2820278

# Panipat Institute Of Engineering & Technology

**Samalkha , Panipat**

Computer Science & Engineering-Emerging Technology Department

**Practical Lab :** Design and Analysis of Algorithms
PC-CS-AIML-317A

**Submitted To:**
Dr. Sumit Rana
Professor
CSE-ET Department

**Submitted By:**
Bhavika
B.Tech. CSE-ET(AIML)
Semester-5th
Roll No. - 2820278
Section-D2

**Affiliated to**

# PRACTICAL NO.-1

## Aim:-Write a program to implement linear search.

**Theory**:- Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm.The time complexity of this type of search is **O(n).**

## Algorithm:-

```
Algorithm Linear search (a, n, item, loc)
Here    a=is an array of n elements.
        n=no. of elements.
        Item=It denotes the value of the elements that is to be searched.
        Loc=It returns the location of the element if present.
1. Flag:=0;
2. Read a[1:n]
3. For(i=1 to n) step 1 do
4. { if (a[i]=item)
        Flag=1;
        Break; }
5. If (flag=1)
6. Print “Element found at loc ‘i’”.
7. Else print “Element not found”.
8. Exit.
```

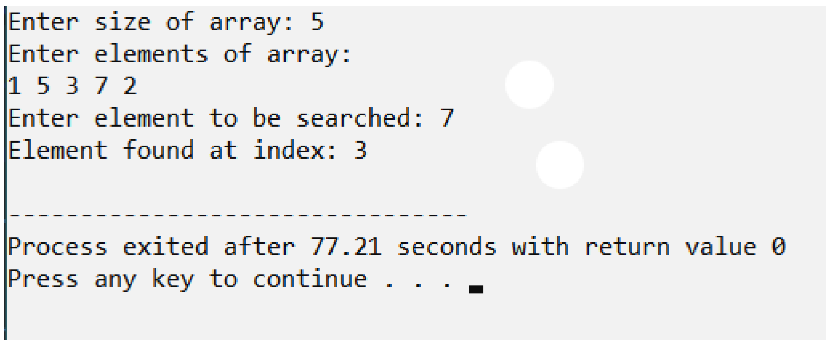
## Program:-

```
#include<bits/stdc++.h>

using namespace std;

int main(){

    int n;

    cout<<"Enter size of array: ";

    cin>>n;

    int a[n];

    cout<<"Enter elements of array: ";

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

        cin>>a[i];

    }

    int x;

    cout<<"Enter element to be searched: ";

    cin>>x;

    int i;
```

```
        for(i=0; i<n; i++){
            if(a[i] == x){
                cout<<"Element found at index: "<<i<<endl;
                break;
            }
        }
        if(i == n){
            cout<<"Element not found\n";
        }
        return 0;
    }
```

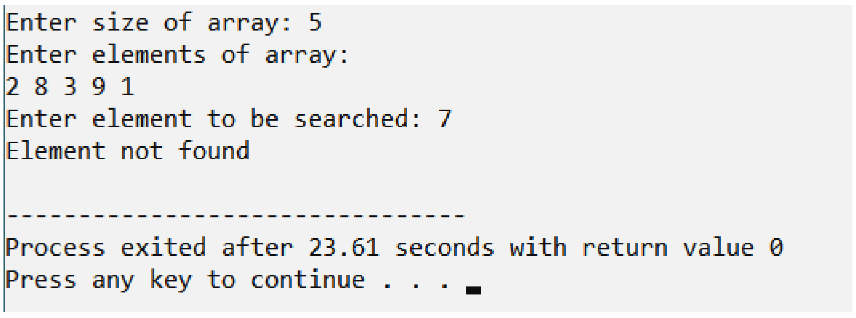
## Outputs:

### 1. Element found



```
Enter size of array: 5
Enter elements of array:
1 5 3 7 2
Enter element to be searched: 7
Element found at index: 3

--------------------------------
Process exited after 77.21 seconds with return value 0
Press any key to continue . . .
```

### 2. Element not found



```
Enter size of array: 5
Enter elements of array:
2 8 3 9 1
Enter element to be searched: 7
Element not found

--------------------------------
Process exited after 23.61 seconds with return value 0
Press any key to continue . . .
```

**Short answer Questions:**

**Q1. What is an algorithm?**
An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

**Q2. What is a linear search?**

**2820278**

A linear search refers to the way a target key is being searched in a sequential data structure. Using this method, each element in the list is checked and compared against the target key, and is repeated until found or if the end of the list has been reached.

