4/07/11

Algorithm :- (1) It is a set of well-defined sequential steps for developing the solution of a particular problem.

(2) In otherwords, it is defined as a set of well-defined sequential instructions for solving a particular problem.

(3) In otherwords, it is a step-by-step procedure for developing the solution of a particular problem.

(4) In otherwords, it is a syrtematic approach / a systematic way for solving a particular problem.

(5) In otherwords, it is a way / method for developing the solution of a particular problem.

eg (1) :- Preparation of the curry

step 1 : start
step 2 : Light up the stove
step 3 : Take the barin onto the stove
4 : Pour the oil into the basin
5 : Boil the oil for a period of two minutes.
6 : Add all necessary ingredients
7 : Add vegetable pieces to the existing paste / mixture.

(1) Algorithm :-

(1) Step-by-step representation for solving a problem.

(2) It is a method of problem solving.

(3) It is a method of problem solving which is easier, effective in finding the problem solution.

(4) It is a systematic representation for solving a problem.

(5) It is a scientific method for solving a problem.

(6) The method of problem solving in which we get accurate values (or)

(7) A formula or set of steps for solving a particular problem.

(8) Algorithm is one of the method of problem solving in programming.

(9) Algorithm is the simple and require the fewest steps posible for solving a problem.

(10) A set of rules for solving a problem in a finite number of steps.

(11) A set of instructions that leads to a predictable result.

(7) It consists of basic instructions that are realizable. This means that the instructions can be performed by using the given inputs in a finite amount of time.

(3) Data type can be regarded as data structure because we can define our own data type by using typedef & function.

(4) Data types are the category in which the variable are listed to hold the specific value.

(5) Data type refers to the type of data that's being stored.

(3) Each data structu built up from the basic data types of the underlying programming language using the available data structuring facilities such as arrays & pointers.

(4) Data structures are the way in which the values and variables are stored in the memory.

(5) Data structure refer to the way data is being organized.

13/07/19

1.

15/07/14

Development of an Algorithm :-

→ The steps involved in the development of an algorithm are as follows :

(i) problem statement
(ii) model formulation (deriving a mathematical descriptions)
(iii) algorithm design
(iv) algorithm correctness
(v) implementation
(vi) Algorithm analysis — Time, space } practically
(vii) Program testing
(viii) Documentation

ADT – an illustration :-

→ A data type refers to the types of values that variables in a programming language hold. Thus, the data types of integer, real, character, Boolean (T,F) which are inherently provided in programming languages are referred to as primitive data types.

→ A list of elements is called as data objects.

For example, w
integers or
data objects.
I = { 1, 2, 3,
s = { "a", "

→ An abstrac
n of the
collection of
(i) Name of
(ii) Types re
data
(iii) Functions
data.
In other w
set of data
domain a
operations.
Data objec
set of all
operations
Addition
into Lec
ADD (
check if
CHECK_P

IV. Generality
V Inputs
VI Outputs
VII Efficiency

I. Finiteness :- Each and every Algorithm must be terminated/ended in a finite amount of time. In other words, the number of steps of each and every Algorithm is finite.

II. Definiteness :- Each and every step of the algorithm must be precisely defined (or) clearly defined.

In other words, each and every step of the Algorithm will be defined with clarity and without having any sort of ambiguity/confusion/inconsistency.

III. Effectiveness :- Each and every step of the algorithm must have its own effectiveness.

IV. Generality :- The Algorithm must be generic enough to solve all problems of a particular class/field/domain/area.

V. Inputs :- An algorithm must have certain initial inputs and as well as some outputs/outcomes

VII. Efficienc
its own e
measure t
I. Time
II. Space
Time com
running
algorithm
Space comp
which is
in the
structure
An
1) Outpu
2) Assig
3) Deci
4) Repe
5) Out

Properties :-

(1) An algorithm contains a beginning point and as well as a stopping / ending / terminating point

(2) The problem solution is done in a finite time.

(3) It contains a set of inputs and outputs.

(4) It terminates after a finite number of steps.

(5) Each step in algorithm is unambiguous. This means the action specified by the step cannot be interrupted in multiple ways & can be performed with any confusion.

(6) It accepts zero (or) more inputs and gives atleast one output.

