→ Fundamentals of Data structure:

The related information cannot be stored using single variable. thus to handle such information or situations the concept of Datastructure is introduced.

In Computer we use variables to store the data, but variables cannot handdle huge amount of related information, so data structures are useful.

Definition:

Data structure is a particular way of organizing and storing data in a computer is called as Data structure.

(or)

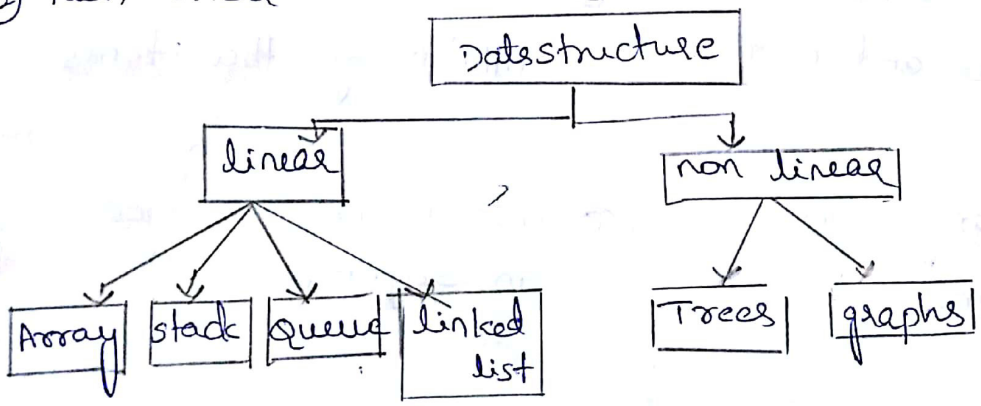
It is a kind of representation of logical relationship b/w related data elements.

- The operations can be storing, retrieving and accessing the related data.

A Data structure can be classified into two ways.

① Linear Datastructure

② Non linear Data structure.




classifications of Data structure.

→ Linear Data structure:

A Data structure is said to be linear, If it's elements are stored in sequential Manner.

- Here A linear relationship if formed b/w elements, with the help of ~~sequential memory~~ sequential memory locations.

Ex: Arrays, stacks, Queue and linked list.

→ Non linear Data structure:

The Data structure where data items are not organized sequentially is called non linear Data structure.

(or)

A data elements of the non linear data structure could be connected to more than one elements to reflect a special relationship among them.

Ex: Trees, graphs.

→ Difference b/w linear and non linear Data structures:

| linear | non linear |
|---|---|
| ① every item is related to its previous and next item. | ① every item is attached with many other items. |
| ② Data is arranged in linear sequence. | ② Data is not arranged in sequence. |

| | |
|---|---|
| ③ Data item can be traversed in a single run. | ③ Data can not be traversed in a sigle run. |
| ④ Ex: Array, stack, queue, linked list | ④ Ex: Trees, graphs. |
| ⑤ implementation is easy. | ⑤ implementation is Difficult. |

⇒ STACK :

A linear Datastructure in which data item is inserted or deleted on one end.

- the end where the data items are inserted or deleted is called "TOP OF THE STACK".
- STACK is called as "LastInFirstOut (LIFO)" (or) "FirstInLastOut (FILO)" structure.
- the initial value of the TOP OF THE STACK is "-1"
- Where we want to insert a value into a stack, increment the stop value by one. and then insert.
- Where we want to delete the value from the stack, the delete the top value and decrement the top value by one.

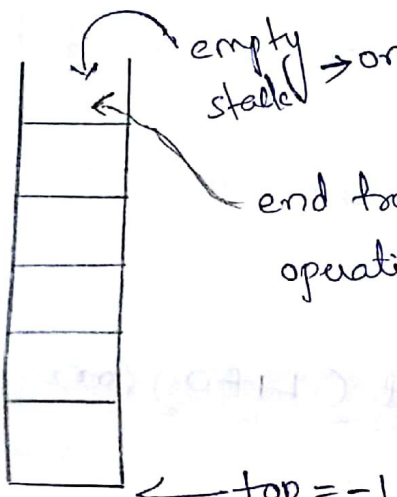
## Operations on stack:

① push operation – Insert some data into stack → top incremented by 1 (top++)

② pop operation – Deleting an element from stack → Top decremented by 1 (top--)

③ overflow – if we try to insert some value, when the stack is full, this operation is executed.

④ underflow. ↓ if we try to pop some elements, when the stack is empty, this operation is executed.




initial stack.