**Q3. Why use sequential search?**

a. If the collection size is relatively small and you would be performing this search relatively a few times, then this might an option.
b. Another case is when the need is to have constant insertion performance (like using a linked list) and the search frequency is less.
c. In addition, linear search places very few restrictions on the complex data types. All that is needed is a match function of sorts.

**Q4. What is best case efficiency of linear search?**
The best case efficiency of linear search is O(1).

**Q5. Linear search is special case of which search.**
Linear search is special case of brute-force search.

**Q6. What are the various applications of linear search?**
Linear search is usually very simple to implement, and is practical when the list has only a few elements, or when performing a single search in an unordered list. When many values have to be searched in the same list, it often pays to pre-process the latter in order to use a faster method.

**Q7.What are ARRAY/s?**
When dealing with arrays, data is stored and retrieved using an index that actually refers to the element number in the data sequence. This means that data can be accessed in any order. In programming, an array is declared as a variable having a number of indexed elements.

**Q8.How do signed and unsigned numbers affect memory?**
In the case of signed numbers, the first bit is used to indicate whether positive or negative, which leaves you with one bit short. With unsigned numbers, you have all bits available for that number. The effect is best seen in the number range (unsigned 8 bit number has a range 0-255, while 8-bit signed number has a range -128 to +127.

**Q9.  How does variable declaration affect memory allocation?**
The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

**Q10. Differentiate NULL and VOID.**
Null is actually a value, whereas Void is a data type identifier. A variable that is given a Null value simply indicates an empty value. Void is used to identify pointers as having no initial size.

**2820278**

# PRACTICAL NO.-2

## Aim:- Write a program to implement binary search.

**Theory:-**In Binary Search is a searching algorithm for finding an element's position in a sorted array.In this approach, the element is always searched in the middle of a portion of an array.Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first. The time complexity of binary search is **O(log n).**

## Algorithm:-

Algorithm binary search (LB, UB, mid, a, item, n)

```
Here    a=is an array of n elements in which item is to be searched.
        LB=is the lower bound of the array that represents the index of first element.
        UB= is the upper bound of the array that represents the index of last  element.
        Mid= represents the middle value index of the array.
        Item= It denotes the value of the elements that is to be searched.
1.  LB:=0, UB:=n-1, flag:=0
2.  Read a[1:n] & item to be searched.
3.  mid = int [LB+UB]/2;
4.  While(LB<=UB&&a[mid]!=item)
5.  { if (a[mid]<item)
        LB=mid+1;
6.  Else UB=mid-1; }
7.  if(a[mid]==item)
8.  {
9.  flag=1;
10. printf("element is found at location:-%d",mid+1);
11.  else print "(element not found") }
12. Exit.
```

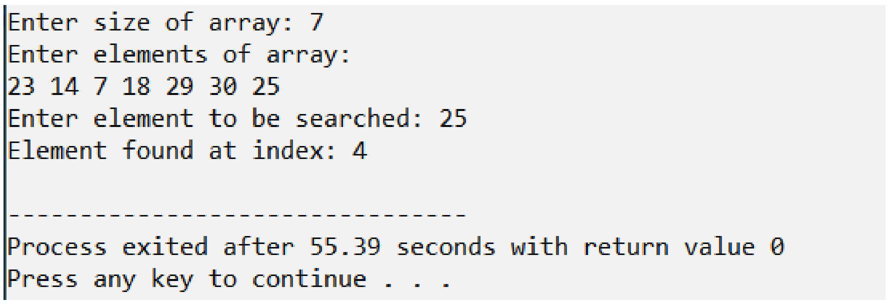
## Program:-

```
#include<bits/stdc++.h>
using namespace std;
int binarySearch(vector<int>& a, int n, int x){
    int l=0, r=n-1;
    while(l<=r){
        int mid = l + (r-l)/2;
        if(a[mid]==x){
            return mid;
        }
        else if(a[mid]<x){
            l = mid+1;
        }
        else{
            r = mid-1;
        }
    }
    return -1;
}
```

```
int main(){
    int n;
    cout<<"Enter size of array: ";
    cin>>n;
    vector<int> a(n);
    cout<<"Enter elements of array: ";
    for(int i=0; i<n; i++){
        cin>>a[i];
    }
    int x;
    cout<<"Enter element to be searched: ";
    cin>>x;
    sort(a.begin(), a.end());
    int result = binarySearch(a, n, x);
    if(result==-1){
        cout<<"Element not found\n";
    }
    else{
        cout<<"Element found at index: "<<result<<endl;
    }
}
```

