EXPERIMENT-2

**Aim:**

Implement different types of searching techniques on a given list

(i) Sequential search

(ii) Binary search

**Description:**

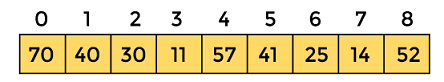
Searching in computer science refers to finding an item with specified properties among a collection of items using a search algorithm or search technique. The searching algorithms vary from one data structure to another. When a list of data elements is represented as a linear list, the search algorithms like sequential search, transpose sequential search, binary search, Fibonacci search, etc., are applicable.

**1. Linear/Sequential Search:**

Linear search or sequential search is a method for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is exhausted. The list need not be ordered.

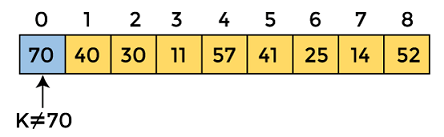
**Example:**

Let the elements of array are -

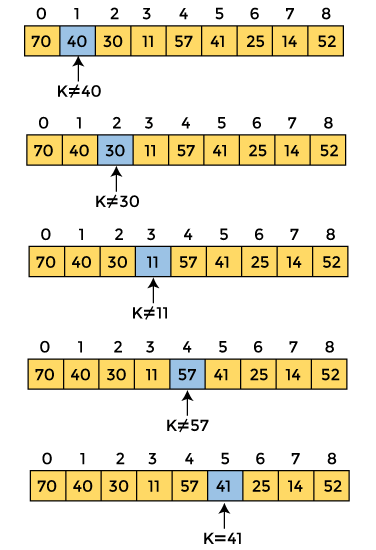


Let the element to be searched is **K = 41**

Now, start from the first element and compare **K** with each element of the array.



The value of **K,** i.e., **41,** is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found.



Now, the element to be searched is found. So algorithm will return the index of the element matched.

**Algorithm:**

The steps used in the implementation of Linear Search are listed as follows -

* First, we have to traverse the array elements using a **for** loop.
* In each iteration of **for loop,** compare the search element with the current array element, and -
  + If the element matches, then return the index of the corresponding array element.
  + If the element does not match, then move to the next element.
* If there is no match or the search element is not present in the given array, return **-1**

**Advantages of Linear Search:**

* Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
* Does not require any additional memory.
* It is a well-suited algorithm for small datasets.

**Drawbacks of Linear Search:**

* Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
* Not suitable for large arrays.

**Applications:**

* **Phonebook Search:** Linear search can be used to search through a phonebook to find a person’s name, given their phone number.
* **Spell Checkers:** The algorithm compares each word in the document to a dictionary of correctly spelled words until a match is found.
* **Finding Minimum and Maximum Values:** Linear search can be used to find the minimum and maximum values in an array or list.
* **Searching through unsorted data:** Linear search is useful for searching through unsorted data.

**1.Perform linear search on intergers**

**Program:**

/\*Linear search on integers\*/

#include<stdio.h>

#define MAX 200

int main(){

int a[MAX];

int n;

printf("Enter the number of elements:\n");

scanf("%d",&n);

printf("Enter the elements:\n");

for(int i=0;i<n;i++){

scanf("%d",&a[i]);

}

int key;

printf("Enter the value to be searched:\n");

scanf("%d",&key);

int count=0;

for(int i=0;i<n;i++){

if(key==a[i]){

printf("Found at index %d\n",i);

count++;

break;

}

}

if(count==0){

printf("Element not found\n");

}

return 0;

}

**Output:**

**Enter the number of elements:**

**5**

**Enter the elements:**

**1**

**2**

**3**

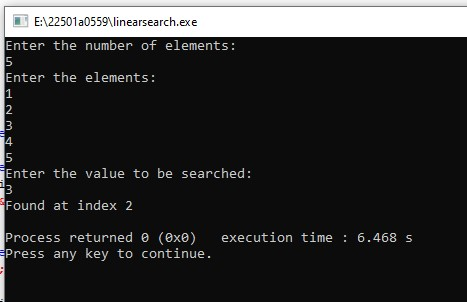
**4**

**5**

**Enter the value to be searched:**

**3**

**Found at index 2**



**Enter the number of elements:**

**4**

**Enter the elements:**

**1**

**2**

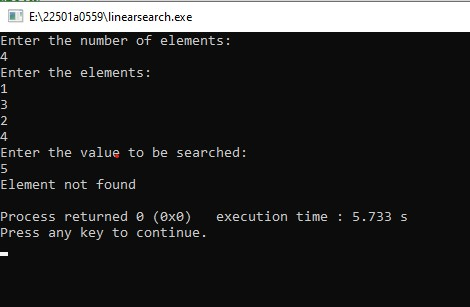
**3**

**4**

**Enter the value to be searched:**

**5**

**Element not found**



**2.Perform linear search on characters**

**Program:**

/\*Linear search with char\*/

#include<stdio.h>

#define MAX 200

int main(){

char a[]={'a','b','c','8','9'};

