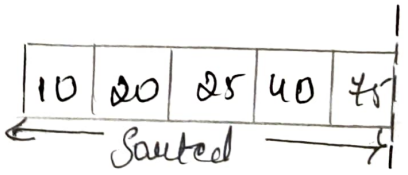
Step 4: Item to be Inserted is 20

i.e, Item = a[4]



20 is Inserted between 10 and 25.

Output:



Final Sorted list.

Write an algorithm for an insertion sort. Also discuss about the complexity of insertion sort. [refer algorithm previous question 20.]

—> Complexity of insertion sort:

- Worst case time complexity : $O(n^2)$
- Average case time complexity : $O(n^2)$
- Best case time complexity : $O(n)$
- Space complexity : $O(1)$

How insertion sort works? Trace the insertion sort algorithm for the following data in ascending order

77, 33, 44, 11, 88, 22, 66, 55

—>. (or)

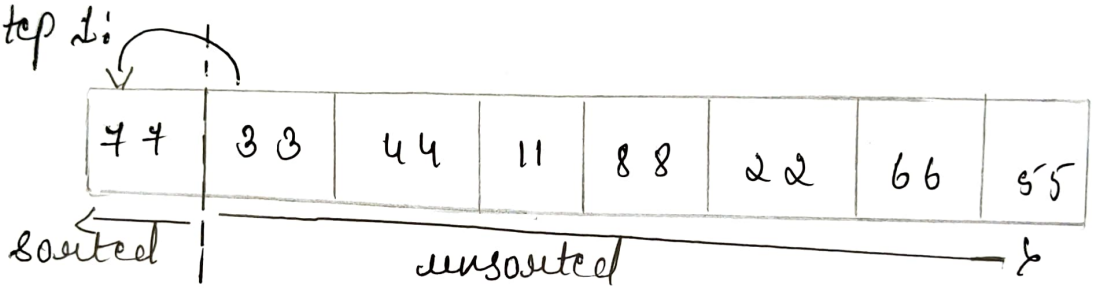
Apply insertion sort technique for the following elements :

77, 33, 44, 11, 88, 22, 66, 55

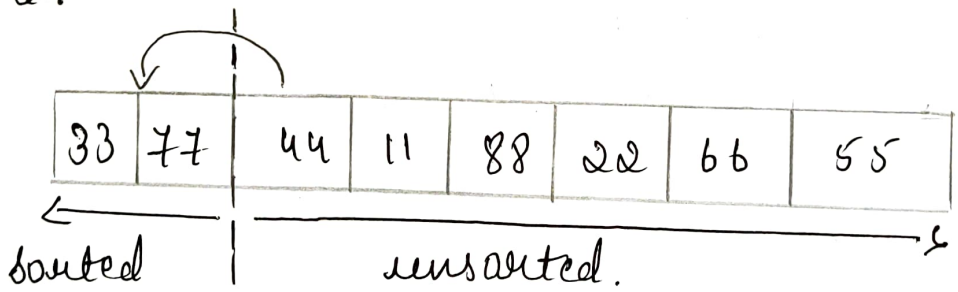
→ Working of Insertion Sort:

- The first element in the array is assumed to be sorted. Take the second element and store it separately in key.
- Now, the first two elements are sorted. Take the third element and compare it with the elements on the left of it.
- Similarly, place every unsorted element at its correct position.

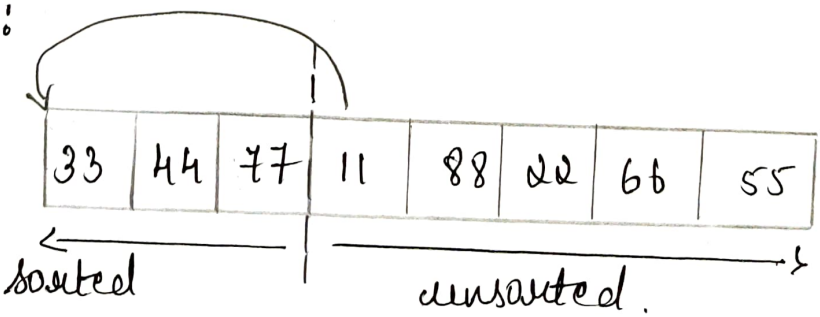
Step 1:



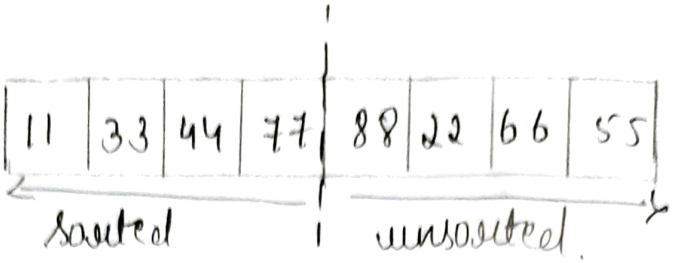
Step 2:



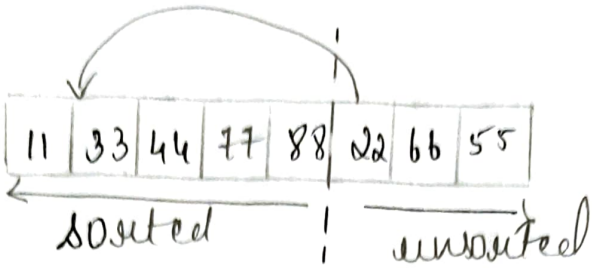
Step 3:



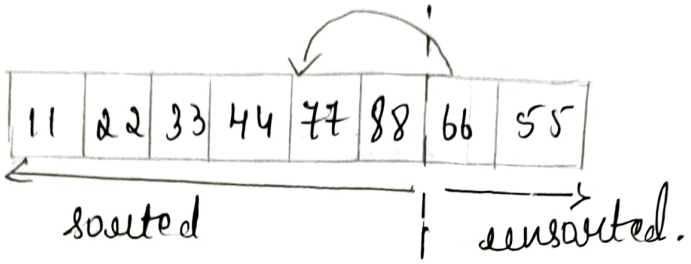
Step 4:



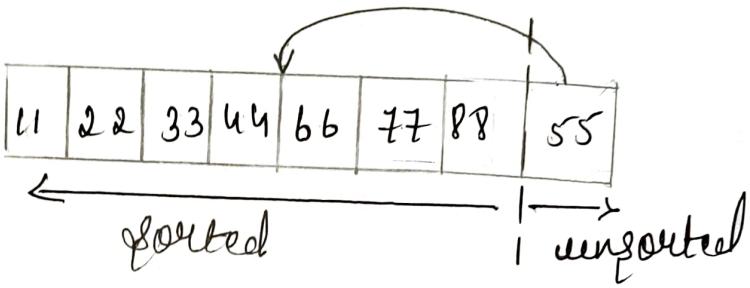
Step 5:



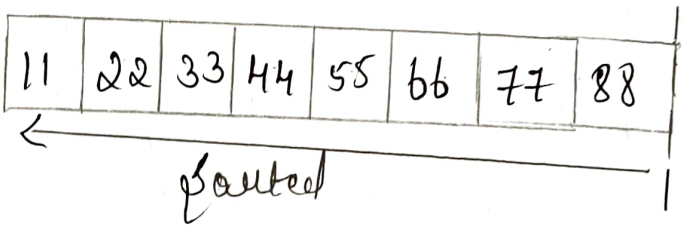
Step 6:



Step 7:



Step 8:



Final sorted list is in ascending order.

11	22	33	44	55	66	77	88

Define file. List basic operations and Explain any 4 operations with syntax & example.

—> A file is a collection of related records.

Eg: We may have a file for each student consisting of student's academic record, student's personal record and student's hostel record & so on. We can have a file of all students also.

Basic file operations:

The basic operations that can be performed on files are:

- Creation
- Updation
- Retrieval
- Maintenance.

*Creation: A file has to be opened in write mode to create a file. The data read from other file or read from the input device can be written into the file. Once, the write operation is over, the file is closed. Now, the file that has been just closed is available for

future read / write operations.

* Updating a file: A file can be opened for updation using "r+" mode. A file be updated by inserting a new record or we can delete an existing record or we can modify the contents of file. After updating the file has to be closed.

* Retrieving from a file: Retrieving from a file is the process of extracting useful information from a file. Information can be retrieved either for analysis purpose or for generating various reports.

* Maintaining a file: It involves re-organization of file contents without changing the contents of the file. This is used only for structural changes such as increasing the field width or adding the fields for specific purpose or deleting the unused fields.