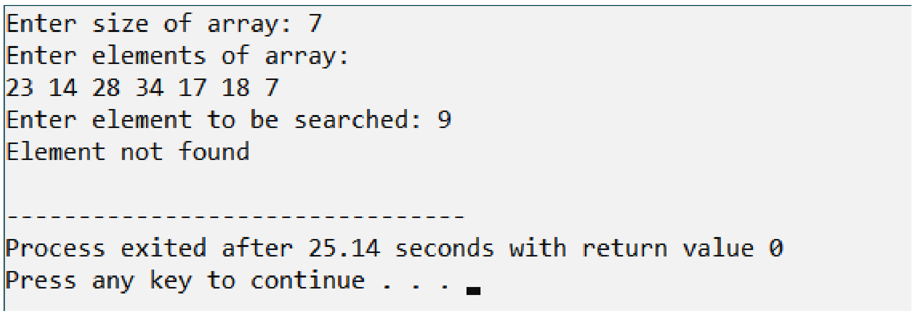
## Outputs:

### 1.Element found



```
Enter size of array: 7
Enter elements of array:
23 14 7 18 29 30 25
Enter element to be searched: 25
Element found at index: 4

--------------------------------
Process exited after 55.39 seconds with return value 0
Press any key to continue . . .
```

### 2.Element not found



```
Enter size of array: 7
Enter elements of array:
23 14 28 34 17 18 7
Enter element to be searched: 9
Element not found

--------------------------------
Process exited after 25.14 seconds with return value 0
Press any key to continue . . .
```

## Short Questions:

**Q1. How do you define binary search?**
A binary search or half-interval search finds the position of a specified value (the input “key”) within a sorted array. Binary search is applied only on the list which is sorted and if the list to be searched is not sorted, it needs to be sorted before binary search can be applied to it.

**2820278**

**Q2. Binary searches can be done on an array[Yes/No]?**
Yes.

**Q3. What is the best case efficiency of binary search?**
The best case efficiency of binary search is O(1).

**Q4. What is the worst case performance of binary search?**
The worst case performance of binary search is O(log n).

**Q5. What is time complexity of Binary Search?**
Time complexity of binary search is O(Logn).

**Q6. Can Binary Search be used for linked lists?**
Since random access is not allowed in linked list, it is not possible to reach the middle element in O(1) time. Therefore Binary Search is not possible for linked lists.

**2820278**

# PRACTICAL NO.-3

## Aim:-Write a program to implement the quick sort.

**Theory:-** QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick median as the pivot.

Mathematical analysis of quicksort shows that, on average, the algorithm takes **O(*n* log *n*)** comparisons to sort *n* items. In the worst case, it makes **O($n^2$)** comparisons, though this behavior is rare.

## Algorithm:-

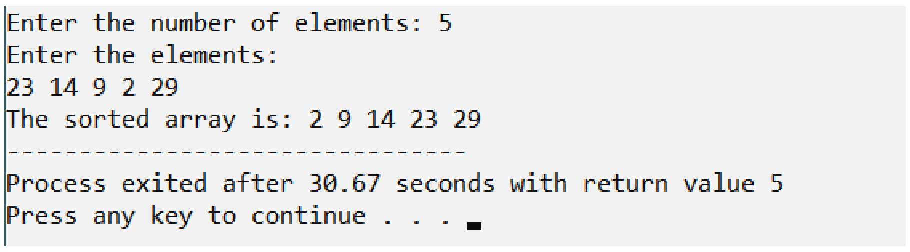
```
Algorithm Quick(A)

1.  If(p<q) then
2.  j=partition(a,p,q+1)
3.  Quicksort(p,j-1);
4.  Quicksort(j+1,q);

Algorithm partition(a,m,b)

1.  V=a[m], i=m, j=b
2.  Repeat {
3.  Repeat {
4.  i=i+1;
5.  until [a[i]>=v]; }
6.  repeat {
7.  j=j-1;
8.  until [a[j]<=v]; }
9.  if(i<j)
10. then interchange (a,i,j)
11. } until (i>=j);
12. a[m]:=a[j];
13. a[j]:=v;
14. return j;
15. }

Algorithm interchange(a ,i, j)

1.  {
2.  temp:=a[i];
3.  a[i]:=a[j];
4.  a[j]:=a[temp];
5.  }
```

**2820278**

## PROGRAM:-

```
#include<stdio.h>

int partition(int a[], int l, int h){
    int pivot, i, j, temp;
    pivot = a[h];
    i = l-1;
    for(j=l; j<h; j++){
        if(a[j] <= pivot){
            i = i+1;
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[i+1];
    a[i+1] = a[h];
    a[h] = temp;
    return i+1;
}

void quicksort(int a[], int l, int h){
    if(l<h){
        int q;
        q = partition(a, l, h);
        quicksort(a, l, q-1);
        quicksort(a, q+1, h);
    }
}

void main(){
    int i, n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter the elements: \n");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    quicksort(a, 0, n-1);
    printf("The sorted array is: ");
    for(i=0; i<n; i++)
        printf("%d ", a[i]);
}
```