char key;

printf("Enter the value to be searched:\n");

scanf("%c",&key);

int count=0;

for(int i=0;i<5;i++){

if(key==a[i]){

printf("Found at index %d\n",i);

count++;

break;

}

}

if(count==0){

printf("Element not found\n");

}

return 0;

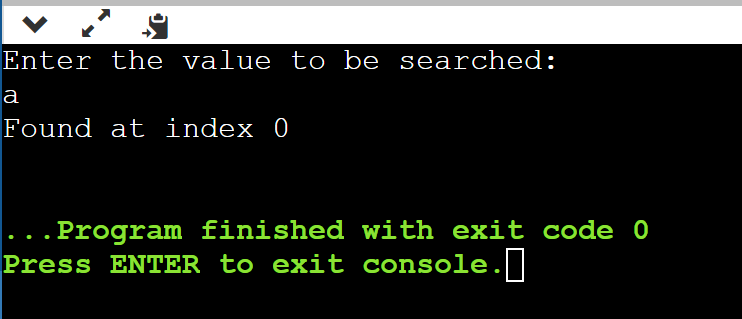
}

**Output:**

**Enter the value to be searched:**

**a**

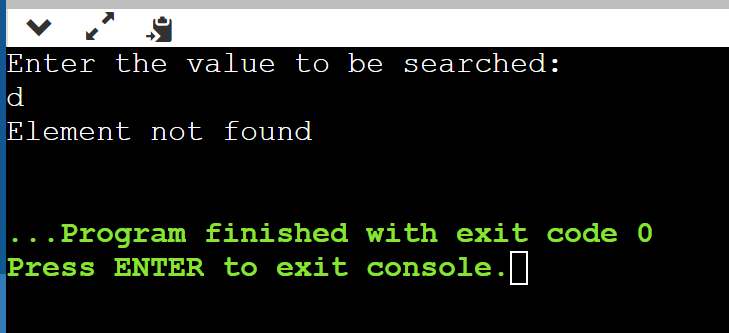
**Found at index 0**



**Enter the value to be searched:**

**d**

**Element not found**

****

**2. Binary search**

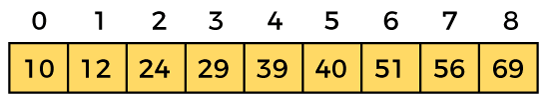
Binary search or logarithmic search or bisection search searches a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

**Algorithm:**

* Start with the middle element of the array.
* If the middle element is equal to the target element, return its index.
* If the middle element is greater than the target element, search the left half of the array.
* If the middle element is less than the target element, search the right half of the array.
* Repeat until the target element is found or there are no more elements to search.

**Example:**

Let the elements of array are -



Let the element to search is, **K = 56**

We have to use the below formula to calculate the **mid** of the array -

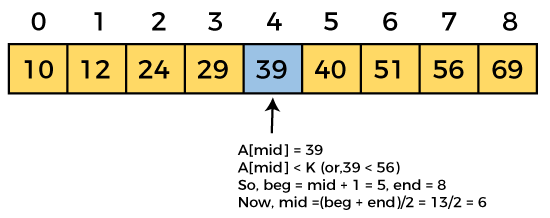
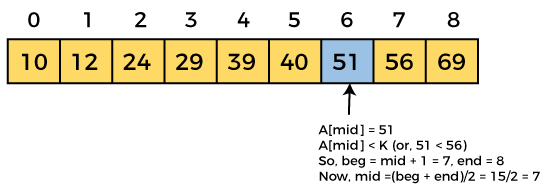
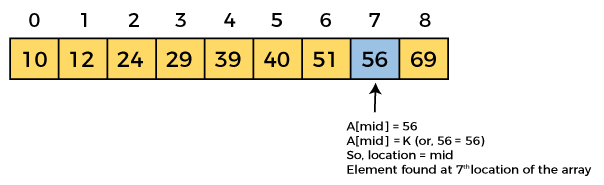
1. mid = (beg + end)/2

So, in the given array -

**beg** = 0

**end** = 8

**mid** = (0 + 8)/2 = 4. So, 4 is the mid of the array.

Now, the element to search is found. So algorithm will return the index of the element matched.