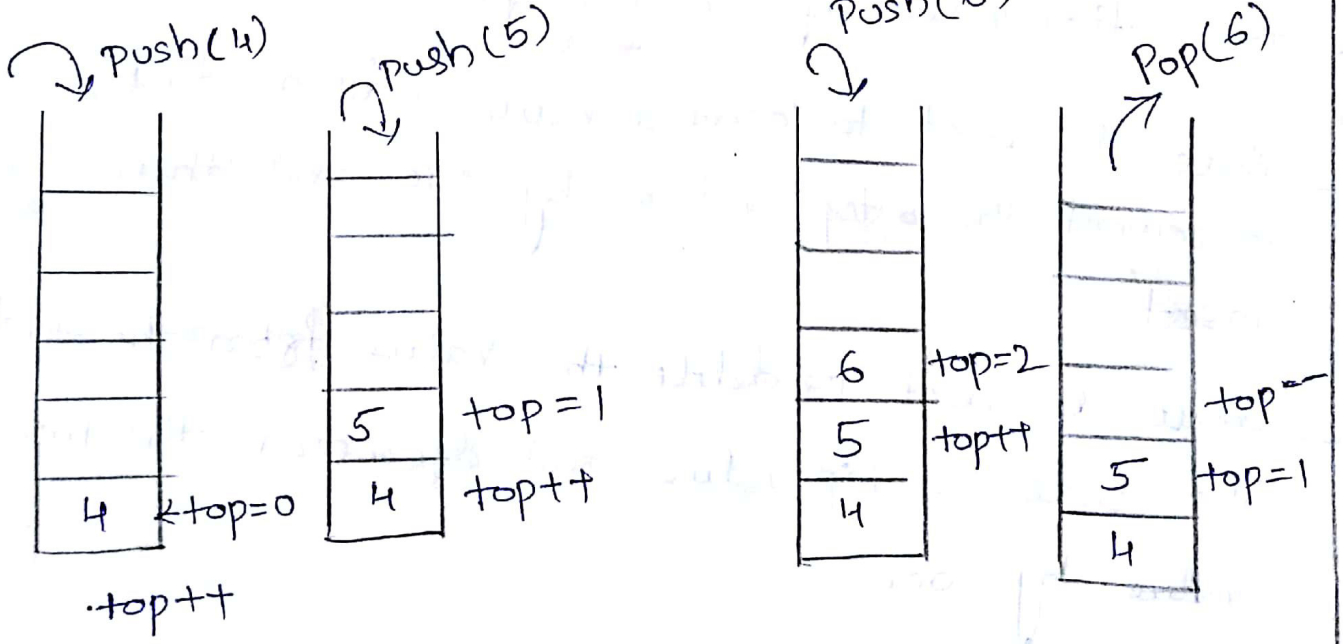



fig: → when the stack size = 6

## Applications of stack:

→ Handling dynamic Memory Allocations.

→ stack is used to evaluate the expressions.

→ stack is used to convert infix to postfix / prefix form.

→ During a function call the return address and aguments are pushed onto a stack. and return they are popped off.

## Implementation of stacks:

→ two types:

① implementation of stack using Arrays.

② implementation of stak using pointers.

## ① implementation of stack using Arrays:

→ TO create a empty stack:-

step1: include headerfiles which are used in the program. and define a Constant size with specific value.

step2: create one dimensional array with fixed size. (int stack[SIZE].)

step3: define an integer variable "top" and initize with '-1'. (int top= -1)

push operation :

step1: check whether stack is full. (top == SIZE-1)

step2: if stack is full insertion is not possible.

step3: If not full, then increment top value. (top++). and set stack[top] to value. (stack[top] = value).

POP operation :

step1: check whether stack is empty (top == -1).

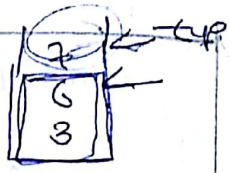
step2: if stack is empty then display stack is empty "Deletion is not possible".

step3: if it is not empty then delete "stack[top]" & decrement top value by one. (top--)

Example program :-

```
/* implementation of stack using arrays */
#include <stdio.h>
#include <conio.h>
void main()
{
int s[20], i, a, size, top= -1; choice;
```

```
clrscr();
printf("enter a stack size:\n");
scanf("%d", &size);
while(1)
{
 Printf("enter choice 1. push\n 2. pop\n 3. display\n
                4. top\n 5. exit\n");
 scanf("%d", &choice);
 switch(choice)
  {
   case 1: if(top == size-1)
            {
             pf("stack is full\n");
            }
           else
           {
            printf("enter the element tobe
                        inseated");
            scanf("%d, &a);
             top++;
             s[top]=a;
            }
           break;
```




```
case 2: if (top==-1)
        {
         printf("stack is empty");
        }
        else
        {
         printf("element popped %d", s[top]);
         s[top]=0;
         top--;
        }
        break;

case 3: if (top==-1)
        {
         printf("stack is empty\n");
        }
        else
        {
         printf("elements of stack are \n");
         for (i=top ; i>=0; i--)
         printf("%d \n", s[i]);
        }
        break;

case 4: printf("top element = %d ", s[top]);
        break;

case 5: exit(1);
        break;
default: printf(" please enter choice between 1 to 5
                                          only ");
        break;
    }
}
getch();
```

O/p:

```
enter the stack size : 5 ↵

enter choice : 1.push
               2.pop
               3.display
               4.top
               5.exit

1 ↵

enter the element tobe inserted : 15 ↵

:
:
:
!
 5 ↵
exit.
```

⇒ **QUEUES:-**

It is a linear Data structure in which a data item is inserted at one end and deleted at other end.

→ The end where we data is inserted is called as "Rear" end and end from where we remove is called "front" end.

→ Queue is called as "First In First Out" (FIFO) or "Last In Last Out" (LILO) structure.

→ Initially both "front" and "rear" are set to -1. That means Queue is empty.

→ We want to insert a new value into Queue increment "rear" value by one.

→ We want to delete the element from the Queue then increment 'front' value by one.