Difference between Datatype and Datastructure:-

Data type	Data structure
(1) Integer, character, Double, float are the data types.	(1) Arrays, LinkList, stacks, Queues, Trees are Data structure.
(2) Data type is the description on a field that determines what kind of information you can enter in the field - Text, Memo and Number.	(2) Data structure is the way data is encoded by a computer. Data structure is not usually the concern of the user

(3) Data type can be regarded as data structure because can define our data type by us typedef & funct

(4) Data types are t category in which the variable are listed to hold the specific value.

(1) Data type refers the type of data that's being store

13/07/19

1.

For example, we could have a list of integers or list of alphabetical strings as data objects.

I = {1, 2, 3, 4, 5, 6, 9, 8, 9}

S = {"a", "boy", "can"}

→ An abstract data type (ADT) is a description of the operations performed on a collection of data, and it has 3 components

(i) Name of the ADT.

(ii) Types represented in the collection of data

(iii) Functions/operations that operates on the data.

In other words, an ADT is defined as a set of data objects D defined over a domain L and supporting a list of operations O:

Data objects:-

set of all the integers D.

Operations:-

Addition of the integers INT1 and INT2 into result

ADD (INT1, INT2, RESULT)

Check if a number INT1 is a the number

CHECK_POSITIVE (INT1) (BOOLEAN FUNCTION)

Average, Best and Worst Case Complexities :-

→ That input instance (or instances) for which the algorithm takes the maximum possible time is called the worst case and the time complexity in such a case is referred to as worst case time complexity.

→ That input instance for which the algorithm takes the minimum possible time is called as the best case and the time complexity in such a case is referred to as best case time complexity.

→ All the other input instances which are neither of the two are categorized as average cases and the time complexity of the algorithm in such a case referred to as average case complexity.

21/07/14

*Recursion : ① A Recursion is a process-method-way by which a function calls itself repeatedly until a certain condition is satisfied.

② Incase of the recursion, the names of the calling function as well as called functions are same.

③ Recursion is used inorder to perform

Apriori Analysis :-

Apriori estimation is interested in the following for the computation of efficiency
(i) the number of times the statement is executed in the program, known as the frequency count of the statement, and
(ii) the time taken for a single execution of the statement

Program segment
A

```
....
x = x+2;
....
```

Program statement	Frequency count
x = x+2;	1
Total frequency count	1

Analyzing Running Time :-

T(n), or the running time of a particular algorithm on input of size n, is taken to be the number of times the instructions in the algorithm are executed. Pseudo code algorithm illustrates the calculation of the mean (average) of a set of n numbers.

1. n = read input from user
2. sum = 0
3. i = 1
4. while i <= n
5. number = read from u
6. sum = sum + num
7. i = i+1
8. mean = sum/n

The computing terms on input

Order of Magn

The order of
is the sum o
statements co

Example :-

```
for(i=1; i<=n;
{
}
```

Assume there
inside the lo
takes 1 uni

18/07/11

Efficiency of Algorithms:-

→ the performance of algorithms can be measured on the scales of time and space.

→ the time complexity of an Algorithm or a program is a function of the running time of the algorithm/program.

→ The space complexity of an algorithm or a program is a function of the space needed by the algorithm or program to run to completion.

→ The empirical or posterfori testing approach calls for implementing the complete algorithms and executing them on a computer for various instances of problem

→ the apriori/theoritical approach calls for mathematical determining the resources such as time and space needed by the algorithm, as a function of a parameter related to the instances of the problem considered.

statement	No. of times executed
1. n = read input from user	1
2. sum = 0	1
3. i = 1	1
4. while i <= n	n+1
5. number = read input from user	n
6. sum = sum + number	n
7. i = i+1	n
8. mean = sum/n	1

The computing time for this algorithm in terms on input size n is : $T(n) = 4n+5$

Order of Magnitude of an algorithm :-

The 'order of magnitude' of an algorithm is the sum of number of occurences of statements contained in it.

Example 1 :-

```
for(i=1; i<=n; i++)
{
    c
}
```

c * i = c * n

Assume there are 'c' number of statements inside the loop and Each statement takes 1 unit of time

Function name	Domain of definition (arguments)	Range (return type)
SQR CUBE	positive integer	positive integer
CUBE SQR	Real	Real.

DATA STRUCTURES :-

→ The design of an efficient algorithm for the solution of problem calls for the inclusion of appropriate data structures.

→ The subject of data structures is intrinsically connected with the design and implementation of efficient algorithms. Data structures deals with the study of methods, techniques and tools to organize or structure data

→ Data structure can be defined as a mathematical or logical way of organizing a data. An other words, it is a method / way of organizing the data.