**Advantages of Binary Search:**

* Binary search is faster than linear search, especially for large arrays.
* More efficient than other searching algorithms with a similar time complexity.
* Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.

**Drawbacks of Binary Search:**

* The array should be sorted.
* Binary search requires that the data structure being searched be stored in contiguous memory locations.
* Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered.

**Applications of Binary Search:**

* Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
* It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
* It can be used for searching a database.

**1.Perform binary search on integers**

**Program:**/\*Binary search on integers\*/

#include<stdio.h>

#define MAX 200

int main(){

int a[MAX];

int n;

printf("Enter the number of elements:\n");

scanf("%d",&n);

printf("Enter the elements:\n");

for(int i=0;i<n;i++){

scanf("%d",&a[i]);

}

int key;

printf("Enter the value to be searched:\n");

scanf("%d",&key);

int count=0;

int low=0;

int high=n-1;

int mid;

while(low<=high){

mid=(low+high)/2;

if(a[mid]==key){

printf("element found at index %d",mid);

count++;

break;

}else if(key<a[mid]){

high=mid-1;

}else{

low=mid+1;

}

}

if(count==0)

printf("element not found");

return 0;

}

**Output:**

**Enter the number of elements:**

**4**

**Enter the elements:**

**1**

**2**

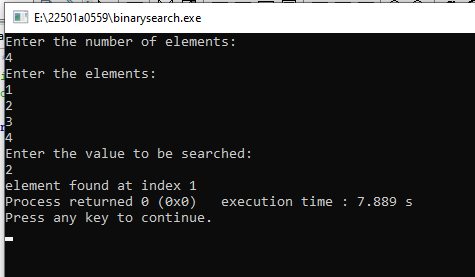
**3**

**4**

**Enter the value to be searched:**

**2**

**element found at index 1**



**Enter the number of elements:**

**4**

**Enter the elements:**

**1**

**2**

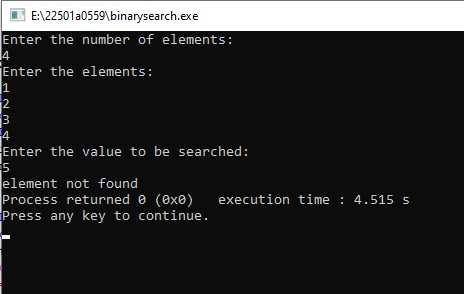
**3**

**4**

**Enter the value to be searched:**

**5**

**element not found**



**2.Perform binary search on characters**

**Program:**

/\*Binary search on characters\*/

#include<stdio.h>

#include<string.h>

#define MAX 200

int main(){

char a[MAX];

printf("Enter the string:");

gets(a);

char key;

printf("Enter the value to be searched:\n");

scanf("%c",&key);

int count=0;

int low=0;

int high=strlen(a);

int mid;

while(low<=high){

mid=(low+high)/2;

if(a[mid]==key){

printf("element found at index %d",mid);

count++;

break;

}else if(key<a[mid]){

high=mid-1;

}else{

low=mid+1;

}

}

if(count==0){

printf("element not found");

}

return 0;

}

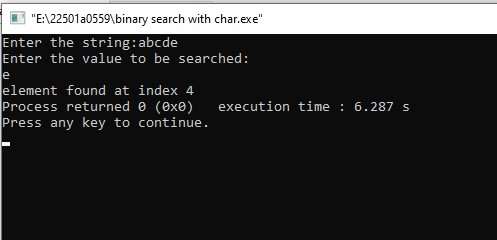
**Output:**

**Enter the string:abcde**

**Enter the value to be searched:**

**e**

**element found at index 4**

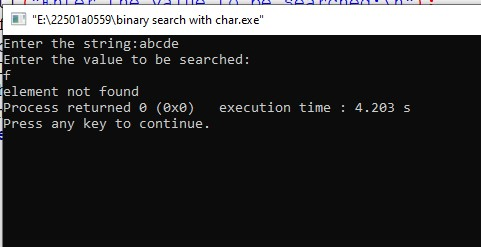


**Enter the string:abcde**

**Enter the value to be searched:**

**f**

**element not found**



**Perform binary search on strings**

**Program:**

/\*Binary search on strings\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

    char str[][100]= {"ABCD","ABcd","abCD","abcd"};

    char x[100];

    printf("Enter the element to be searched: ");

    scanf("%s",x);

    int n = sizeof(str)/sizeof(str[0]);

    int low=0,high=n-1;

int count=0;

    while(low<=high)