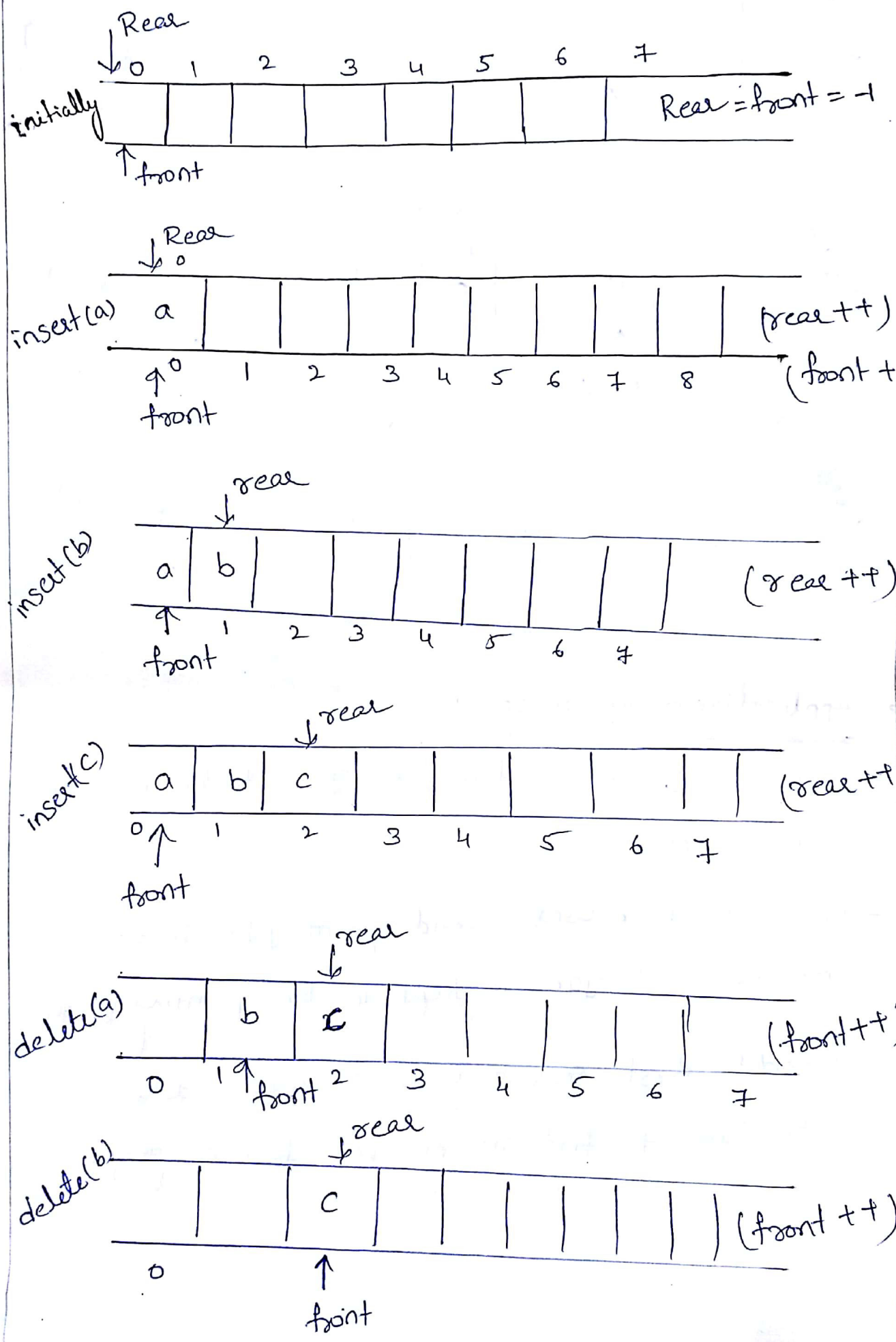
**Operations on Queue:**

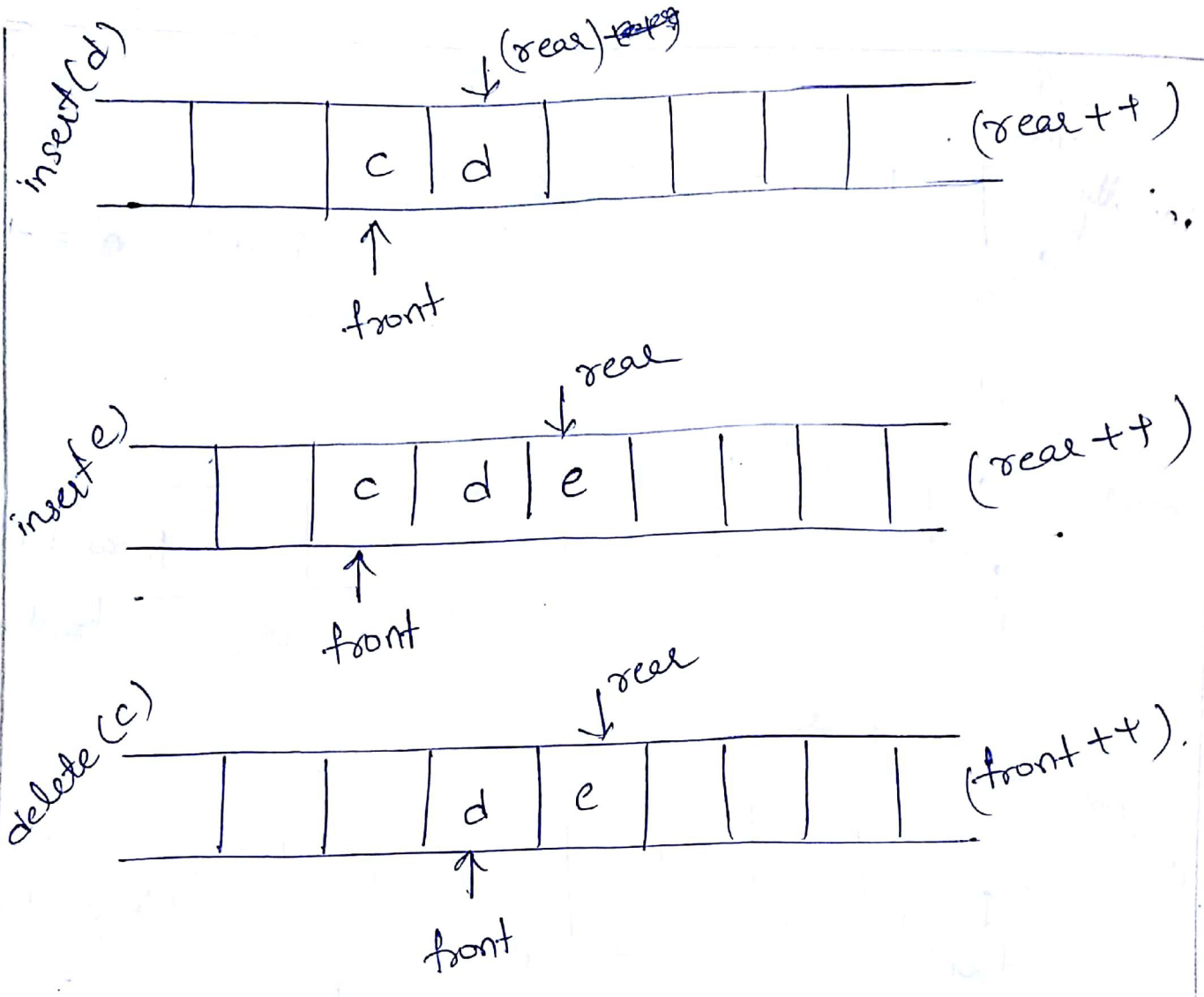
① Insert operation - placing data item into queue (enQueue)

② Delete operation - Removing data item into queue (deQueue)

③ Display operation - used to display the elements in queue.

