MAULANA AZAD NA TIONAL INSTITUTE OF TECHNOLOGY

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

**Department: Computer Science and Engineering**

**Section:3**

**Semester:4th**

**Subject: ADA Lab**

**Date:06/01/2020**

|  |  |  |
| --- | --- | --- |
| **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 1

**Prog.1:** Implement Recursive Binary search and Linear search and determine the time taken to search an element

**Linear Search**

*// 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

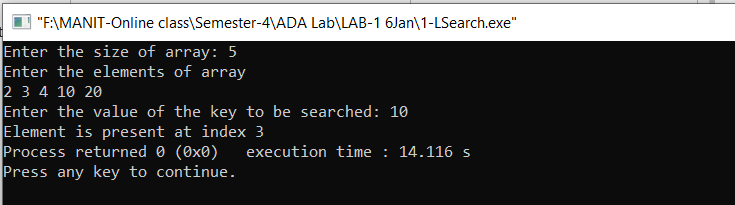
    {

        cout << "Element is present at index " << result;

    }

    return 0;

}

****

**Time Taken is 14.116sec.**

**Recursive Binary Search**

*// 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

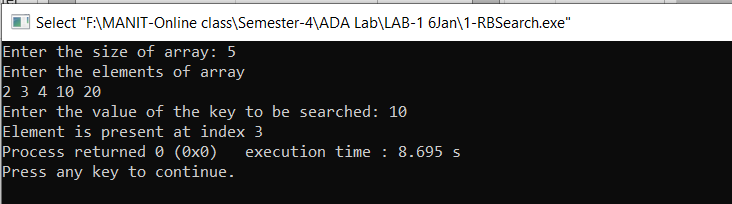
    {

        cout << "Element is present at index " << result;

    }

    return 0;

}

****

**Time Taken is 8.695sec.**

Time complexity of Linear search is O(n)

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

**Prog.2:** Write a program to determine if a given matrix is a sparse matrix? Calculate its time and Space complexity. How it is more efficient than the conventional matrix?

*// Keep Changing....@Vi*

*/\**

*\* Write a program to determine if a given  matrix is a sparse matrix?*

*\* Calculate its time and Space complexity. How it is more efficient than the conventional matrix?*

*\* Sparse martix has more zero elements than nonzero elements.*

*\*/*

#include <iostream>

using **namespace** std;

**int** main()

{

**int** i, j, m, n;

**int** count = 0;

    cout << "Enter the order of the matix \n";

    cin >> m >> n;

**int** a[m][n];

    cout << "Enter the element of the matix \n";

    for (i = 0; i < m; ++i)

    {

        for (j = 0; j < n; ++j)

        {

            cin >> a[i][j];

            if (a[i][j] == 0)

            {

                ++count;

            }

        }

    }

    if (count > ((m \* n) / 2))

    {

        printf("The given matrix is sparse matrix \n");

    }

    else

    {

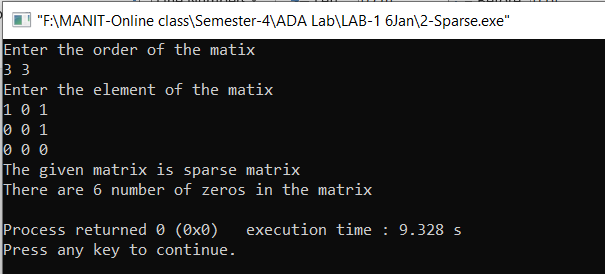
        printf("The given matrix is not a sparse matrix \n");

    }

    cout << "There are " << count << " number of zeros in the matrix" << endl;

    return 0;

}



**Time Complexity: The time complexity of this program is O(MN) if order of matrix is M\*N.**

**Sparse matrix is better then normal matrix because of following reasons**

* **Storage:**There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
* **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements.

**Prog.3:** What is Bubble Sort. Write algorithm of mention the Time & Space complexity of the Algorithm. Also suggest improvements which will improve the best case running time of Algorithm to O(n).

*// Keep Changing....@Vi*

#include <bits/stdc++.h>

using **namespace** std;

*//Bubble Sort*

**void** Bubble(**int** A[], **int** n)

{

**int** i, j;

    for (i = 0; i < n - 1; i++)

    {

        for (j = 0; j < n - i - 1; j++)

        {

            if (A[j] > A[j + 1])

            {

                swap(A[j], A[j + 1]);

            }

        }

    }

}

*//Optimised Bubble Sort*

**void** BubbleSort(**int** A[], **int** n)

{

**int** i, j, flag = 0;

    for (i = 0; i < n - 1; i++)

    {

        flag = 0;

        for (j = 0; j < n - i - 1; j++)

        {

            if (A[j] > A[j + 1])

            {

                swap(A[j], A[j + 1]);

                flag = 1;

            }

        }

        if (flag == 0)

            break;

    }

}

**int** main()

{

**int** n;

    cout << "Enter the size of array: ";

    cin >> n;

**int** A[n];

    cout << "Enter the elements of array\n";

    for (**int** i = 0; i < n; i++)

    {

        cin >> A[i];

    }

    BubbleSort(A, n);

    cout << "Sorted Array" << endl;

    for (**int** i = 0; i < n; i++)

    {

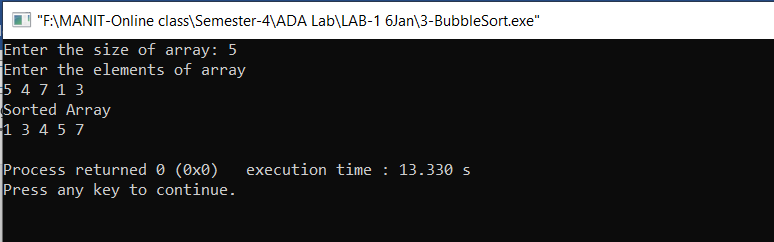
        cout << A[i] << " ";

    }

    cout << endl;

    return 0;

}



Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order

**Worst and Average Case Time Complexity: O(n\*n):** Worst case occurs when array is reverse sorted.

**Best Case Time Complexity: O(n):** Best case occurs when array is already sorted.

**Space: O (1)**

Bubble sort is **stable** sorting algorithm.