MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

Name: Vivek Kumar Ahirwar **Scholar No:** 191112419

Department: CSE **Section:**3

Semester: 4th **Subject:** ADA Lab

Date: 27/01/2021

Subject	**Analysis and Design of Algorithms LAB**	**Session: Jan 2021**
Sub. Code:	**CSE-228**	**Semester: IV (CSE)**
Name of Teacher	**Prof. Manish Pandey**	

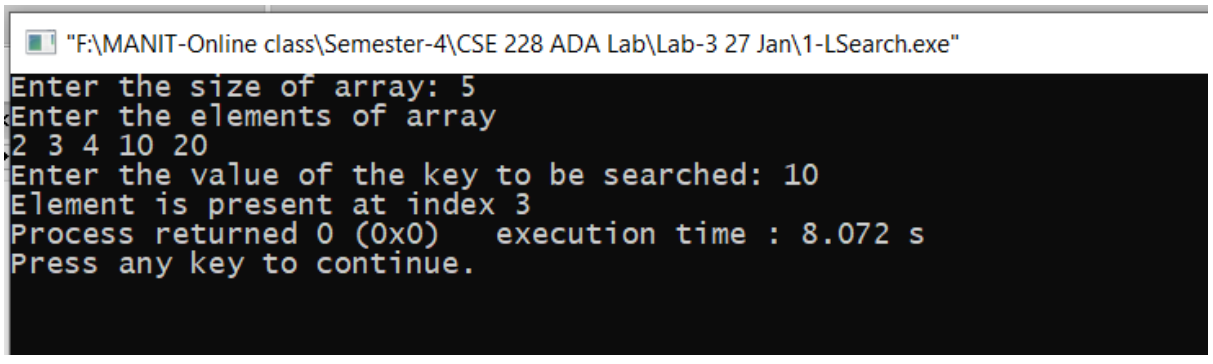
ADA: LAB-ASSIGNMENT 3

Prog: Write programs for implementing the following searching techniques and analyze the time complexity:
a. Linear search
b. Binary search
c. Fibonacci search

Linear Search

```cpp
// Keep Changing....@Vi

#include <iostream>
using namespace std;

// Linear search function
int linearSearch(int arr[], int n, int x)
{
    int i = 0;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

int main(void)
{
    int n;
    cout << "Enter the size of array: ";
    cin >> n;
    int arr[n];
```

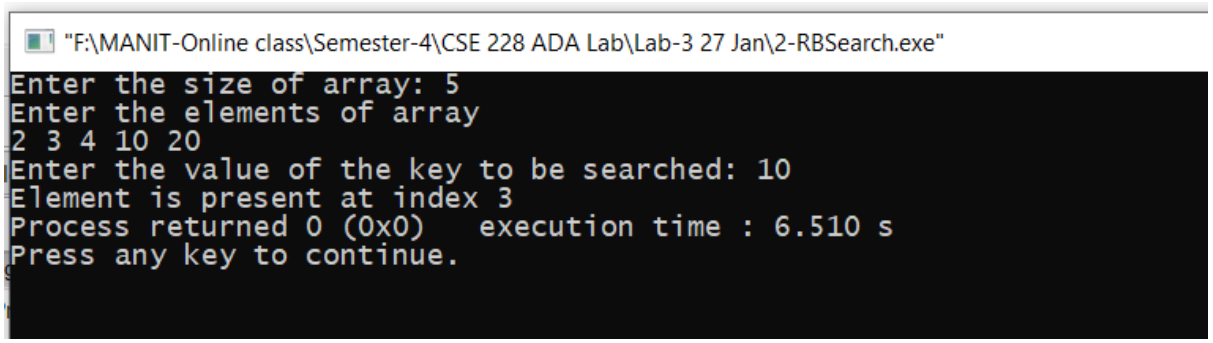
```
    cout << "Enter the elements of array\n";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    int key;
    cout << "Enter the value of the key to be searched: ";
    cin >> key;

    int result = linearSearch(arr, n, key);

    if (result == -1)
    {
        cout << "Element is not present in array";
    }

    else
    {
        cout << "Element is present at index " << result;
    }

    return 0;
}
```