    {

        int mid = (low+high)/2;

        if(strcmp(x,str[mid])==0)

        {

         printf("Element found at index: %d",mid);

count++;

         exit(0);

        }

        else if(strcmp(x,str[mid])<0)

            high = mid - 1;

        else

            low = mid + 1;

    }

    if(count==0)

        printf("Element not found");

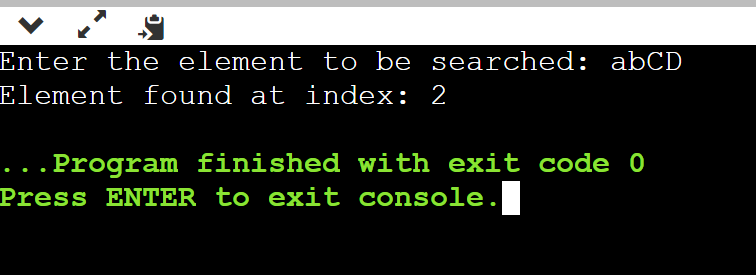
return 0;

}

**Output:**

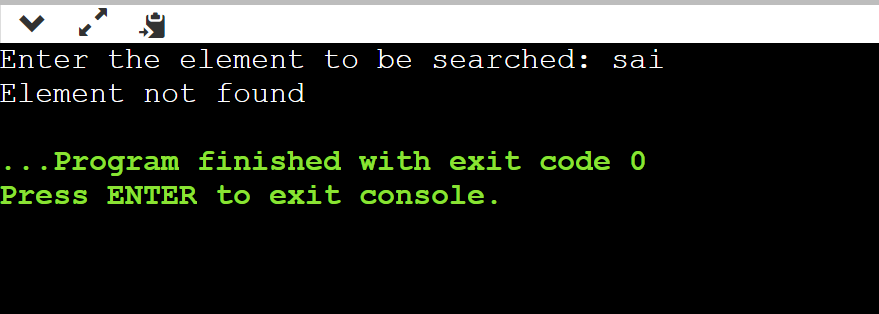
**Enter the element to be searched: abCD**

**Element found at index: 2**

****

**Enter the element to be searched: sai**

**Element not found**

****

**Difference between Linear search and Binary search:**

|  |  |  |
| --- | --- | --- |
| **Base of comparison** | **Linear search** | **Binary search** |
| **Time complexity** | **O(n)** | **O(log n)** |
| **Best case time** | **O(1) first element** | **O(1) center element** |
| **Prerequisite of an array** | **No prerequisite** | **Array must be in sorted order** |
| **Input data** | **No need to be sorted** | **Need to be sorted** |
| **Access** | **Sequential** | **Random** |

**Viva questions:**

**Linear Search:**

**What is a linear search?**

Linear search is a simple search algorithm that scans through a list of items one by one to find a specific target item.

**How does linear search work?**

Linear search starts from the beginning of the list and compares each item with the target item until a match is found or the entire list has been searched.

**What is the time complexity of linear search in the worst case?**

The worst-case time complexity of linear search is O(n), where n is the number of items in the list. This is because, in the worst case, the algorithm may need to scan through all items.

**Under what circumstances might linear search be a good choice?**

Linear search is suitable for small lists or when the list is not sorted. It's also useful when you need to find all occurrences of an item in a list.

**Binary Search:**

**What is binary search?**

Binary search is a more efficient search algorithm that works on sorted lists by repeatedly dividing the search interval in half.

**How does binary search work?**

Binary search compares the middle element of the list with the target item. If the middle element is the target, the search is successful. If the target is smaller, the search continues in the lower half of the list; if the target is larger, the search continues in the upper half.

**What is a prerequisite for using binary search?**

Binary search requires the list to be sorted in ascending (or descending) order.