Note: When we insert "first item" into queue, both rear and front will be incremented.






→ Applications of queue:

- It is used to schedule the Jobs to be processed by the cpu.
- When Multiple users send print jobs to be printer, each Job is kept in the printing queue.
- Breadth first search uses a queue data structure to find an element from a graph.

## Implementation of Queue :-

two types:

1. implimentation of Queue using Arrays
2. implimentation of Queue using pointers.

→ **implementation of Queue using Arrays:**

**To create a empty stack:-**

step1: include headerfiles which are used in the program and define a constant size with specific value.

step2: Create one dimensional array with fixed size (int queue[SIZE])

step3: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front=-1, rear=-1).

**enQueue / insertion operation:**

step1: check whether Queue is full (rear==size-1)

step2: if Queue is full insertion is not possible.

step3: if it is notfull, then increment rear value. (rear++). and set "queue[rear] = value".

dequeue / Deleting operation :-

step1: check whether queue is empty. (front == rear).

step2: if queue is empty then display queue is empty. "Deletion is not possible".

step3: if it is not empty then increment the front value by one (front++). then display queue[front] as deleted element.

→ then check whether both front and rear are equal (front == rear); if it true, then set both front and rear to -1. (front = rear = -1).

Example program:

```
/* implementation of Queue using array */.
# include <stdio.h>
# include <conio.h>
void main ( )
{
 int s[20] , i, a , size , front = -1 , rear = -1 ,
     choice ;
 clrscr ( );
 printf ("enter a Queue size : \n" ) ;
 scanf ("%d" , & size ) ;
 while (1)
  {
   printf ( " enter choice 1. insert \n 2. delete\n.
          3. display \n 4. exit \n ) ;
   scanf ( " %d " , & choice) ;
   switch (choice)
   {
case 1: if ( rear = = size-1)
      {
       printf ( " Queue is full\n" ) ;
      }
```

```
else
{
 if (front == -1) // initially Queue is empty)
 {
   front = 0;
 }
pf ("enter element to add in queue \n");
sf ("%d", &add_value);
 rear = rear + 1;
  a[rear] = add_value;
}
break;
```

Case2:
```
if (front == -1 || front > rear)
 {
   pf("Queue is empty \n");
 }
 else
  {
   pf("element deleted from queue=%d\n",
                              a[front]);
   front = front + 1;
  }
 break;
```

```
case 3: if (front == -1)
        {
          pf ("Queue is empty \n");
        }
        else
        {
          pf("Queue is \n");
          for (i=front ; i<=rear; i++)
          {
            pf("%d", s[i]);
          }
        break;

case 4: exit(1);
        break;

default: printf("please enter choice between
                 1 to 4 only");
          break;
      }
   }

getch();

}.
```

o/p:

enter the Queue size : 15 ↵

enter choice : 1. insert
2. delete
3. display
4. exit.

1 ↵

enter element to add in queue : 10 ↵

.
.
.
.

4 ↵

exit:

Non linear Data structures:

① Trees
② graphs.

⇒ ① ~~Trees~~ Trees:

Tree is a non-linear data structure. which organizes data in hierarchical manner.

(OR)

Tree data structure is a collection of data (Node). which is organized in hierarchical.

→ starting Node is called as root Node.

→ each node consist of one value.

Ex:

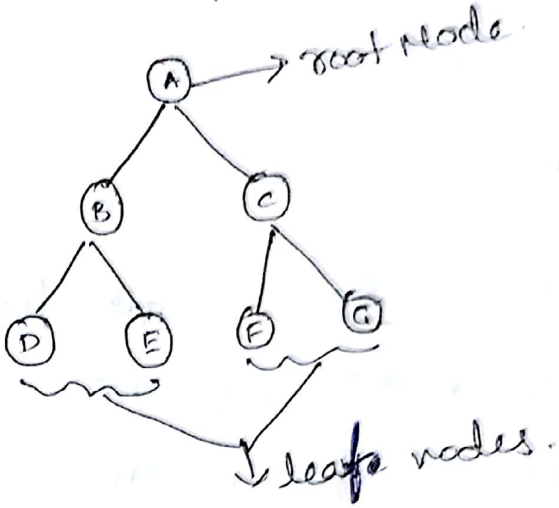



fig: sample tree.

graphs: graph is a non-linear Data structure.
graph consist of finite set of vertices.
(or) nodes (or) points to gethes with a set of unordered pairs.

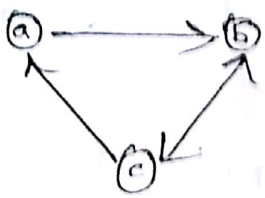
these pairs are know as edges, arces.

* graphs in two types:

① directed graph
② undirected graph.

directed graph: A graph is said to be directed graph. whose definition Makes reference to edges. which are directed. i.e edges which are ordered - pair of vertices.

Ex:




i. graph with 3 vertices & 3 edges.

## Undirected graph:

A graph is said to be undirected graph whose definition makes references to unordered pairs of vertices as edges is know as an undirected graph.