18/07/18

Efficiency

→ The perfo
measured

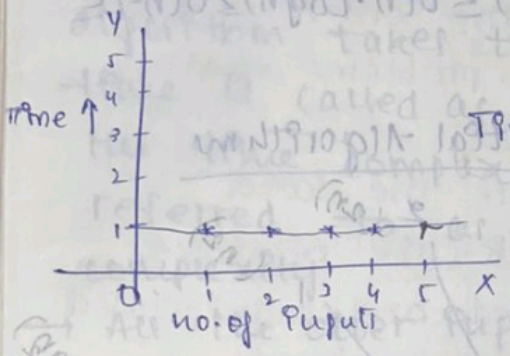
→ The time
program
of the

→ The spa
a progra
by the a
completio

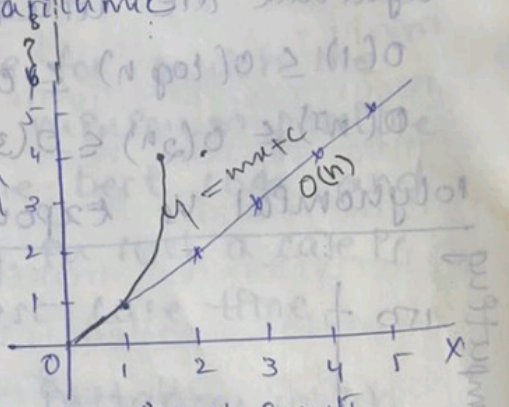
→ The emp
calls for
algorithm
computer
problem

→ The ap
mathem
such as
algorith
related
consider

$T(n) = O(1)$ – constant (Nature)
$T(n) = O(n)$ – linear
$T(n) = O(n^2)$, $T(n) = O(n^3)$ } – quadratic
$T(n) = O(k^n)$, $T(n) = O(n^k)$ } exponential / polynomial
$T(n) = O(\log_2 4)$, $T(n) = O(n \log_2 n)$ } logarithmic



$T(n) = O(1)$



$T(n) = O(n)$

Time complexity of an Algorithm Using O Notation:

→ Algorithms reporting $O(1)$ time complexity indicate constant running time.

→ The time complexities of $O(n)$, $O(n^2)$ and $O(n^3)$ are called linear, quadratic and cubic time complexities respectively. $O(\log n)$ time complexity is referred to as logarithmic.

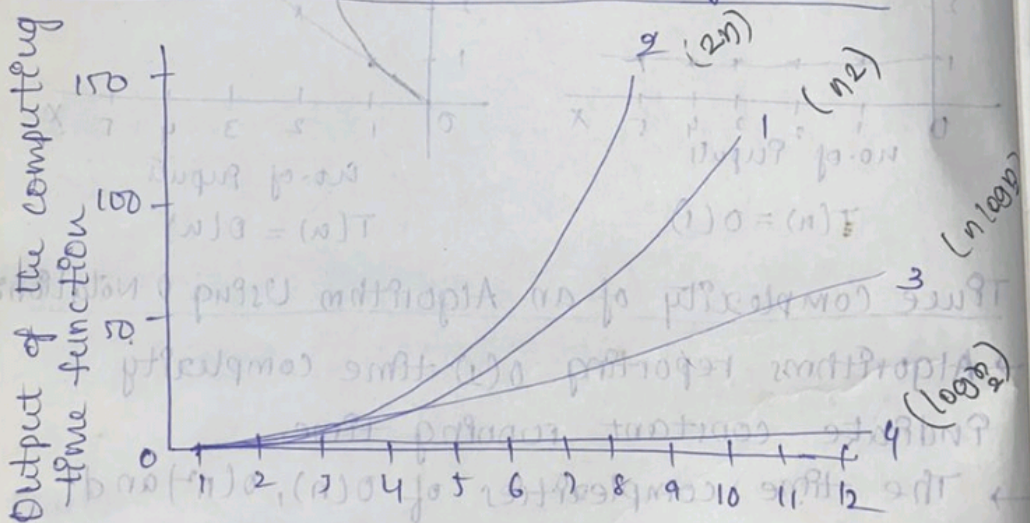
→ In general, time complexities of the type, $O(n^k)$ are called polynomial time complexities.

→ Time complexities such as $O(2^n)$, $O(3^n)$ in general $O(k^n)$ are called as exponential time complexities.

