WEEK-1
SEARCHING TECHNIQUES

1.1 OBJECTIVE:

a. Write a Python script to for implementing linear search technique.
b. Write a Python script to for implementing binary search technique.
c. Write a Python script to for implementing Fibonacci search technique.

1.2 RESOURCES:

Python 3.4.0

1.3 PROGRAM LOGIC:

Linear Search Algorithm

```
Algorithm linsrch (a[], x)
 { // a[1:n] is an array of n elements
      index := 0; flag := 0;
      while (index < n) do
      {
           if (x = a[index]) then
               {    flag := 1; break;
               }
              index ++;
      }
    if(flag =1)
          write("Data found ");
   else
          write("data not found");
}
```

Example: Given a list of n elements and search a given element x in the list using linear search.

a. Start from the leftmost element of list a[] and one by one compare x with each element of list a[].
b. If x matches with an element, return the index.
c. If x doesn't match with any of elements, return -1.

Consider a list with 10 elements and search for 9.

a = [56, 3, 249, 518, 7, 26, 94, 651, 23, 9]

Index →	0	1	2	3	4	5	6	7	8	9
Iteration 1	56	3	249	518	7	26	94	651	23	9
Iteration 2	56	3	249	518	7	26	94	651	23	9
Iteration 3	56	3	249	518	7	26	94	651	23	9
Iteration 4	56	3	249	518	7	26	94	651	23	9
Iteration 5	56	3	249	518	7	26	94	651	23	9
Iteration 6	56	3	249	518	7	26	94	651	23	9
Iteration 7	56	3	249	518	7	26	94	651	23	9
Iteration 8	56	3	249	518	7	26	94	651	23	9
Iteration 9	56	3	249	518	7	26	94	651	23	9
Iteration 10	56	3	249	518	7	26	94	651	23	9

Binary Search Algorithm

```
Algorithm binsrch (a[], n, x)
 { // a[1:n] is an array of n elements
       low = 1;
       high = n;
       while (low < high) do
       {
            mid = (low + high)/2 ;
           if (x < a[mid]) then
                  high = mid – 1;
          else if (x > a[mid]) then
                  low = mid + 1;
          else
               return mid;
        }
      return 0;
}
```

Example: Given a sorted list of a[] of n elements, search a given element x in list.

a. Search a sorted list by repeatedly dividing the search interval in half. Begin with an interval covering the whole list.
b. If the search key is less than the item in the middle item, then narrow the interval to the lower half. Otherwise narrow it to the upper half.
c. Repeat the procedure until the value is found or the interval is empty.

Consider a sorted list a[] with 9 elements and the search key is 31.

0	1	2	3	4	5	6	7	8
11	23	31	33	65	68	71	89	100

Let the search key = 31.
First low = 0, high = 8, mid = (low + high) = 4
a[mid] = 65 is the centre element, but 65 > 31.
So now high = mid - 1= 4 - 1 = 3, low = 0, mid = (0 + 3) / 2 = 1

a[mid] = a[1] = 23, but 23 < 31.
Again low = mid +1 = 1 +1 =2, high = 3, mid = (2 + 3) /2 = 2
a[mid] = a[2] = 31 which is the search key, so the search is successful.

Fibonacci Search Algorithm

```
Algorithm  fib_Search (arr, x, n)
 { // arr[1:n] is an array of n elements
        M2 := 0 ;
        M1 := 1 ;
        M := M2 +M1 ;
        while (fibM < n) do
         {
                M2 := M1;
               M1: =M;
                M := M2 + M1;
         }
        Offset: = -1;
        while (fibM > 1) do
         {
                i := min(offset+M2, n-1);
               if (arr[i] < x) then
               {
                        M := M1;
                        M1 := M2;
                        M2: = M -M1;
                        offset = i
               }
                else if (arr[i] > x) then
                {
                        M:= M2;
                        M1: = M1 -M2;
                        M2 := M - M1;
               }
               else
                         return i;
        }
        if(M1 and arr[offset+1] = x) then
               return offset+1;

        return -1;
}
```

Example: Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1.