**What is the time complexity of binary search?**

The time complexity of binary search in the worst case is O(log n), where n is the number of items in the list. This is because the search space is halved with each comparison.

**Why is binary search more efficient than linear search for larger lists?**

Binary search reduces the search space by half with each comparison, making it much faster than linear search for larger lists.

**Can binary search be applied to a list that is not sorted?**

No, binary search requires the list to be sorted. If the list is not sorted, binary search cannot guarantee accurate results.

**Are there any limitations to binary search?**

Binary search requires a sorted list, and it might not be the best choice if the list is frequently changing, as resorting the list can be time-consuming.

**What is the role of the middle element in binary search?**

The middle element is used to determine whether the target element is in the upper or lower half of the current search interval.

EXPERIMENT-1

**Aim:**

Demonstrate recursive algorithms with examples.

(i) Factorial of a Number

(ii) Towers of Hanoi Problem

**Description:**

A recursive algorithm calls itself with smaller input values and returns the result for the current input by carrying out basic operations on the returned value for the smaller input. Generally, if a problem can be solved by applying solutions to smaller versions of the same problem, and the smaller versions shrink to readily solvable instances, then the problem can be solved using a recursive algorithm.

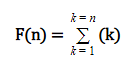
To build a recursive algorithm, you will break the given problem statement into two parts. The first one is the base case, and the second one is the recursive step.

* Base Case: It is nothing more than the simplest instance of a problem, consisting of a condition that terminates the recursive function. This base case evaluates the result when a given condition is met.
* Recursive Step: It computes the result by making recursive calls to the same function, but with the inputs decreased in size or complexity.

For example, consider this problem statement: Print sum of n natural numbers using recursion. This statement clarifies that we need to formulate a function that will calculate the summation of all natural numbers in the range 1 to n. Hence, mathematically you can represent the function as:

F(n) = 1 + 2 + 3 + 4 + …………..+ (n-2) + (n-1) + n

It can further be simplified as:



You can breakdown this function into two parts as follows:

Base Case: if(n==1) return 1;

Recursive step: F(n) = n + F(n-1)

**Advantages of recursion**

1. The code may be easier to write.

2. To solve such problems which are naturally recursive such as tower of Hanoi.

3. Extremely useful when applying the same solution.

4. Recursion reduce the length of code.

**Disadvantages of recursion**

1. Recursive functions are generally slower than non-recursive function.

2. It may require a lot of memory space to hold intermediate results on the system stacks.

3. Hard to analyze or understand the code.

4. It is not more efficient in terms of space and time complexity.

5. The computer may run out of memory if the recursive calls are not properly checked.

**1. Factorial of a given number:**

The factorial of a non-negative integer n, denoted by n!. It is the product of all positive integers less than or equal to n. For example, 5! =5\*4\*3\*2\*1=120

**Algorithm:**

Input: Integer ‘n’

Output: Factorial of ‘n’

Factorial(n)

{

  If n= 0 or 1 then

fact=1

else

fact= n\* factorial(n-1)

return(fact)

}

**Program:**

/\*This program returns factorial of a given number\*/

#include<stdio.h>

int factorial(int n){

if(n==0||n==1)

return 1;

else

return n\*factorial(n-1);

}

int main(){

int num;

printf("Enter the number:");

scanf("%d",&num);

int fact=factorial(num);

printf("Factorial:%d",fact);

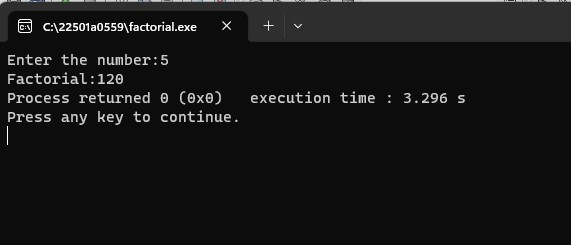
return 0;