Ex:

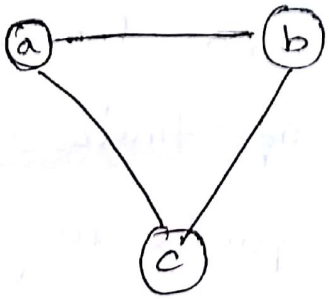



fig: Sample graph.

## Array

→ An array is a collection of elements of the same datatype.

→ we can use arrays to represent not only simple list of values but also tables of data in two or three dimensional array, the types of arrays are:

1) Single dimensional array
2) Two-dimensional array
3) multi-dimensional array

### Single dimensional array:-

→ A list of elements can be given one variable name using only one subscript and such variable is called a single-dimensional array.

Syntax

```
type Variable_name [size];
```

Eg:- int A[5];

### Initialization:-

→ we can initialize the elements of array in the same way as the ordinary variables when they are declared. The general form of initialization of array is:

```
type arr_name [size] = { list of values };
```

Eg:- int A[5] = {1, 2, 3, 4, 5};

Eg:- write a program for finding largest elements in the array.

```
#include <stdio.h>
#include <conio.h>
main()
{
  int arr[7] = { 7, 13, 21, 16, 9, 31, 24 };
  int i;
  large = arr[0];
  for(i=1; i<7; i++)
  {
    if (arr[i] > large)
    {
      large = arr[i];
    }
  }
}
```

## Two-Dimensional Arrays:-

→ Array which contain two subscripts are called two-dimensional array.

Syntax:

```
type arr_name [row] [column];
```

Eg:- int A[3][2];

→ In 2-D array first index selects row & the second index selects column within that row.

### Initialization:-

→ 2-D arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

Eg:- int A[2][3] = {0,0,0,1,1,1};

Eg:- Addition of a matrices.

```
#include <stdio.h>
#include <conio.h>
Void main()
{
   int i, j, a[3][3], b[3][3], c[3][3];
   clrscr();
   printf ("Enter the first matrix:\n");
   for (i=0; i<3; i++)
   {
     for(j=0; j<3; j++)
       {
```

```
printf("%d", a[i][j]");
scanf("%d", &a[i][j]");
}
}
printf("Enter the second matrix:\n");
for(i=0; i<3; i++)
{
  for(j=0; j<3; j++)
  {
    printf("%d", b[i][j]");
    scanf("%d", &b[i][i]);
  }
}
printf("The entered matrices are:\n");
for(i=0; i<3; i++)
{
  printf("\n");
  for(j=0; j<3; j++)
  printf("%dit", a[i][j]);
  printf("\t");
  printf("%d for(j=0; j<3; j++)
  printf("%dit", b[i][j]);
}
```

```
for(i=0; i<3; i++)
for(j=0; j<3; j++)
c[i][j] = a[i][j] + b[i][j];
printf ("\n the sum of two matrices are");
for(i=0; i<3; i++)
{
printf ("\n\t");
for(j=0; j<3; j++)
printf ("%d\t", c[i][j]);
}
getch();
}
```

Multi-dimensional array:-

→ 'C' allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is:

```
type arr-name [s1][s2][s3]---[sm];
```

Eg:- int table[5][5][5];

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int i,j,k;
 int arr[3][3][3] =
                    {
                      {
                       {11,12,13},{14,15,16},{17,18,19}
                       }
                       {
                       {21,22,23},{24,25,26},{27,28,29}
                      }
                      {
                      {31,32,33},{34,35,36},{37,38,39}
                       },
                       };
              clrscr();
              printf("3D array elements are\n");
for(i=0;i<3;i++)
{
 for(j=0;j<3;j++)
  {
   for(k=0;k<3;k++)
   {
    printf("%d\t",arr[i][j][k]);
    }
   printf("\n");
  }
  printf("\n"); } }
```

## Linear Search:-

→ The linear search or the sequential searching is most simple searching method.

→ The Key, which is to be searched is compared with each element of list one by one, if the match exists, the search is terminated.

→ If the end of the list is reached, it means that the search has failed and the key has no matching element in the list.

Eg:-

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

Key = 12

Step1

Search element 12 is compared with first element 60

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are not matching - So move to next element.

Step2

Search element 12 is compared with next element 20.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are not matching, so move to next element 10.

Step 3

Search element 12 is compared with next element 10

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are not matching, so move to next element.

Step 4

Search element 12 is compared with next element 55

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are not matching, so move to next element. 32

Step 5

Search element 12 is compared with next element 32

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are not matching, so move to next element

Step 6

Search element 12 is compared with next element 12

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 60 | 20 | 10 | 55 | 32 | 12 | 50 | 99 |

12

Both are matching, so we stop comparing and display element found at index 5.