First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
Let a[0..n-1] be the input list and element to be searched be x.

1. Find the smallest Fibonacci Number greater than or equal n. Let this number be M (m'th Fibonacci Number). Let the two Fibonacci numbers preceding it be M1 [(m-1)'th Fibonacci Number] and M2 [(m-2)'th Fibonacci Number].
1. While the array has elements to be inspected: Compare x with the last element of the range covered by M2
2. If x matches, return index.
3. Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
4. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate elimination of approximately front one-third of the remaining array.
2. Since there might be a single element remaining for comparison, check if M1 is 1. If Yes, compare x with that remaining element. If match, return index.

Consider a list a[] with 11 elements and the search element is 85.
n = 11

Index	0	1	2	3	4	5	6	7	8	9	10
a[i]	10	22	35	40	45	50	80	82	85	90	100

Smallest Fibonacci number greater than or equal to 11 is 13.
M2 = 5, M1 = 8, M = M1 + M2 = 13
Initialize offset = 0
Check the element at index i = min(offset + M2, n)

M2	**M1**	**M**	**Offset**	**I = min(offset + M2, n)**	**A[i]**	**Consequence**
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

1.4 PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

1.5 SOURCE CODE:

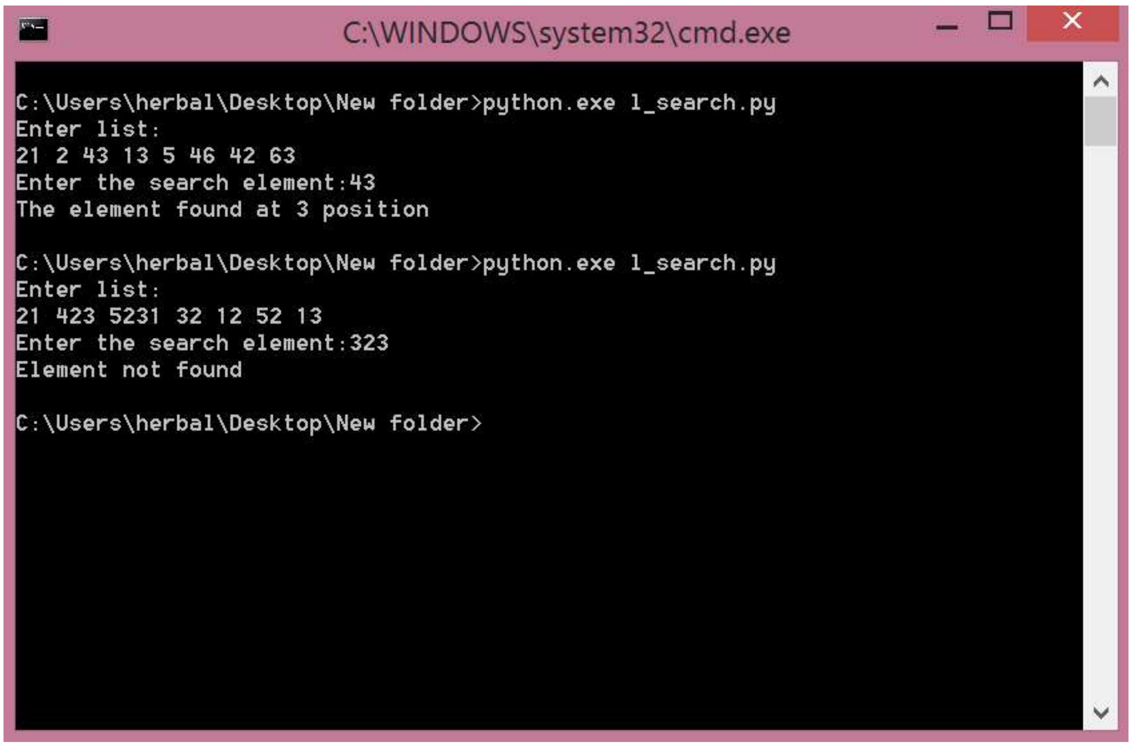
Implementation of Linear Search

```
def l_search(a,x,l,n):
    if l<n:
        if a[l]==x:
            print("The element found at",l+1,"position")
        else:
```

```
            l_search(a,x,l+1,n)
    else:
        print("Element not found")

print("Enter list:")
a=[int(b) for b in input().split()]
x=eval(input("Enter the search element:"))
n=len(a)
l_search(a,x,0,n)
```