Function Syntax : Description

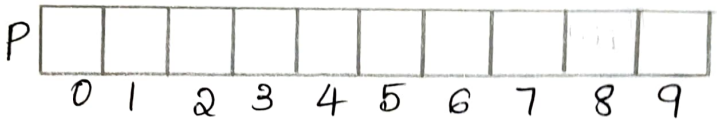
- fopen() - To open a file.
- fprintf() - write data into the file
- fscanf() - reads data from the file
- fputc() - writes a character into the file.

Sort the following list of numbers using Radix sort.

1132, 8344, 2148, 5247, 6214, 9132, 0378, 3666, 4259, 7589

⇒ Let us arrange numbers in ascending order using Radix sort. Since, we are sorting decimal numbers (having base 10), we assume there are 10 pockets ranging from 0 to 9.

Initially, the 10 pockets are empty as shown below



consider, the following elements

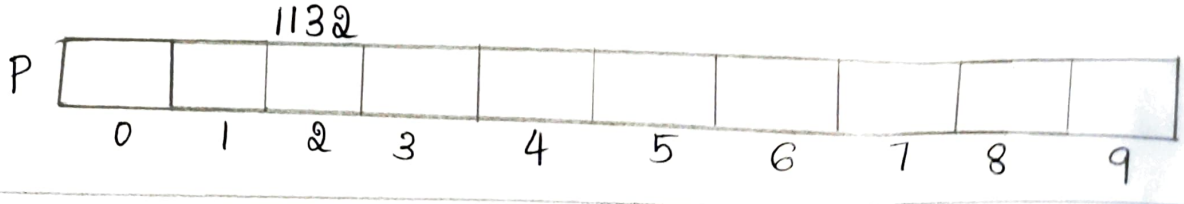
1132, 8344, 2148, 5247, 6214, 9132, 0378, 3666, 4259, 7589

Pass 1 : Activities :

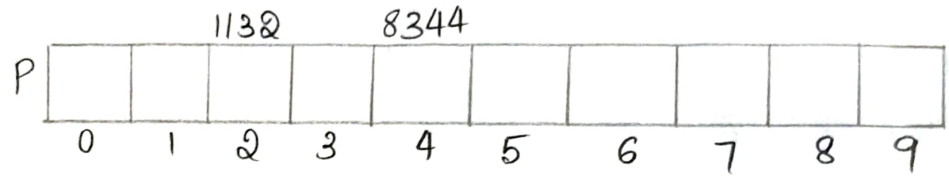
Step 1 :- Scan each item (number) in the list. Obtain least significant digit say d. Insert item into the pocket at position p[d] then insert item above the last item in p[d].

Each item is inserted as shown below,

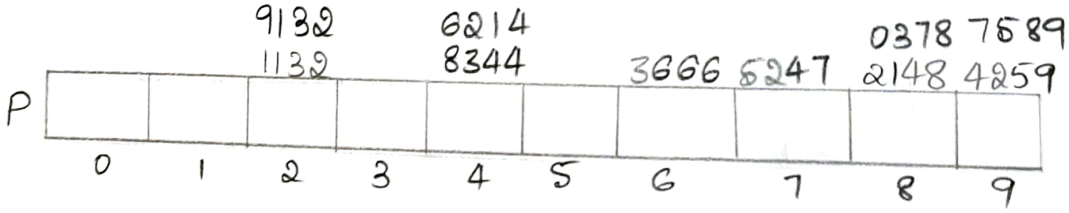
item = 1132, least significant digit, d = 2, so we need to insert at p[2].



item = 8344, least significant digit, d=4, insert 8344 into p[4]



In the similar way all the other items are scanned one after the other, and inserted into appropriate pocket, as shown below.

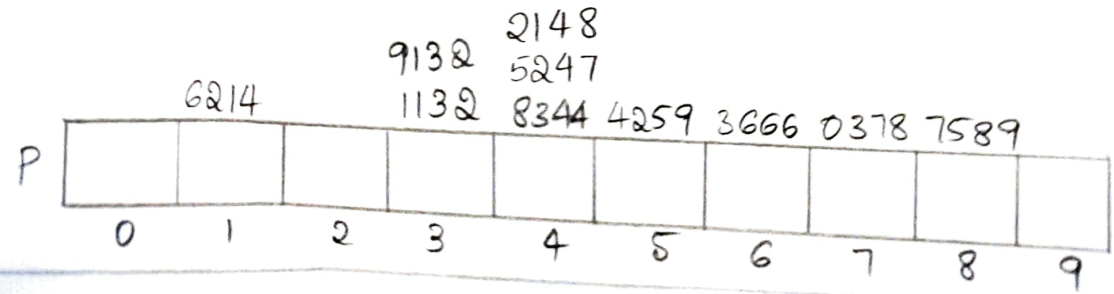


Step 2 :- Now, access each pocket and copy the item present in each pocket one after the other starting from 0 to 9 into original array as shown below.