```
#include <stdio.h>
#include <conio.h>
main()
{
int A[5] = {10,20,30,40,50};
int Key, flag = 0;
printf("Enter search Key");
scanf("%d", &Key);
for(i=0; i<5; i++)
{
   if(Key == A[i])
    {
      flag = 1;
      break;
    }
}
if(flag == -1)
{
  printf("Key is found");
}
   else
    {
      printf("Key not found");
    }
}
```

Binary search:-

→ The main constraint of binary search is that the elements should be in ascending order.

→ The given list of elements are divided into two equal halves, the given key is compared with the middle element of the list. Now three situations may occur:

i) The middle element matches with the key - the search will end peacefully here.

ii) The middle element is greater than the key - then the value which we are searching is in the first half of the list.

iii) The middle element is lower than the key - then the value which are searching is in the second half of the list.

Eg:-

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 20 | 23 | 52 | 56 | 89 | 93 |

Key = 93

Step 1

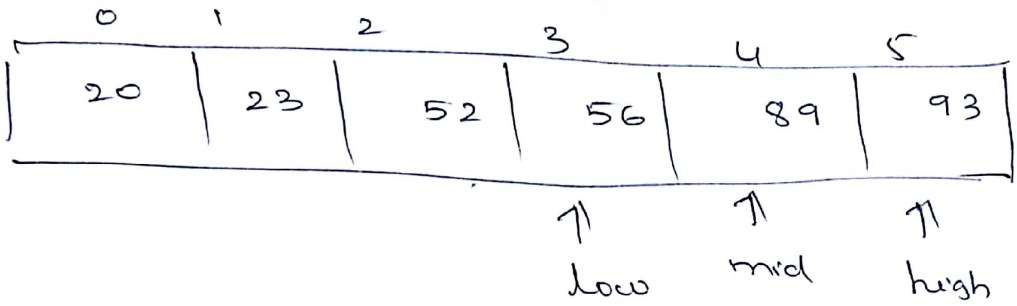
| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 20 | 23 | 52 | 56 | 89 | 93 |

↑ low ↑ mid ↑ high

→ the key element is not matched with middle element

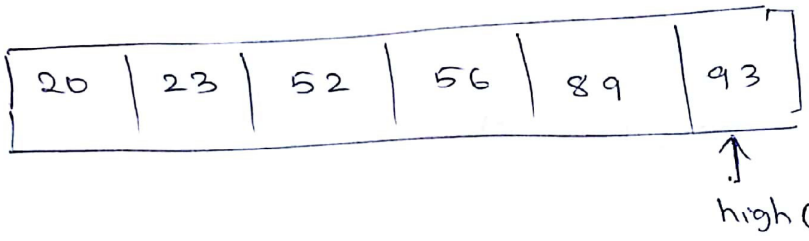
∵ 52 < 93, so the searching is moved to right side of the list.

Step2




∵ 89 < 93 (moved to right side part of mid element)

Step 3




∵ Key is found.

Binary search:-

```
#include <stdio.h>
#include <conio.h>
main()
{
 int A[5] = {10, 20, 30, 40, 50};
 int key = 50, flag = 0;
 int low = 0; high = 4;
 while (low < high)
```

```
{
    mid = (low+high)/2;
    if(key == A[mid])
    {
        flag = 1;
        break;
    }
    else if (key < A[mid])
        high = mid - 1;
    else
        low = mid + 1;
    }
    if(flag == 1)
    printf ("Key found");
    else
    printf ("Key not found");
}
```

Bubble sort :-

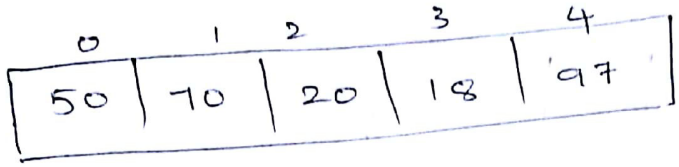
→ The simplest and the oldest sorting technique is bubble sort.

→ The method takes two elements at a time, it compares these two elements, if the first element is less than the second element, they are left undisturbed, if the first element

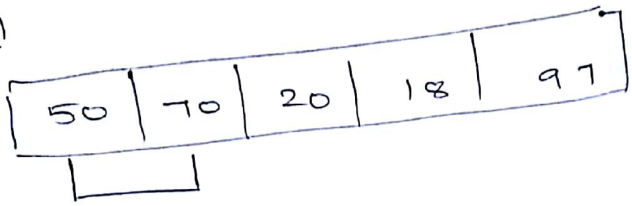
.:. greater than the second element then they are swapped.

→ The procedure continues with the next two elements goes and ends when all the elements are sorted.

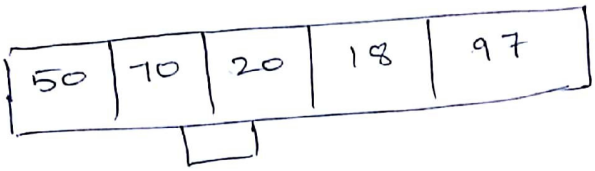
Eg:-



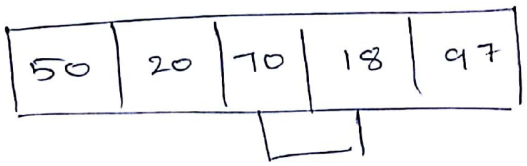

Pass 1



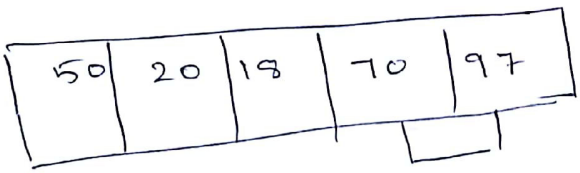

∵ 50 < 70
(No need to swap)




∵ 70 > 20
(need to swap)

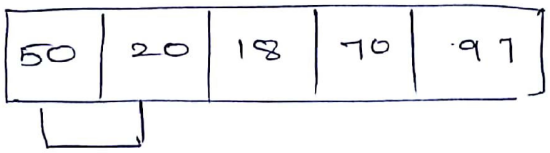



∵ 70 > 18
(need to swap)

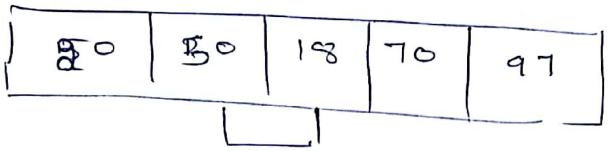



∵ 70 < 97
(no need to swap)

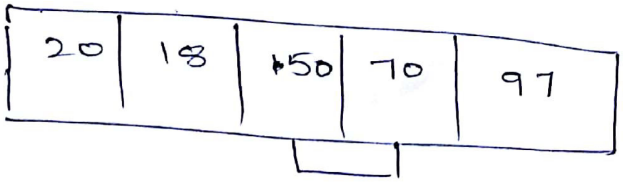
Pass 2



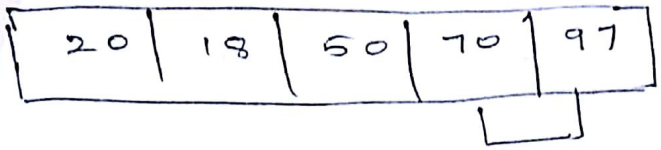

∵ 50 > 20
(need to swap)




∵ 50 > 18
(need to swap)

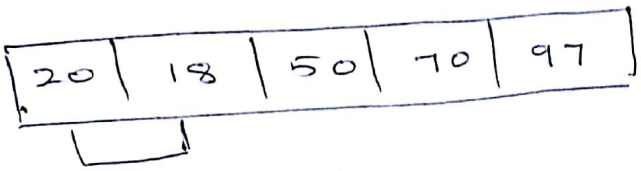



∵ 50 < 70
(no need to swap)

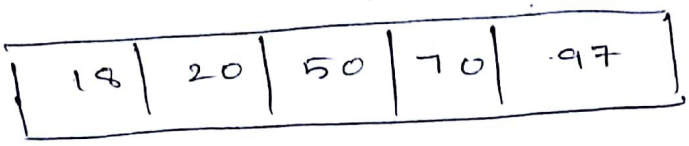



∵ 70<97
(no need to swap)

pass 3




∵ 20>18
(need to swap)




∴ finally elements are in ascending order.

Bubble sort:-