## Output:



```
Enter the number of elements: 5
Enter the elements:
23 14 9 2 29
The sorted array is: 2 9 14 23 29
--------------------------------
Process exited after 30.67 seconds with return value 5
Press any key to continue . . .
```

**2820278**

## Short Questions:

**Q1. How Quick Sort works?**
Quick sort uses divide and conquer approach. It divides the list in smaller 'partitions' using 'pivot'. The values which are smaller than the pivot are arranged in the left partition and greater values are arranged in the right partition. Each partition is recursively sorted using quick sort.

**Q2. What is running time of Quick Sort?**
Quick Sort takes O(NLOG N) time on average and its worst case running time is O($N^2$).

**Q3. What is the worst case behaviour for  Quick Sort?**
Because of a bad pivot selection, the worst run is onto quadratic time complexity.

**Q4. In its worst case Quick Sort behaves as which sorting algorithm?**
Bubble Sort

**Q5. What is the basic idea behind Quick Sort?**
- Pick one element in the array, which will be the pivot.
- Make one pass through the array, called a partition step, re-arranging the entries so that:
- the pivot is in its proper place.
- entries smaller than the pivot are to the left of the pivot.
- entries larger than the pivot are to its right.

**Q6. Compare Quick Sort with Merge Sort?**
- Mergesort guarantees O(NlogN) time, however it requires additional memory with size N.
- Quicksort does not require additional memory; however the speed is not quaranteed.
- Usually mergesort is not used for main memory sorting, only for external memory sorting.

**Q7. What are the advantages of Quick Sort?**
- One of the fastest algorithms on average.
- Does not need additional memory (the sorting takes place in the array - this is called in-place processing). Compare with mergesort: mergesort needs additional memory for merging.

# PRACTICAL NO.-4

## Aim:-Write a program to implement Matrix Chain Multiplication.

**Theory:-** It is a Method under Dynamic Programming in which previous output is taken as input for next. Here, Chain means one matrix's column is equal to the second matrix's row [always].
In general:
If A = ⌊ aij⌋ is a p x q matrix
B = ⌊ bij⌋ is a q x r matrix
C = ⌊ cij⌋ is a p x r matrix
Then

$$AB = C \text{ if } c_{ij} = \sum_{k=1}^{q} a_{ik}\, b_{kj}$$

Given following matrices {A1,A2,A3,...An} and we have to perform the matrix multiplication, which can be accomplished by a series of matrix multiplications

A1 xA2 x,A3 x.....x An

Matrix Multiplication operation is associative in nature rather commutative. By this, we mean that we have to follow the above matrix order for multiplication but we are free to parenthesize the above multiplication depending upon our need.

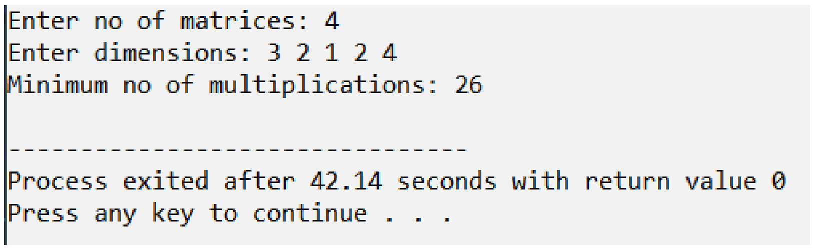
## Algorithm:-

```
1. n <--- length [p] - 1
2. for i <--- 1 to n
3.      do m [i , i] <--- 0
4. for l <--- 2 to n
5.     do for i <--- 1 to n-l + 1
6.        do j<--- l + l -1
7.        m [i , j] <--- ∞
8.       for k <-- I to j-1
9.             do q <-- m [i, k] + m[k+1 , j ] + Pi - 1Pk Pj
10.                  if q < m [i, j]
11.                    then m [i,j] <--- q
12.                            s [i,j]  <--- k
13. Return m and s
```