1132, 9132, 8344, 6214, 3666, 5247, 2148, 0378, 4259, 7589

Pass 2 : Activities : Consider the above items as i/p.

Step 1 :- Scan each item in the above list. Obtain the next least significant digit say d. Insert item into pocket at position p[d].

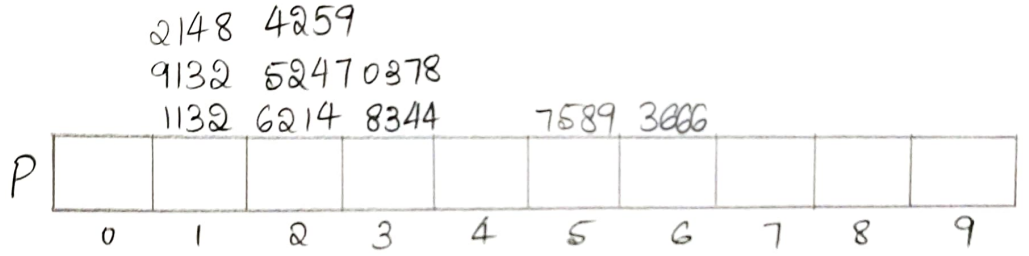


Step2 :- Now, access the pockets one by one and copy the items present in each pocket into original array.

6214, 1132, 9132, 8344, 5247, 2148, 4259, 3666, 0378, 7589

Pass3 : Activities :-

Step1 :- Scan each item in the above list. Obtain the next significant digit say d. Insert item at the position p[d].

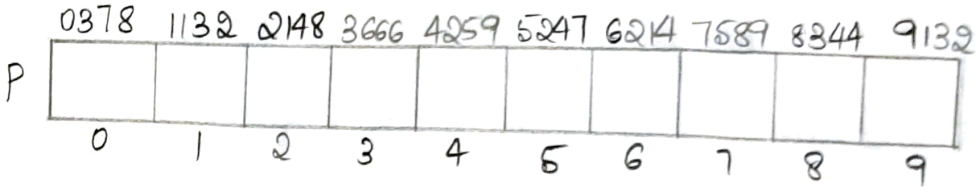


Step2 :- Now, access the pockets one by one and copy items into original array.

1132, 9132, 2148, 6214, 5247, 4259, 8344, 0378, 7589, 3666

Pass 4 : Activities :-

Step1: Scan each item in the above list. Obtain the next least significant digit say d. Insert item into pocket at position p[d].



Step2 :- Now, access item from each pocket one after the other into original array as shown below.

0378, 1132, 2148, 3666, 4259, 5247, 6214, 7589, 8344, 9132

The sorted order is,

0378, 1132, 2148, 3666, 4259, 5247, 6214, 7589, 8344, 9132

Explain in detail about static hashing (OR)

Discuss division, mid square and folding hash function

Static hashing :-

* Division method :-

In this method, we choose a number m which is prime value and it is larger than the number of given Elements n in array a. Here, the key item K is divided by some number m and the remainder is used as the hash value. formally it is written as:

$$h(K) = K \% m$$

Because, we are taking modulo m values, the integers we get from the above function are in the range: 0 to m-1

Ex:-

Consider the Elements: 11, 12, 13, 14 and 15 and Consider the following function

$$H(K) = K \% 5$$

The value of Each function for the item: 10, 11, 12, 13 & 14 can be obtained as shown below:

$H(K) = K \% 5$

$H(10) = 10 \% 5 = 0$ — Insert 10 into table a[0]

$H(11) = 11 \% 5 = 1$ — Insert 11 into table a[1]

$H(12) = 12 \% 5 = 2$ — Insert 12 into table a[2]

$H(13) = 13 \% 5 = 3$ — Insert 13 into table a[3]