Output:



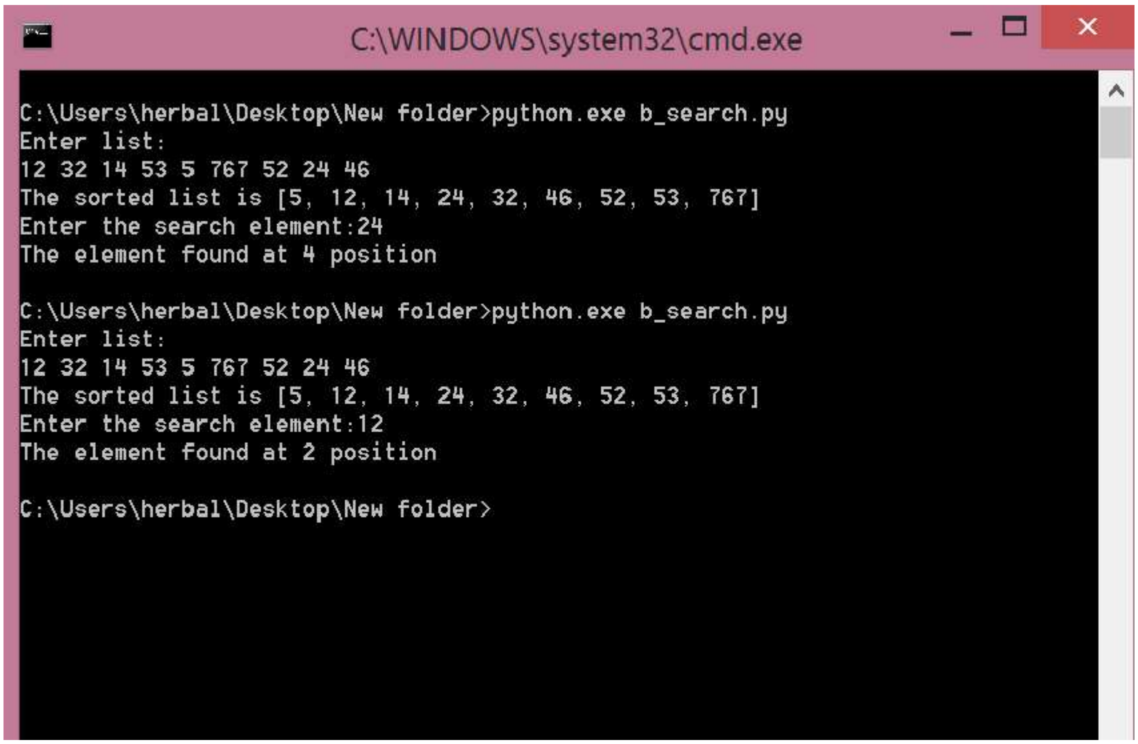
Implementation of Binary Search

```
def b_search(a,x,l,n):
    if l<=n:
        mid=(l+n)//2
        if a[mid]==x:
            print("The element found at",mid+1,"position")
        else:
            if a[mid]>x:
                b_search(a,x,l,mid-1)
            else:
                b_search(a,x,mid+1,n)
    else:
        print("Element not found")

print("Enter list:")
```

```
a=[int(b) for b in input().split()]
list.sort(a)
print("the sorted list is",a)
x=eval(input("Enter the search element:"))
n=len(a)
b_search(a,x,0,n)
```

Output:



