Experiment #11

Searching algorithms

Student's Name:	
Semester:	Date:

Assessment:

Assessment Point	**Weight**	**Grade**
Methodology and correctness of results		
Discussion of results		
Participation		
Assessment Points' Grade:		

Comments:

Experiment #11:

Searching and Fibonacci Algorithms

Objectives:

1. To introduce the students with the searching algorithms
2. To implement the linear search algorithm
3. To implement the linear search algorithm
4. To store duplicated and non-duplicated random numbers in arrays
5. To implement the Fibonacci Algorithm using iteration and recursion.

Discussion:

Linear search and binary search are the two methods which are used in arrays for **searching** the elements. Searching is a process of finding an element within the list of elements stored in any order or randomly.

In a **linear search**, each element of an array is retrieved one by one in a logical order and checked whether it is desired element or not. A search will be unsuccessful if all the elements are accessed, and the desired element is not found. The number of comparisons made in searching a record in a search table determines the efficiency of the technique. If the desired item is present in the first position of the search table, then only one comparison is made. When the desired item is the last one, then n comparisons have to be made. So, the worst-case efficiency of this technique is O(n) stands for the order of execution.

Binary search is an extremely efficient algorithm. This search technique consumes less time in searching the given item in minimum possible comparisons. To do the binary search, first, we have to sort the array elements.

There are three cases could arise :

1. If the element is the required element, then the search is successful.
2. When the element is less than the desired item, then search only the first half of the array.
3. If it is greater than the desired element, then search in the second half of the array.

Repeat the same steps until an element is found or exhausts in the search area. In this algorithm, every time search area is reducing. Therefore, the number of comparisons is at most log (N+1). As a result, it is an efficient algorithm when compared to linear search, but the array has to be **sorted** before doing the binary search.

```
/* C++ Program - Linear Search */

#include<iostream >
using namespace std;
#include<conio.h>
void main()
{
        int arr[10], i, num, n, c=0, pos;
        cout<<"Enter the array size : ";
        cin>>n;
        cout<<"Enter Array Elements : ";
        for(i=0; i<n; i++)
        {
                cin>>arr[i];
        }
        cout<<"Enter the number to be search : ";
        cin>>num;
        for(i=0; i<n; i++)
        {
                if(arr[i]==num)
                {
                        c=1;
                        pos=i+1;
                        break;
                }
        }
        if(c==0)
        {
                cout<<"Number not found..!!";
        }
        else
        {
                cout<<num<<" found at position "<<pos;
        }
        getch();
}
```

```
/* C++ Program - Binary Search */

#include<iostream >
using namespace std;
#include<conio.h>
void main()
{

        int n, i, arr[50], search, first, last, middle;
```

```
            cout<<"Enter total number of elements :";
            cin>>n;
            cout<<"Enter "<<n<<" number :";
            for (i=0; i<n; i++)
            {
                        cin>>arr[i];
            }
            cout<<"Enter a number to find :";
            cin>>search;
            first = 0;
            last = n-1;
            middle = (first+last)/2;
            while (first <= last)
            {
                        if(arr[middle] < search)
                        {
                                    first = middle + 1;

                        }
                        else if(arr[middle] == search)
                        {
                                    cout<<search<<" found at location "<<middle+1<<"\n";
                                    break;
                        }
                        else
                        {
                                     last = middle - 1;
                        }
                        middle = (first + last)/2;
            }
            if(first > last)
            {
                        cout<<"Not found! "<<search<<" is not present in the list.";
            }
            getch();
}
```

Example 1:

Write a c++ program to generate random numbers to be stored in an array.

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
```

```
int random[26];
srand((unsigned)time(NULL));
for (int i = 1; i < 26; i++)
{
random[i] = 1+ rand() % 10;
cout << random[i] << endl;
}
return 0;
}
```

Example 2:

Write a c++ program to generate non-duplicate random numbers to be stored in an array of size 50.

```
#include <iostream>
#include <ctime>
using namespace std;
void shuffle(int *arr, size_t n)
{
    if (n > 1)
    {
        size_t i;
        srand(time(NULL));
        for (i = 0; i < n - 1; i++)
        {
          size_t j = i + rand() / (RAND_MAX / (n - i) + 1);
          int t = arr[j];
          arr[j] = arr[i];
          arr[i] = t;
        }
    }
}

void main()
```

```
{
    int i;
      const int s=50;
    int arr[s];
    for (i=0; i<s; i++){
        arr[i] = i;
    }
    shuffle(arr, s);
    for (i=0; i<s; i++){
        cout<< arr[i]<<"  ";
    }
}
```

Exercise 1:

Modify the binary search program such that the array will hold non-duplicate random numbers.

Example 2:

Write a c++ program to generate Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……) using iteration.

```
#include<iostream>
using namespace std;
void main()
{
  int n, c, first = 0, second = 1, next;
  cout << "Enter the number of terms of Fibonacci series you want" << endl;
  cin >> n;
  cout << "First " << n << " terms of Fibonacci series are :- " << endl;
  for ( c = 0 ; c < n ; c++ )
  {
    if ( c <= 1 )
      next = c;
    else
    {
      next = first + second;
      first = second;
      second = next;
    }
    cout << next << endl;
  }}
```

Exercise 2:

Write a c++ program to generate Fibonacci numbers using recursion.

Output