$H(14) = 14 \% 5 = 4$ — Insert 14 into table a[4]

(Hash values)

Thus, the table obtained after inserting Each of the items using hash value as the index is shown below:

a[0]	10
a[1]	11
a[2]	12
a[3]	13
a[4]	14

* mid - Square method :-

In this method, the key K is squared. A number in the middle of K^2 is selected by removing the digits from both ends. The hash function is defined as shown below:

$h(K) = l$

where l is obtained by removing digits from both ends of K^2. By selecting the middle portion of squared number, different keys are expected to give different hash values. Normally, the size of the hash table using this technique will be power of 2.

for Ex :- Consider key K = 2345, It square i.e $K^2 = 5745 25$

h(2345) = 57] 45 [25

= discarding 57 in the beginning and 25 from the end

$\therefore h(2345) = 45$

* Folding method

In this method, the key k is divided into number of parts $K_1, K_2, K_3 \ldots K_n$ of same length except the last part. then all parts are added together as shown below

$h(k) = K_1 + K_2 + K_3 + \ldots K_n$

for Ex: Consider key $K = 123987234876$. The partitions are $K_1 = 12$, $K_2 = 39$, $K_3 = 87$, $K_4 = 23$, $K_5 = 48$, $K_6 = 76$. now, the hash value can be obtained by adding all the partitioned keys as shown below:

$h(k) = 12 + 39 + 87 + 23 + 48 + 76$

$= 285$

what is hashing? what is collision? what are the methods used to resolve collision? Explain linear probing with an Example. (OR) Explain dynamic hashing.

* Hashing is process of mapping large amounts of data into a smaller table using hash function. hash value and hash table is called hashing.
* The phenomenon of two or more keys being hashed to the same location of hash table is called collision
* The methods used to resolve collision are
  1. Open addressing (linear probing)
  2. Chaining.

1. Open addressing:- Collision can be avoided by finding another, unoccupied location in the array. The collision can be avoided using linear probing.

Ex:- Construct a hash table ht of size 15 (using open addressing method) to store the following words:
like, a, tree, you, first, find, a, place, to, grow, and, then, branch and out.

Solution:- Now, we "Find hash address for each word". let us find the hash address by taking the sum of positions of alphabets in the word and taking the remainder by dividing it by 15. the various hash address or hash values are shown below:

Sl no	words	Sum of positions			hash address = Sum % 15
1	like	12+9+11+5	=	37	7
2	a	1	=	1	1
3	tree	20+18+5+5	=	48	3
4	you	25+15+21	=	61	1
5	first	6+9+18+19+20	=	72	12
6	find	6+9+14+4	=	33	3
7	a	1	=	1	1
8	place	16+12+1+3+5	=	37	7
9	to	20+15	=	35	5
10	grow	7+18+15+23	=	63	3
11	and	1+14+4	=	19	4
12	then	20+8+5+14	=	47	2
13	branch	2+18+1+14+3+8	=	46	1
14	out	15+21+20	=	56	11

The words are taken in the order of Serial number from the above table and inserted into hash table one by one based on the hash address

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	a	you	tree	find	to	grow	like	place	and	then	branch	first	out	