Binary Search

```cpp
// Keep Changing....@Vi

#include <bits/stdc++.h>
using namespace std;

// Recursive binary search function
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;

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

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

        return binarySearch(arr, mid + 1, r, x);
    }

    return -1;
}

int main(void)
{
    int n;
    cout << "Enter the size of array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of array\n";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    int key;
    cout << "Enter the value of the key to be searched: ";
    cin >> key;

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

    if (result == -1)
    {
        cout << "Element is not present in array";
    }
```

```
    else
    {
        cout << "Element is present at index " << result;
    }

    return 0;
}
```



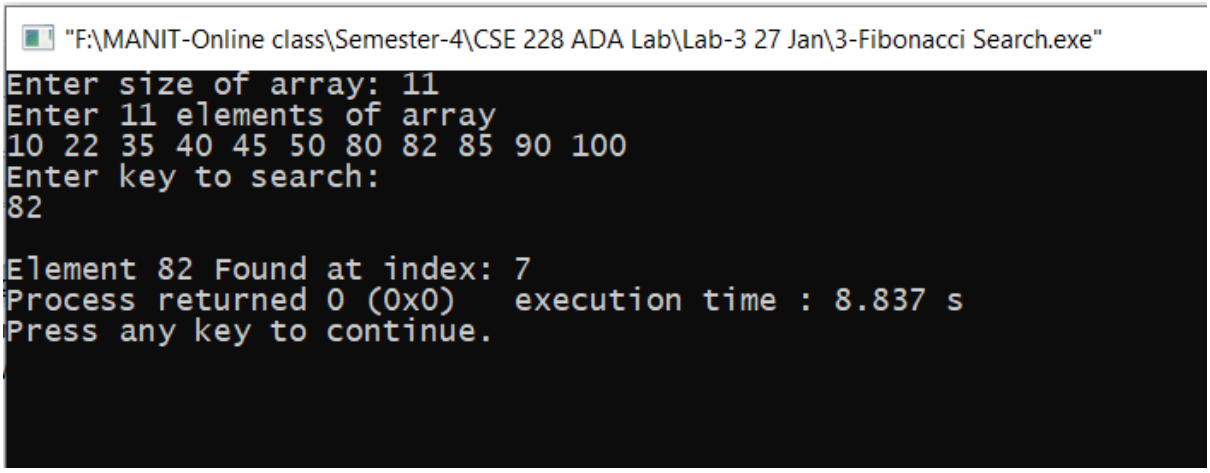
Time complexity of Linear search is O(n)

Time complexity of Recursive binary search is O(logn).

Fibonacci Search

```cpp
// Fibonacci Search

#include <iostream>
using namespace std;

//  Returns index of x if present, else returns -1
int fibMonaccianSearch(int arr[], int x, int n)
{
    /* Initialize fibonacci numbers */
    int fibMMm2 = 0;              // (m-2)'th Fibonacci No.
    int fibMMm1 = 1;              // (m-1)'th Fibonacci No.
    int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

    /* fibM is going to store the smallest Fibonacci
    Number greater than or equal to n */
    while (fibM < n)
    {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marks the eliminated range from front
    int offset = -1;

    /* while there are elements to be inspected. Note that
    we compare arr[fibMm2] with x. When fibM becomes 1,
    fibMm2 becomes 0 */
    while (fibM > 1)
    {
        // Check if fibMm2 is a valid location
        int i = min(offset + fibMMm2, n - 1);

        /* If x is greater than the value at index fibMm2,
        cut the subarray array from offset to i */
        if (arr[i] < x)
        {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        /* If x is greater than the value at index fibMm2,
        cut the subarray after i+1 */
        else if (arr[i] > x)
```

```
        {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* element found. return index */
        else
            return i;
    }

    /* comparing the last element with x */
    if (fibMMm1 && arr[offset + 1] == x)
        return offset + 1;

    /*element not found. return -1 */
    return -1;
}

int main()
{
    int n = 0;
    cout << "Enter size of array: ";
    cin >> n;

    int arr[n];

    cout << "Enter " << n << " elements of array" << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    // 10 22 35 40 45 50 80 82 85 90 100

    int x;
    cout << "Enter key to search: " << endl;
    cin >> x;
    //  x = 85;

    cout << endl;
    cout << "Element " << x << " Found at index: " << fibMonaccianSearch(arr,
x, n);

    return 0;
}
```



Analysis of Time complexity :

Worst case time complexity: Θ(logn) Average case time complexity: Θ(log n) Best case time complexity: Θ(1) Space complexity: Θ(1) With each step, the search space is reduced by 1/3 on average, hence, the time complexity is O(log N) where the base of the logarithm is 3.