}

**Output:**

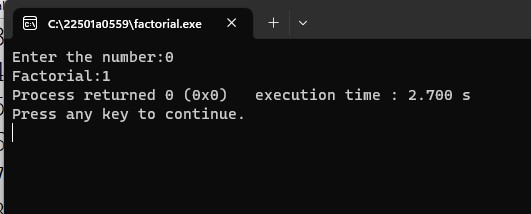
**Enter the number:5**

**Factorial:120**



**Enter the number:0**

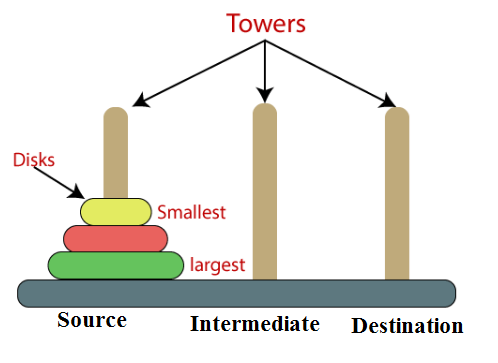
**Factorial:1**



**2. Towers of Hanoi Problem:**

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which should be transferred from source to destination using the intermediate rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

* Only one disk may be moved at a time
* Each move consists of taking the upper disk from one of the rods and sliding into another rod, on the top of the other disks that may be already present on the rod.
* No disk may be placed on top of a smaller disk.

****

**Algorithm:**

Input: Integer ‘n’ number of disks

Output: ‘n’ disks are to be transferred from peg S (Source) to peg D (destination) with Peg I as the intermediate peg

1. Start

2. Read N value as the no. of disks

3. Call TOH(N, S, I, D).

4. Stop

TOH (N, S, I, D)

{

  if n = 1 then

Transfer disk from S to D and stop

Else

// transfer N-1 disks from peg S to peg I with peg D as the intermediate peg

Call TOH(N-1, S, D, I)

Transfer disk from S to D

// transfer N-1 disks from peg I to peg D with peg S as the intermediate peg

Call TOH(N-1, I, S, D);

}

**Program:**

/\*Towers of hanoi\*/

#include <stdio.h>

int toh(int n,int src,int dest,int aux){

if(n>=1){

toh(n-1,src,aux,dest);

printf("move disk %d from rod %d to rod %d\n",n,src,dest);

toh(n-1,aux,dest,src);

}

}

int main() {

int N;

printf("Enter the number of disks:");

scanf("%d", &N);

toh(N, 1, 3, 2);

return 0;

}

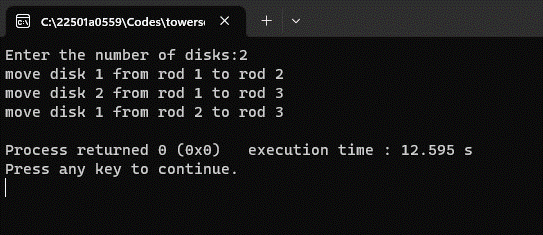
**Output:**

**Enter the number of disks:2**

**move disk 1 from rod 1 to rod 2**

**move disk 2 from rod 1 to rod 3**

**move disk 1 from rod 2 to rod 3**



**Enter the number of disks:4**

**move disk 1 from rod 1 to rod 2**

**move disk 2 from rod 1 to rod 3**

**move disk 1 from rod 2 to rod 3**

**move disk 3 from rod 1 to rod 2**

**move disk 1 from rod 3 to rod 1**

**move disk 2 from rod 3 to rod 2**

**move disk 1 from rod 1 to rod 2**

**move disk 4 from rod 1 to rod 3**

**move disk 1 from rod 2 to rod 3**

**move disk 2 from rod 2 to rod 1**

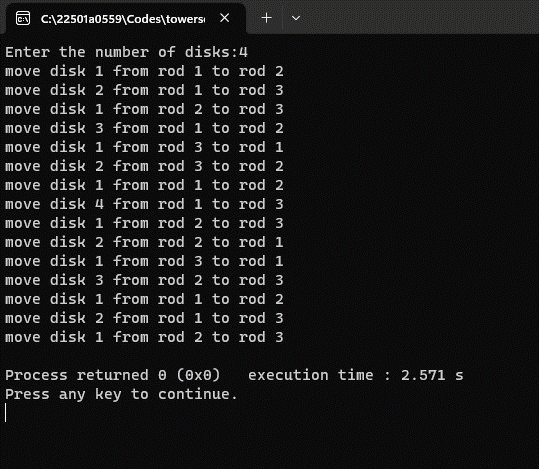
**move disk 1 from rod 3 to rod 1**

**move disk 3 from rod 2 to rod 3**

**move disk 1 from rod 1 to rod 2**

**move disk 2 from rod 1 to rod 3**

**move disk 1 from rod 2 to rod 3**



**3. Program to perform Binary Search using Recursion.**

**Program:**

/\*Binary search with recursion\*/

#include <stdio.h>

int binarySearch(int arr[], int low, int high, int x)