ht

2. Chaining :-

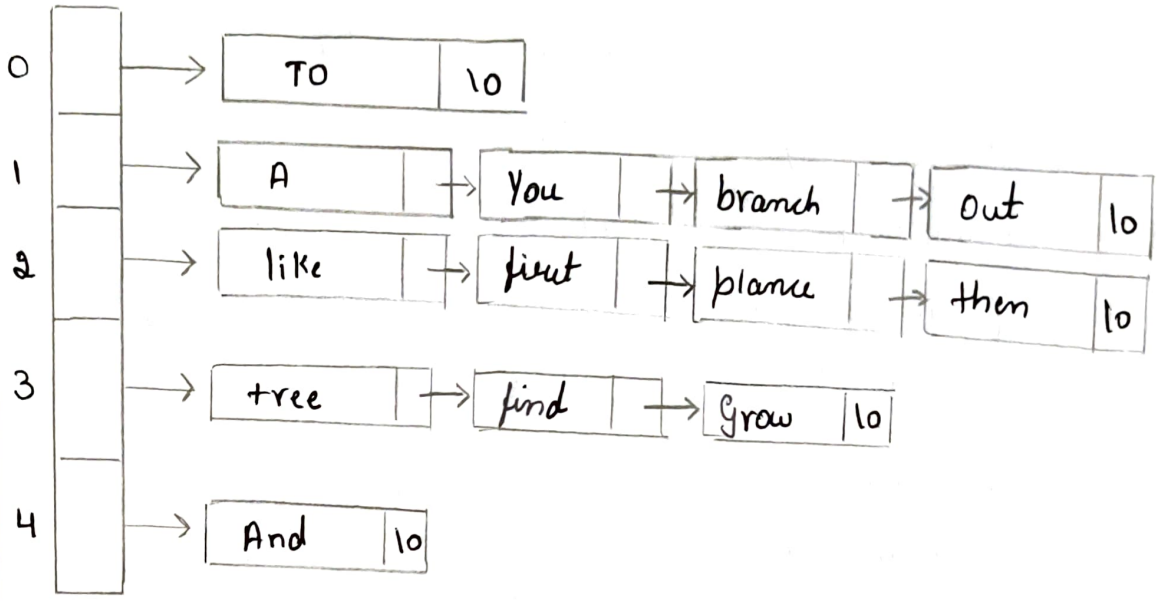
The chaining technique is a very simple method using which collisions can be avoided. This is achieved using an array of linked list.

Ex :- Construct a hash table of size 5 and store the following words : like, a, tree, you, first, find, a, place, to, grow, and, then, branch, out.

Design of Hashing function :- let us design a simple hashing function which will accept a string consisting of a key and add the positions of each letter of the string and compute the remainder by dividing the sum by the size of the table.

Sl no	words	Sum of positions	hash address = Sum % 5
1	like	12+9+11+5 = 37	2
2	a	1 = 1	1
3	tree	20+18+5+5 = 48	3
4	you	25+15+21 = 61	1
5	first	6+9+18+19+20 = 72	2
6	find	6+9+14+4 = 33	3
7	a	1 = 1	1
8	place	16+12+1+3+5 = 37	2
9	to	20+15 = 35	0
10	grow	7+18+15+23 = 63	3
11	and	1+14+4 = 19	4
12	then	20+8+5+14 = 47	2
13	branch	2+18+1+14+3+8 = 46	1
14	out	15+21+20 = 56	1

Construction of Hash table:-



Initially, following keys 10, 16, 11, 1, 3, 4, 23, 15 are inserted into an empty length of 10. Using open addressing with Hash function $h(K) = K \bmod 10$, and linear probing. What is the resultant hash table?

⇒ Given hash function, $h(K) = K \bmod 10$

The values of each function for the items: 10, 16, 11, 1, 3, 4, 23, 15 can be obtained as shown below.

$h(K) = K \% 10$

$h(10) = 10 \% 10 = 0$

$h(16) = 16 \% 10 = 6$

$h(11) = 11 \% 10 = 1$

$h(1) = 1 \% 10 = 1$

$h(3) = 3 \% 10 = 3$

$h(4) = 4 \% 10 = 4$

$h(23) = 23 \% 10 = 3$

$h(15) = 15 \% 10 = 5$

} hash values

Hash table :-

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
10	11	1	3	4	23	16	5		

*) The hash value we obtain for key 10 using the hash function is 0, so we store it at the index position 0.

*) Similarly hash value for key 16 and 11 are 6 and 1 respectively, so we store them at position 1 and 6.

*) Here, the hash value of key 1 is also 1, so collision occurs, so to resolve this we use linear probing, according to this, we need to store 1 in the next unoccupied location, in this case its position 2.

*) key 3 and 4 are stored in positions 3 and 4.

*) Hash value of key 23 is also 3 again collision occurs, because of that we store 23 in next unoccupied location i.e., position 5.

*) Similarly hash value of key 15 is 5, as we can't store 15 in 5 as the location is occupied we store it in next unoccupied location i.e., position 6.

Write address calculation Sort algorithm Sort Z ; A, P, B, Q, I, J, K using the address Calculation Sort algorithm.