```
#include<stdio.h>
#include<conio.h>
Void main()
{
  int i,j, temp=0;
  for(i=0; i<n; i++)
  {
   for(j= i+1; j<n; j++)
   {
    if(arr[j]>arr[j+1])
    {
      temp =arr[j];
      arr[j] =arr[j+1];
      arr[j+1] =temp;
    }
   }
  }
```

# Data structures, Stacks and Queues

1.A letter means push and an asterisk means pop in the following sequence. Give the sequence of values returned by the pop operations when this sequence of operations is performed on an initially empty LIFO stack.

```
E A S * Y * Q U E * * * S T * * * I O * N * * *
```

2.Suppose that an intermixed sequence of push and pop operations are performed. The pushes push the integers 0 through 9 in order; the pops print out the return value. Which of the following sequences could not occur?

```
(a)  4 3 2 1 0 9 8 7 6 5
(b)  4 6 8 7 5 3 2 9 0 1
(c)  2 5 6 7 4 8 9 3 1 0
(d)  4 3 2 1 0 5 6 7 8 9
```

3. STACK push returns 0 if the stack was full, 1 otherwise (requires changing the interface); STACK pop returns NULL item if the stack was empty (requires definition of NULL item in Item.h).

```
int STACKpush(Item item)
  { if (N == maxN) return 0;
    s[N++] = item;
    return 1; }
Item STACKpop()
  { if (N == 0) return NULLitem; else return s[--N]; }
```

4. A letter means put and an asterisk means get in the following sequence. Give the sequence of values returned by the get operation when this sequence of operations is performed on an initially empty FIFO queue.

```
E A S * Y * Q U E * * * S T * * * I O * N * * *
```

5.An implementation of a queue Q, using two stacks S1 and S2, is given below:

```
void insert(Q, x) {
   push (S1, x);
}
```

```
void delete(Q){
   if(stack-empty(S2)) then
      if(stack-empty(S1)) then {
          print("Q is empty");
          return;
      }
      elsewhile(!(stack-empty(S1))){
          x=pop(S1);
          push(S2,x);
      }
   x=pop(S2);
}
```

6.Consider the following operation along with Enqueue and Dequeue operations on queues, where k is a global parameter.

```
MultiDequeue(Q){
   m = k
   while (Q is not empty and m  > 0) {
      Dequeue(Q)
      m = m - 1
   }
}
```

What is the worst case time complexity of a sequence of n MultiDequeue() operations on an initially empty queue?

(A) $\Theta(n)$
(B) $\Theta(n+k)$
(C) $\Theta(nk)$
(D) $\Theta(n^2)$

7.Suppose implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack?

a) A queue cannot be implemented using this stack.

b)A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.

c)A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.

d)A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

8.Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element at the head of the queue Q **without** removing it from Q. Similarly Top(S) returns the element at the top of S **without** removing it from S. Consider the algorithm given below.

```
while Q is not Empty do
    if S is Empty OR Top(S) ≤ Head(Q) then
        x := Dequeue(Q);
        Push(S,x);
    else
        x := Pop(S);
        Enqueue(Q,x);
    end
end
```

The maximum possible number of iterations of the while loop in the algorithm is_____.?

9.**The five items:A,B,C,D, and E are pushed in a stack, one after the other starting from A.The stack is popped four times and each element is inserted in a queue.Then two elements are deleted from the queue and pushed back on the stack.Now one item is popped from the stack.**
**The popped item is**----------?

10.Using
**Pop(S1, Item)**
**Push(S1,Item)**
**Read(Item)**
**Print(Item)**
the variables S1 represents (stack) and Item are given the input file:
**A,B,C,D,E,F <EOF>**
Which stacks are possible:

A)
5 A
4 B
3 C
2 D
1 E

B)
5
4
3 D
2 A
1 F

C)

5
4
3 F
2 D
1 B

D)

5
4
3 C
2 E
1 B

**11.**Following sequence of operations is performed on a stack
**push(1),push(2),pop, push(1),push(2)pop,pop,pop,push(2),pop.**
The sequence of poped out values are_____?
12.
**Stack A has the entries a,b,c(with a on top).**
**Stack B is empty.An entry popped out of stack A can be printed immediately or pushed to stack B.An entry popped out of the stack B can only be printed. In this arrangement, which of the following permutations of a,b,c are not possible?**

13.**If the sequence of operations- push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop on a stack, the sequenceof popped out values are_____?**

# Arrays, Searching & Sorting

1) Which of the following statements are correct about an array?

**1: The array int num[26]; can store 26 elements.**
**2: The expression num[1] designates the very first element in the array. 3: It is necessary to initialize the array at the time of declaration.**
**4: The declaration num[SIZE] is allowed if SIZE is a macro. A: 1 B: 1,4**
**C:2,3 D: 2,4**

2) Which of the following statements are correct about 6 used in the program?

**A. In the first statement 6 specifies a particular element, whereas in the second statement it specifies a type.**
**B. In the first statement 6 specifies a array size, whereas in the second specifies a particular element of array.**
**C. In the first statement 6 specifies a particular element, whereas in the second statement it specifies a array size.**
**D. In both the statement 6 specifies array size.**

3) Which of the following statements are correct about the program below?