→ Some of the commonly occuring time complexities in their ascending orders of magnitude are listed below:

$O(1) \leq O(\log n) \leq O(n) \leq O(n \cdot \log n) \leq O(n^2) \leq O(n^3) \leq O(2^n) \leq O(3^n)$

Polynomial Vs Exponential Algorithms



1: n^2 2: 2^n 3: $n \cdot \log n$ 4: $\log_2 n$

Average,

→ that sup
the algo
time &
time c
referre
complex

→ that
algorith
time
the t
refer
comple

→ All
are u
as a
of m
to a

21/07/11

*Recur
way
repea
satisf

② An
Calle
are

③ Re

Execution time for 1 loop = c * 1 = c
Total execution time = n * c
Since 'c' is constant, n is the order.

Example 2

```
for(i=1; i<=n; i++)
{
  for(j=1; j<=m; j++)
  {
    ;
  }
}
```

Assume we have 'c' number of statements inside the loop and following the same assumptions as the earlier example.

Execution time for 1 loop = c * 1

Execution time for the inner loop = m * c

Total execution time = n * (m * c)

Since c is a constant, the total execution time = m * n.

The most common asymptotic notations:-

→ 'Big Oh' ('O') notation: It represents the upper bound of the resources required to solve a problem. It is represented by 'O'.

'Omega' no
bound of
problem.
Definition

we
(Read 'f
big 'oh'
integer
positive
Assumpti
the le
highest
'n' are
ex:- T(n)
Represe
is T(n)
Example

→ In th
up w
that
grows
we c

→ To s

VII. Efficiency :- Each and every algorithm has its own efficiency where, efficiency is measure by making use of two parameterr.

I. Time complexity

II. Space complexity.

Time complexity :- It is defined as a running time / execution time of an algorithm / program.

Space complexity :- It is defined as a space which is occupied by an algorithm / program in the memory of a computer.

Structure of an Algorithm :-

1) An Output step
2) Assignment step
3) Decision step.
4) Repetative step
5) Output step

'Omega' notation :- It represents the lower bound of the resources required to solve a problem. It is represented by Ω.

Definition: Let f(n) and g(n) be two functions.

we write $f(n) = \Omega(g(n))$ or $f = \Omega(g)$ (Read "f of n is big oh of g of n" or "f is big oh of g") if there is a positive integer c such that $f(n) <= c * g(n)$ for all positive integers $n >= n_0$. ($n >= n_0$), $c > 0$, $n_0 >= 1$

Assumption :-

the leading constants and coefficients of highest power of 'n' and all low powers of 'n' are ignored in f(n).

ex :- $T(n) = \Omega(100n^3 + 29n^2 + 19n)$

Represented the same in big oh notation is $T(n) = \Omega(n^3)$.

Example 1 :-

→ In the previous timing analysis, we ended up with $T(n) = 4n + 5$ and we concluded that $T(n) = \Omega(n)$ because the running time grows linearly as n grows. Now, however, we can prove it mathematically.

→ To show that $f(n) = 4n + 5 = \Omega(n)$, we need

to produce a constant c such that:

$f(n) <= c*n$ for all n.

→ If we try $c = 4$, this doesn't work because $4n+5$ is not less than $4n$. We need c to be at least 9 to cover all n. If $n=1$, c has to be 9, but c can be smaller for greater values of n (if $n=100$, c be 8, ~~xxxx~~ ~~xxxx~~ ~~xxxxxxxxx~~ ~~xx~~ 5).

Since the chosen c must work for all n, we must use 9.

$4n + 5 <= 4n + 5n = 9n$

→ Since we have produced a constant c that works for all n.

eg: $T(n) = f(n) = 2n^2 - n$, $g(n) = n^2$

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

$n^2 - n \quad c * n^2$

Case (i):-

If $n=1$ $0 <= 0$.

$c = 0$

If $n=2$ $2^2 - 2 \leq 1 * 4$

$c = 1$

$2 < 4$

8 : Add some amount of chilli powder to the existing paste

9 : Add some amount of salt to the existing paste/mixture inorder to get taste/effect.

10 : Finally delicious/tastable curry is ready

11 : stop

eg(2) :- Algorithm for finding a average of 3 numbers.

1 : start

2 : Read some values for a, b and c

3 : Perform sum operation by making use of a, b and c

4 : store the result/sum in d.

5 : Perform the division operation with d and 3 (d/3)

6 : store the average in e

7 : Display/print the average by making use of e.