{

if (high >= low){

int mid = (low+high)/2;

if (arr[mid] == x) return mid;

if (arr[mid] > x) return binarySearch(arr, low, mid-1, x);

return binarySearch(arr, mid+1, high, x);

}

return -1;

}

int main(void)

{

int arr[200];

int n,x;

printf(“Enter the number of elements:\n”);

scanf(“%d”,&n);

printf(“Enter the elements:\n”);

for(int i=0;i<n;i++){

scanf(“%d”,&arr[i]);

}

printf(“Enter the value to be searched:\n”);

scanf(“%d”,&x);

int result = binarySearch(arr, 0, n-1, x);

if (result == -1) {

printf("Element is not present in array");

}else{

printf("Element is present at index %d", result);

}

return 0;

}

**Output:**

**Enter the number of elements:**

**5**

**Enter the elements:**

**4**

**3**

**2**

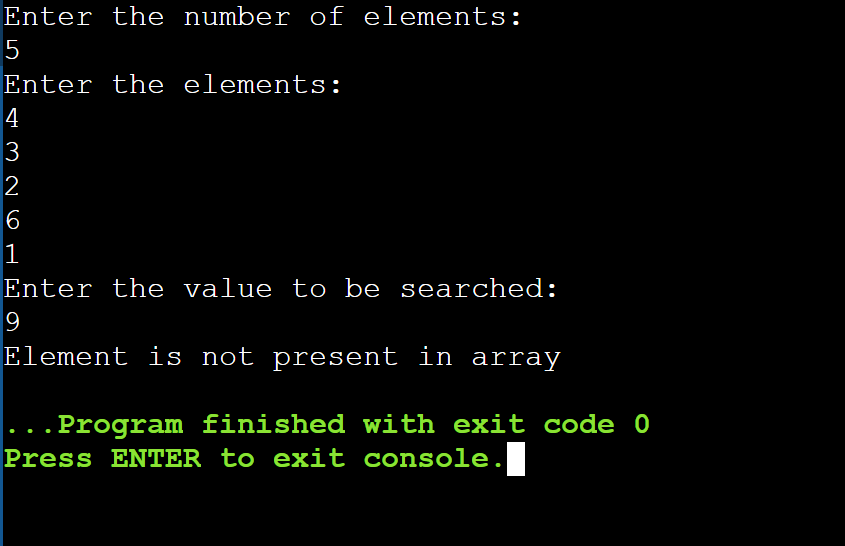
**6**

**1**

**Enter the value to be searched:**

**9**

**Element is not present in array**

****

**Enter the number of elements:**

**4**

**Enter the elements:**

**1**

**2**

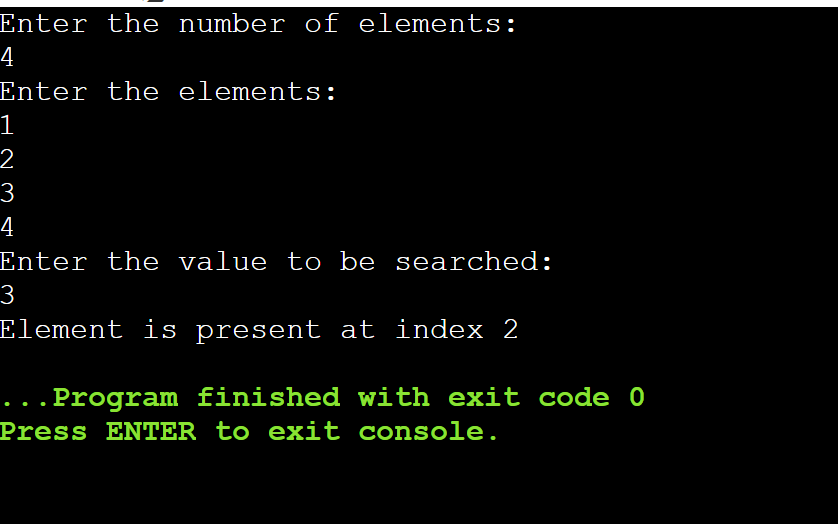
**3**

**4**

**Enter the value to be searched:**

**3**

**Element is present at index 2**



**4.Difference between recursion and iteration**

**Program:**

/\*factorial of given number\*/

#include <stdio.h>

// ----- Recursion -----

int factorialUsingRecursion(int n)