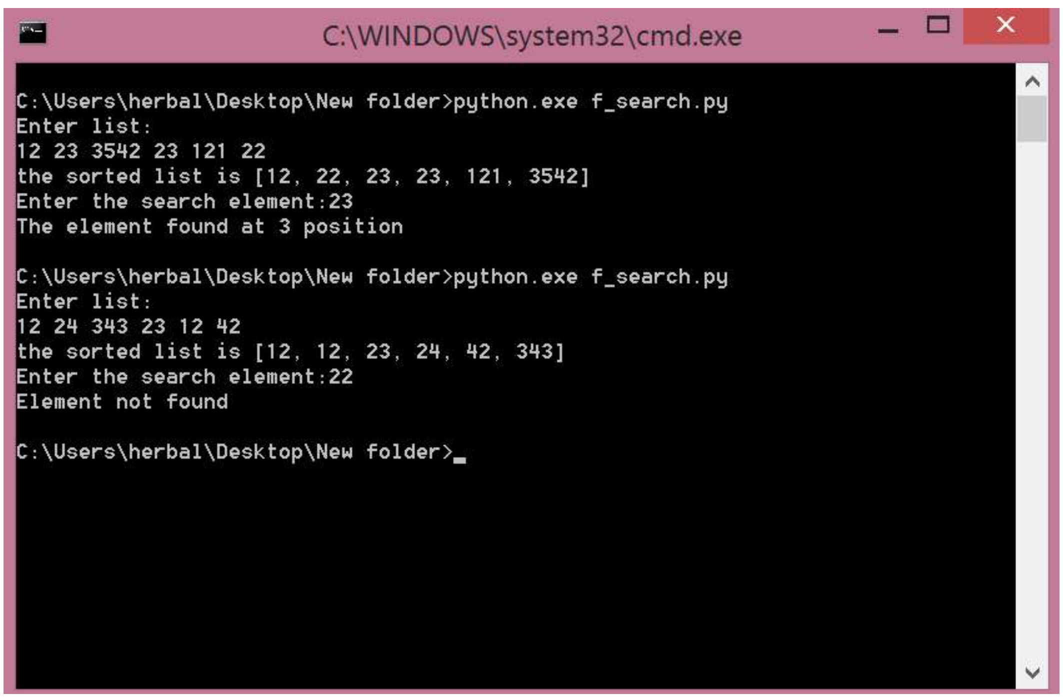
Implementation of Fibonacci Search

```
def f_search(a,x,n):
    f0=0
    f1=1
    f2=f0+f1
    while f2<n:
        f0=f1
        f1=f2
        f2=f0+f1
    offset=-1
    while f2>1:
        i = min(offset+f2, n-1)
        if (a[i]<x):
            f2=f1
            f1=f0
            f0=f2-f1
            offset = i
        elif (a[i]>x):
            f2=f0
```

```
        f1=f1-f2
        f0=f2-f1
      else :
        return i
    if(f1 and a[offset+1]==x):
      return offset+1
    return -1

print("Enter list:")
a=[int(b) for b in input().split()]
list.sort(a)
print("the sorted list is",a)
x=eval(input("Enter the search element:"))
n=len(a)
pos=f_search(a,x,n)
if pos>=0:
    print("The element found at",pos+1,"position")
else:
    print("Element not found")
```

Output:



1.6 PRE LAB VIVA QUESTIONS:

1. Define searching?
2. Define a list?
3. List out different types of searching techniques?
4. Differentiate between list and dictionary?
5.

1.7 LAB ASSIGNMENT:

1. A person has registered for voter id , he received a voter number and he need to check whether it exist in the voter or not. Use a binary searching in a recursive way to find whether the voter number exist in the list or not.
2. Use linear search technique to search for a key value in a given list of characters and print the message found or not.

1.8 POST LAB VIVA QUESTIONS:

1. Find the time complexity of linear search?
2. Find the time complexity of binary search?
3. Find the time complexity of Fibonacci search?