8 : stop

Properties/Attributes/characteristics of an Algorithm :-

The following are the properties of an algorithm

I) Finiteness

II. Definiteness

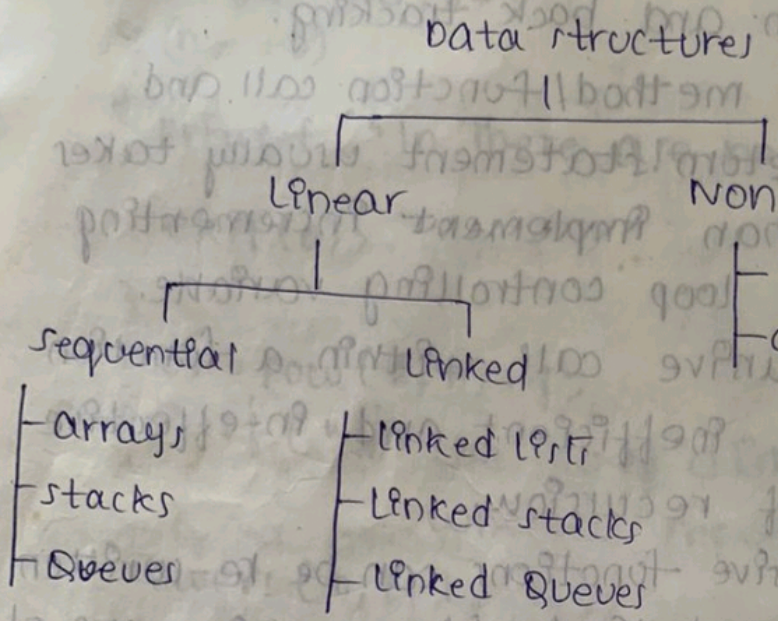
III. Effectiveness

Types of data structures :-

Data structures are classified into mainly two types

I. Linear

II. Non-linear



Implementation of Data structures :-

Implementation of Data structures can be done in two ways.

I) Data structures can be implemented by making use of arrays.

II) Data structures can be implemented by making use of pointers/links/addresses/references.

Operations to be performed on data structures :-

(1) There are basically 6 operations which can be performed on data structure.

(i) Inserting :- The process of adding a new element into the data structure / list / table.

(ii) Deleting :- The process of removing an element / data item from a list / data structure / table.

(iii) Searching :- The process of finding a desired element / data item from a given list / table / data structure by making use of a "key" value.

(iv) Sorting :- The process of arranging all the elements of a given list in some logical fashion (The logical fashion may be ascending order or descending order and even the alphabatical order.

(v) Traversing :- The process of accessing the elements of record / exactly once.

(vi) Merging :- The process of combining two sorted lists / files into another data structure.

SEARCHING

Introduction :-

(i) Searching is one of the most important applications of computer science.

(ii) It is a process / method of finding desire record / data item / data element from a table / list, by making use of a key information.

(3) We have so many searching mechanisms in order to find out the desired record/ data item/data element/piece of information from a given table/list.

(4) But mainly we have 3 searching mechanisms/methods.

(i) Linear search

(ii) Binary search

(iii) Fibonacci search

I. Linear search:-

(1) It is very simplest and earliest method.

(2) In this linear search method, we start at the beginning of the list/table and search for a desired record/data item/ data element by examining each and every subsequent data item/record/data element, until the list is desired element is found (or) the list is completed/exhausted, by making using of the key element/value.

1: write a program for implementing the linear search method.

```
#include <stdio.h>
#include <conio.h>
int a[20], n, i, key, pos=-1;
lsearch(int a[], int n, int key);
main()
{
  clrscr();
  printf("\n Enter the required size of an
                                    array");
  scanf("%d", &n);
  /* Reading the ele */
   printf("\n Enter the elements");
   for(i=0; i<n; i++)
   scanf("%d", &a[i]);
   /* printing the elements */
  printf("\n the elements are ar-----");
  for(i=0; i<n; i++)
  printf("%5d", a[i]);
  printf("\n Enter a value for key");
  scanf("%d", &key);
  lsearch(a, n, key);
  getch();
}
lsearch(int a[], int n, int key)
{
  for(i=0; i<n; i++)
  {
```

```
    if(a[i]==key)
    {
      pos=i;
      printf("\n the desired ele %d is found
            and its pos %d is", a[i], pos);
      break;
    }
  }
  if(pos==-1)
  printf("\n the desired ele %d is not
                found", key);
}
```