$Z \rightarrow \frac{26}{10}$

$F[2] = 2$

$A \rightarrow \frac{1}{10}$

$= 0$

$F[0]$

$B \rightarrow \frac{2}{10}$

$F[0] = 0$

$P \rightarrow \frac{16}{10}$

$F[1] = 1$

$Q \rightarrow \frac{17}{10}$

$F[1] = 1$

$I \rightarrow \frac{9}{10}$

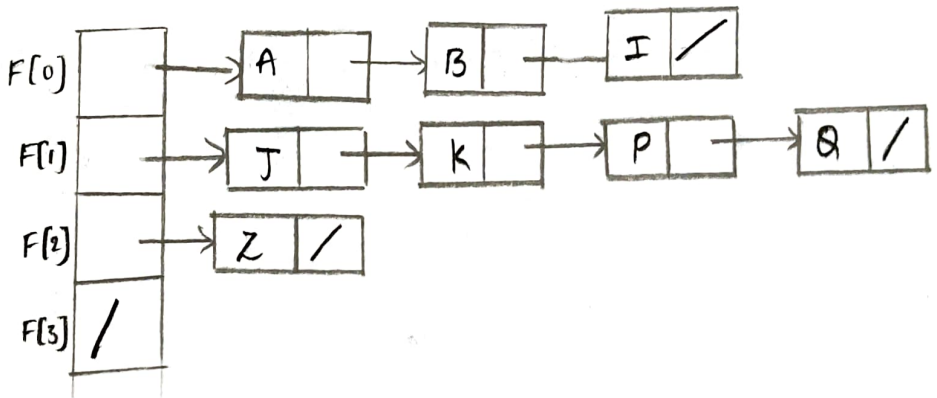
$F[0] = 0$

$J \rightarrow \frac{10}{10}$

$F[1] = 1$

$K \rightarrow \frac{11}{10}$

$F[1] = 1.$



Q8) What are the basic operations that can be performed on a file? List the methods used for file Organization any (2).

→ Basic Operations that can be performed on file.

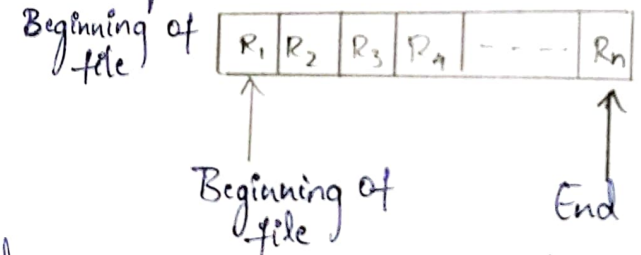
* Creation
* Updation
* Open
* Write
* Retrieved
* Maintenance
* Delete
* Close.

Methods Used for file Organization.

1. Sequential file Organization.
2. Relative file Organization.
3. Indexed file Organization.

1. Sequential file Organization:

In this technique, the records are stored in the order in which they were entered i.e., they are stored one after other in a sequential manner as shown in fig.



* Records are written in the order in which they are entered.
* Records are read and written sequentially.
* All records will have same size and width.
* Records may be sorted based on search key

* The design is very simple
* Design is very easy to handle.
* There is no much effort involved to store the data.
* This Organization is very efficient when large amount of data has to processed.
* files can be stored on magnetic disks and tapes which are cheap.

Disadvantages:

* Record are read only sequentially
* If the sorted files are used, they involve more effort as every updation requires sorting of the file.

Relative file Organization:

In this technique, the records of the file are stored one after the other both physically & logically. This technique is also called as random access file Organization @ direct access file Organization.

0	record 0
1	record 1
2	record 2
3	free
4	free
5	free
98	record 98
99	free

* All the records are identified by their position to relative to the beginning of the file.
* Any record can be read by specifying the key value. The location of record i can be obtained using the formula

Address of i^{th} record = base address + $(i-1)*w$

* Every location in the table either stores a record @ it is marked as free.

Advantages.

* we can access the desired record immediately.
* Accessing of the records is easier.
* New records can be added very easily in free location based on relative record no of the record to be inserted.
* Updating the records is very easy.

Disadvantages

* Record should be of fixed length only.
* The relative position of the record must be known before processing.

Explain how does an append mode in a file operation differ from the write mode.

Append mode	write mode.
* Append mode is used to add the data at the end of the file if the file is already exists	* write mode creates a new file
* Append mode don't erase previous data, it append new data with previous one.	* write mode erase the previous data of the file and insert new data.
* The append mode add the data at the end of the file if the file already exists otherwise creates a new one.	* The write mode creates a new file, if the file is already existing mode over writes it.