```
1.  #include<stdio.h>
2.  int main() 3.  {
4.          int size, i;
5.          scanf("%d", &size);
6.          int arr[size];
7.          for(i=1; i<=size; i++)
8.          {
9.                  scanf("%d", arr[i]);
10.                 printf("%d", arr[i]);
11.         }
12.         return 0;
13. }
```

**A. The code is erroneous since the subscript for array used in for loop is in the range 1 to size.**
**B. The code is erroneous since the values of array are getting scanned through the loop.**
**C. The code is erroneous since the statement declaring array is invalid.**
**D. The code is correct and runs successfully.**

4) Which of the following statements mentioning the name of the array begins DOES NOT yield the base address?

**1: When array name is used with the sizeof operator. 2: When array name is operand of the & operator.**
**3: When array name is passed to scanf() function. 4: When array name is passed to printf() function.**

**A. 1 B. 1,2**
**C. 2 D. 2,4**

5) What will happen if in a C program you assign a value to an array element whose subscript exceeds the size of array?

**A. The element will be set to 0.**
**B. The compiler would report an error.**
**C. The program may crash if some important data gets overwritten.**
**D. The array size would appropriately grow.**

6) In C, if you pass an array as an argument to a function, what actually gets passed?

**A. Value of elements in array**
**B. First element of the array**
**C. Base address of the array**
**D. Address of the last element of array**

7) What will be the output of the program ?

```
1. #include<stdio.h>
2. int main() 3.  {
4.       int a[5] = {5, 1, 15, 20, 25};
5.       int i, j, m;
6.       i = ++a[1];
7.       j = a[1]++;
8.       m = a[i++];
9.       printf("%d, %d, %d", i, j, m);
10.      return 0;
11. }
```

**A. 2, 1, 15**
**B. 1, 2, 5**
**C. 3, 2, 15**
**D. 2, 3, 20**

8) Does this mentioning array name gives the base address in all the contexts?

**A. Yes**
**B. No**

9) Is there any difference in the following declarations?

**A. Yes**
**B. No**

15) An array elements are always stored in _memory locations.

**A. Sequential**
**B. Random**
**C. Sequential and Random**
**D. None of the above**

16) What is the maximum number of dimensions an array in C may have?

**A. 2**
**B. 8**
**C. 20**
**D. 50**
**E. Theoratically no limit. The only practical limits are memory size and compilers**

17) Size of the array need not be specified, when

**A. Initialization is a part of definition**
**B. It is a declaratrion**
**C. It is a formal parameter**
**D. All of these**

18) What will be printed after execution of the following code?

```
1.  void main()
```

```
2.      {
3.          int arr[10] = {1,2,3,4,5};
4.          printf("%d", arr[5]); 5.     }
```

**A. Garbage Value**
**B. 5**
**C. 6**
**D. 0**
**E. None of these**

19) The worst case occur in linear search algorithm when …….
**A. Item is somewhere in the middle of the array**
**B. Item is not in the array at all**
**C. Item is the last element in the array**
**D. Item is the last element in the array or item is not there at all**

20) If the number of records to be sorted is small, then …… sorting can be efficient.
**A. Merge**
**B. Heap**
**C. Selection**
**D. Bubble**

22) Which of the following is not a limitation of binary search algorithm?
**A. Must use a sorted array**
**B. Requirement of sorted array is expensive when a lot of insertion and deletions are needed**
**C. There must be a mechanism to access middle element directly**
**D. Binary search algorithm is not efficient when the data elements more than 1500.**

23) The Average case occurs in linear search algorithm …….
**A. when item is somewhere in the middle of the array**
**B. when item is not the array at all**
**C. when item is the last element in the array**
**D. Item is the last element in the array or item is not there at all**

24) Binary search algorithm cannot be applied to …
**A. sorted linked list**
**B. sorted binary trees**
**C. sorted linear array**
**D. pointer array**

25) Complexity of linear search algorithm is ……
**A. O(n)**
**B. O(logn)**
**C. O(n2)**
**D. O(n logn)**

26) The complexity of bubble sort algorithm is …..
**A. O(n)**
**B. O(logn)**
**C. O(n2)**
**D. O(n logn)**

27) Where is linear searching used?
**A. When the list has only a few elements**
**B. When performing a single search in an unordered list**
**C. Used all the time**
**D. Both A and B**

28) What is the best case for linear search?

A. **O(nlogn)**
B. **O(logn)**
C. **O(n)**
D. **O(1)**

29) What is the worst case for linear search?

A. **O(nlogn)**
B. **O(logn)**
C. **O(n)**
D. **O(1)**

**30)** Which of the following is a disadvantage of linear search?

A. **Requires more space**
B. **Greater time complexities compared to other searching algorithms**
C. **Not easy to understand**
D. **All of the mentioned**

**31)** Finding the location of a given item in a collection of items is called ……

A. **Discovering**
B. **Finding**
C. **Searching**
D. **Mining**

**1.** Write a C program for finding the Largest and Smallest Numbers in an Array.
**2.** Write a C program for Printing the Elements of an Array.
**3.** Write a program that reads in a sequence of characters and print them in reverse order using stack?
**4.** Write a program in C to read n number of values in an array and display it in reverse order?
**5.** Write a program in C to find the second largest element in an array?
**6.** Write a program in C for 2d array of size 3*3 and print the matrix?
**7.** Write a C program to find transport of a given matrix?
**8.** Write a C program to sort elements using bubble sort algorithm?