Linear search recursion using recursion :-

```
#include<stdio.h>
#include<conio.h>
int a[20], n, i, key, pos=-1;
int rlsearch(int a[], int n, int key);
main()
{
  clrscr();
  printf("\n Enter the req size of an
  scanf("%d", &n);                array");
  /* Reading the elements */
  printf("\n Enter elements");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
```

```
/* printing the elements */
printf ("\n the elements are");
for (i=0; i<n; i++)
printf ("%5d", a[i]);
printf ("\n Enter a value for key element");
scanf ("%d", &key);
pos = rlsearch (a, n, key);
if (pos == -1)
printf ("\n the ele %d is not found", key);
else
printf ("\n the ele %d and its position %d",
                                key, pos);
getch ();
}
Int rlsearch (Int a[], Int n, Int key)
{
  if (n==0)
  {
    return (-1);
  }
  else
  {
    if (a[n-1] == key)
    return (n-1);
      return (rlsearch (a, n-1, key));
  }
}
```

Key comparisons of linear search:-

(i) Best Case:-

The total no. of comparisons which are required in the best case is 1.

(ii) The total no. of comparisons which are required in the average case is

$$\frac{1+2+3+\cdots+n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}.$$

(iii) The total no. of comparisons which are required in the worst case is 'n'.

Note:- where 'n' is the size of the list/table

Drawback of the linear search:-

It is ineffective and inefficient when 'n' is large / when the list is very big.

30/07/11

Binary search:- (1) In this method, the length of the list is reduced by half at each time.

In otherwords, the list is divided into two parts at each step.

(2) In case of the binary search method, all the elements should be arranged either in some logical fashion (The logical fashion may be ascending order or descending

(3) In thi
in the
when co
(4) At i
compared
(5) The
required
$\log_2 n$,

Pr:- Write
binary
#inclu
#inclu
int a[2
bsearch
main()
{
clrs
pri
sca
hig
/*
pr
fo
s
i
pr
fo
pr

descending order).

③ In this method, the no. of key comparisons in the worst case scenario, will be reduced, when compared to the linear search.

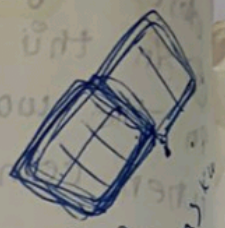
④ It is effecient and effective when compared to the linear search.

⑤ The no. of key comparisons which are required in the worst case scenario is $\log_2 n$, where n is the size of the list/table.

Pr:- Write a program for implementing the binary search.

```
#include<stdio.h>
#include<conio.h>
int a[20], i, n, key, mid, pos=-1, low=0, high;
bsearch(int a[], int low, int high, int key);
main()
{
  clrscr();
  printf("\n Enter the req size of an array");
  scanf("%d",&n);
  high = n-1;
  /* Reading */
  printf("\n Enter the ele");
  for(i=0; i<n; i++)
  scanf("%d",&a[i]);
  /* printing the ele */
  printf("\n the elements are");
  for(i=0; i<n; i++)
  printf("%5d",a[i]);
```

```
printf("\nEnter the value for key");
scanf("%d", &key);
bsearch(a, low, high, key);
getch();
}
bsearch(int a[], int low, int high, int key)
{
  while(low<=high)
  {
    mid=(low+high)/2;
    if(a[mid]==key)
    {
      pos=mid;
      printf("\n the desired ele %d is found
              and its position %d", a[mid], pos);
      break;
    }
    else if(a[mid]>key)
    {
      high=mid-1;
    }
    else
    {
      low=mid+1;
    }
  }
  if(pos==-1)
  printf("\n the desired ele %d is not
          found", key);
}
```



Pr: write a
search
#include
#include
int a[20
int bse
main()
{
cl
p
}

Pr: Write a program for implementing the binary search method by making use of Recursion.

```
#include<stdio.h>
#include<conio.h>
int a[20], i, n, key, low=0, mid, high, pos=-1;
int bsearch(int a[], int low, int high, int key);
main()
{
    clrscr();
    printf("\n enter the req size of an array");
    scanf("%d", &n);
    high=n-1;
    printf("\n enter the ele");
    for(i=0;i<n;i++)
    scanf("%d", &a[i]);
    printf("\n the ele are as");
    for(i=0;i<n;i++)
    printf("%5d", a[i]);
    printf("\n enter the key value");
    scanf("%d", &key);
    pos=bsearch(a, low, high, key);
    if(pos!=-1)
    printf("\n the ele %d is found and
                pos is %d",
            key a[mid], pos);
    else
    printf("\n the ele is %d is not found",
                key);
    getch();
}
```

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