{

if (n == 0)

return 1;

return n \* factorialUsingRecursion(n - 1);

}

// ----- Iteration -----

int factorialUsingIteration(int n)

{

int fact = 1, i;

for (i = 2; i <= n; i++)

fact \*= i;

return fact;

}

// Driver Code

int main()

{

int num ;

printf(“Enter the number:\n”);

scanf(“%d”,&num);

printf("Factorial of %d using Recursion is: %d\n", num,

factorialUsingRecursion(num));

printf("Factorial of %d using Iteration is: %d", num,

factorialUsingIteration(num));

return 0;

}

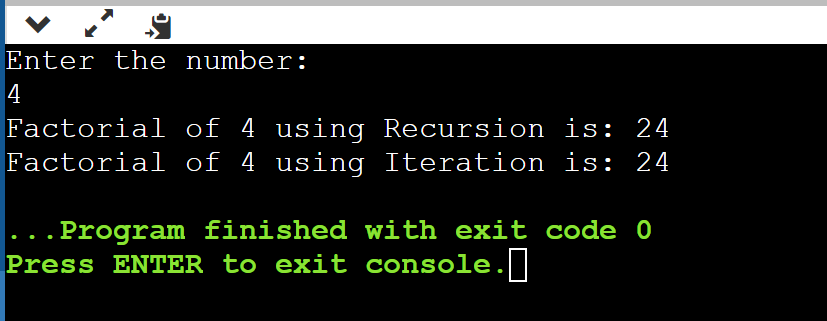
**Output:**

**Enter the number:**

**4**

**Factorial of 4 using Recursion is: 24**

**Factorial of 4 using Iteration is: 24**

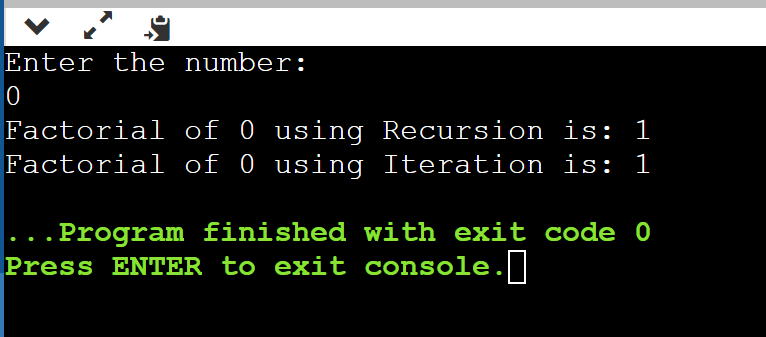
****

**Enter the number:**

**0**

**Factorial of 0 using Recursion is: 1**

**Factorial of 0 using Iteration is: 1**

****

**Viva questions:**

**What is recursion?**

Recursion is a programming technique where a function calls itself to solve a problem**.**

**How does recursion work?**

In recursion, a problem is divided into smaller subproblems, and each subproblem is solved using the same algorithm as the original problem.

**What are the two main components of a recursive function?**

The base case and the recursive case are the two main components of a recursive function. The base case defines the terminating condition, while the recursive case defines how the problem is broken down into smaller instances.

**What is a base case in recursion?**

The base case is the stopping condition in a recursive function. It defines a scenario in which the function does not call itself and returns a specific result.

**What is the purpose of a base case?**

The base case prevents the recursive function from calling itself infinitely and ensures that the recursion terminates successfully.

**What is a recursive case?**

The recursive case is where the function calls itself with a modified input to solve a smaller subproblem.

**What is the significance of a recursive case?**

The recursive case allows the problem to be divided into smaller subproblems that eventually lead to the base case. This is how recursion progresses towards solving the overall problem.

**What happens if a base case is not defined in a recursive function?**

Without a base case, the recursive function will continue to call itself indefinitely, leading to a stack overflow and program crash.

**What are the advantages of using recursion?**

Recursion can lead to elegant and concise solutions for certain problems, especially those that have a naturally recursive structure.

**What are the limitations of recursion?**

Recursion can be less efficient in terms of memory and performance compared to iterative solutions for some problems. It can also be harder to debug due to the complex call stack.

**Provide an example of a problem that is well-suited for recursion.**

Examples could include calculating factorials, computing Fibonacci numbers, or towers of hanoi.