## PROGRAM:-

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a;
    cout<<"Enter no of matrices: ";
    cin>>a;
    int p[a+1];
    cout<<"Enter dimensions: ";
    for(int i=0; i<a+1; i++){
        cin>>p[i];
    }
    int m[a][a];
    int s[a][a];
    for(int i=0; i<a; i++){
        m[i][i] = 0;
    }
    for(int l=2; l<=a; l++){
        for(int i=0; i<a-l+1; i++){
            int j = i+l-1;
            m[i][j] = INT_MAX;
            for(int k=i; k<j; k++){
                int q = m[i][k] + m[k+1][j] + p[i]*p[k+1]*p[j+1];
                if(q<m[i][j]){
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
    cout<<"Minimum no of multiplications: "<<m[0][a-1]<<endl;
    return 0;
}
```

## Output:-



```
Enter no of matrices: 4
Enter dimensions: 3 2 1 2 4
Minimum no of multiplications: 26

--------------------------------
Process exited after 42.14 seconds with return value 0
Press any key to continue . . .
```

**2820278**

# Practical no. 4

## Aim:- Write a program to find Longest Common Subsequence.

**Theory:-**Here longest means that the subsequence should be the biggest one. The common means that some of the characters are common between the two strings. The subsequence means that some of the characters are taken from the string that is written in increasing order to form a subsequence.
Let's understand the subsequence through an example.
Suppose we have a string 'w'.
W1 = abcd
The following are the subsequences that can be created from the above string:

- ab
- bd
- ac
- ad
- acd
- bcd

The above are the subsequences as all the characters in a sub-string are written in increasing order with respect to their position. If we write ca or da then it would be a wrong subsequence as characters are not appearing in the increasing order. The total number of subsequences that would be possible is 2n, where n is the number of characters in a string. In the above string, the value of 'n' is 4 so the total number of subsequences would be 16.
$W_2$= bcd

## Algorithm:-

- **LCS_Length-Table-Formulation (X, Y)**

```
m := length(X)
n := length(Y)
for i = 1 to m do
C[i, 0] := 0
for j = 1 to n do
C[0, j] := 0
for i = 1 to m do
for j = 1 to n do
if xi = yj
C[i, j] := C[i - 1, j - 1] + 1
B[i, j] := 'D'
else
if C[i -1, j] ≥ C[i, j -1]
C[i, j] := C[i - 1, j] + 1
B[i, j] := 'U'
else
C[i, j] := C[i, j - 1]
B[i, j] := 'L'
return C and B
```

- **Print_LCS (B, X, i, j)**

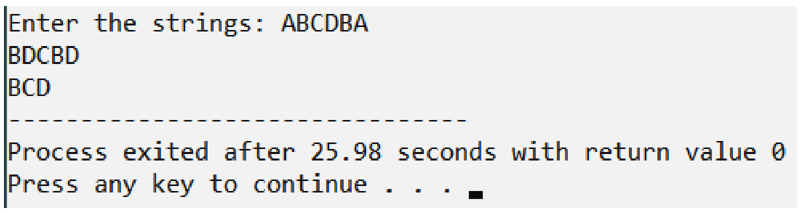
```
if i = 0 and j = 0
return
if B[i, j] = 'D'
Print-LCS(B, X, i-1, j-1)
Print(xi)
else if B[i, j] = 'U'
```

```
Print-LCS(B, X, i-1, j)
else
Print-LCS(B, X, i, j-1)
```

## PROGRAM:-

```
#include<bits/stdc++.h>

using namespace std;

int c[10][10];
int b[10][10];

void print_lcs(int i, int j, string s1){
    if(i==0 || j==0){
        return;
    }
    if(b[i][j] == 1){
        print_lcs(i-1, j-1, s1);
        cout<<s1[i-1];
    }
    else if(b[i][j] == 2){
        print_lcs(i-1, j, s1);
    }
    else{
        print_lcs(i, j-1, s1);
    }
}

int main(){
    string s1, s2;
    cout<<"Enter the strings: ";
    cin>>s1>>s2;
    int m = s1.length();
    int n = s2.length();
    for(int i=1; i<=m; i++){
        c[i][0] = 0;
    }
    for(int j=0; j<=n; j++){
        c[0][j] = 0;
    }
    for(int i=1; i<=m; i++){
        for(int j=1; j<=n; j++){
            if(s1[i-1] == s2[j-1]){
                c[i][j] = c[i-1][j-1] + 1;
                b[i][j] = 1;
            }
            else{
                if(c[i-1][j] >= c[i][j-1]){
                    c[i][j] = c[i-1][j];
                    b[i][j] = 2;
                }
                else{
                    c[i][j] = c[i][j-1];
                    b[i][j] = 3;
                }
            }
        }
    }
    print_lcs(m, n, s1);
    return 0;
}
```

## Output:-



```
Enter the strings: ABCDBA
BDCBD
BCD
--------------------------------
Process exited after 25.98 seconds with return value 0
Press any key to continue . . .
```