**Lab Assignment 2 by Gavisht Singh**

****

1) Implement the Binary search algorithm regarded as a fast search algorithm with

run-time complexity of Ο(log n) in comparison to the Linear Search.

**Solution**: -

#include <iostream>

using namespace std;

int main() {

    int arr[] = {2, 3, 4, 10, 40, 55, 67, 89};

    int n = sizeof(arr) / sizeof(arr[0]);

    int x = 10;

     int left = 0;

    int right = n - 1;

    while (left <= right) {

        int mid = (left+right)/ 2;

               if (arr[mid] == x) {

            cout<<”Element found at index”<<mid<<endl;

return 0;

        }

        if (arr[mid] < x) {

            left = mid + 1;

        }

        else {

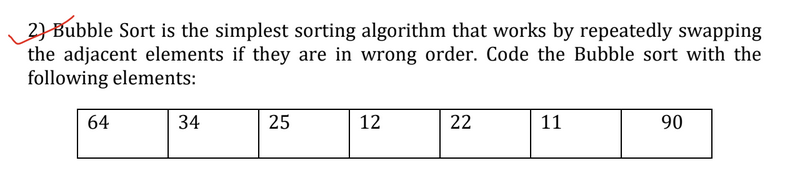
            right = mid - 1;

        }

    }

    return 0;

}



Solution: -

#include <iostream>

using namespace std;

int main() {

    int arr[] = {64, 34, 25, 12, 22, 11, 90};

    int n = sizeof(arr) / sizeof(arr[0]);

    for (int i = 0; i < n - 1; i++) {

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

            if (arr[j] > arr[j + 1]) {

swap(arr[j],arr[j+1]);            }

        }

    }

    for (int i = 0; i < n; i++) {

cout << arr[i] << " ";

}

    cout << endl;

    return 0;

}

3) Design the Logic to Find a Missing Number in a Sorted Array. Given an array of n-1

distinct integers in the range of 1 to n, find the missing number in it in a Sorted Array

(a) Linear time

(b) Using binary search.

a)

#include <iostream>

using namespace std;

int main() {

    int arr[] = {1, 2, 3, 4, 6, 7, 8};

    int n = sizeof(arr) / sizeof(arr[0]);

    int missingNumber = -1;

    for (int i = 0; i < n; i++) {

        if (arr[i] != i + 1) {

            missingNumber = i + 1;

            break;

        }

    }

    cout << "The missing number is: " << missingNumber << endl;

    return 0;

}

b)

#include <iostream>

using namespace std;

int main() {

    int arr[] = {1, 2, 3, 4, 6, 7, 8};

    int n = sizeof(arr) / sizeof(arr[0]);

    int missingNumber = -1;

    int left = 0, right = n - 1;

    while (left <= right) {

        int mid = (left+right) / 2;

        if (arr[mid] == mid + 1) {

            left = mid + 1;

        } else {

            missingNumber = mid + 1;

            right = mid - 1;

    }

    }

    cout << "The missing number is: " << missingNumber << endl;

    return 0;

}

4) String Related Programs

(a) Write a program to concatenate one string to another string.

(b) Write a program to reverse a string.

(c) Write a program to delete all the vowels from the string.

(d) Write a program to sort the strings in alphabetical order.

(e) Write a program to convert a character from uppercase to lowercase.

Solution: -

a)

#include <iostream>

#include <string>

using namespace std;

int main() {

    string str1 = "Hello, ";

    string str2 = "World!";

    string result = str1 + str2;

    cout << "Result: " << result << endl;

    return 0;

}

b)

#include <iostream>

#include <string>

using namespace std;

int main() {

    string str = "Programming";

    int n = str.length();

    for (int i = 0; i < n / 2; i++) {

        swap(str[i],str[n-i-1]);

    }

    cout << "Reversed: " << str << endl;

    return 0;

}

c)

#include <iostream>

#include <string>

using namespace std;

int main() {

    string str = "Apple";

    int n=str.length();

 string result=" ";

    for (int i=0;i<n;i++) {

        str[i]=tolower(str[i]);

        if (str[i]!= 'a' && (str[i]!= 'e' && str[i]!= 'i' && str[i]!= 'o' && str[i]!= 'u') {

            result =result+str[i];

        }

    }

    cout << "Without Vowels: " << result << endl;

    return 0;

}

d)

#include <iostream>

#include <string>

using namespace std;

int main() {

    string words[] = {"Apple", "Orange", "Banana", "Grape", "Cherry"};

    int n = 5;

    for (int i = 0; i < n - 1; i++) {

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

            if (words[j] > words[j + 1]) {

                swap(words[j],words[j+1]);

            }

        }

    }

    cout << "Sorted order: ";

    for (int i = 0; i < n; i++) {

        cout << words[i] << " ";

    }

    cout << endl;

    return 0;

}

e)

#include <iostream>

using namespace std;

int main() {

    char upperChar = 'A';

    char lowerChar = upperChar;

       if (upperChar >= 'A' && upperChar <= 'Z') {

        lowerChar = upperChar + 32;

    }

    cout << "Uppercase: " << upperChar << endl;

    cout << "Lowercase: " << lowerChar << endl;

    return 0;

}

5) Space required to store any two-dimensional array is number oƒ rows × number oƒ

columns. Assuming array is used to store elements of the following matrices,

implement an efficient way that reduces the space requirement.

(a) Diagonal Matrix.

(b) Tri-diagonal Matrix.

(c) Lower triangular Matrix.

(d) Upper triangular Matrix.

(e) Symmetric Matrix

Solution: -

a)

#include <iostream>

using namespace std;

int main() {

    int n = 4;

    int matrix[4][4] = {

        {5, 0, 0, 0},

        {0, 8, 0, 0},

        {0, 0, 9, 0},

        {0, 0, 0, 2}

    };

    int compactArray[4];

    for (int i = 0; i < n; i++) {

        compactArray[i] = matrix[i][i];

    }

    cout << "Compact 1D array: ";

    for(int i = 0; i < n; i++) cout << compactArray[i] << " ";

    cout << endl;

    return 0;

}

b)

#include <bits/stdc++.h>

using namespace std;

int main() {

    int n = 4;

    int matrix[4][4] = {

        {1, 2, 0, 0},

        {3, 4, 5, 0},

        {0, 6, 7, 8},

        {0, 0, 9, 10}

    };

    int Size = 3\*n - 2 ;

    int compactArray[Size];

    int k = 0;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if (abs(i - j) <= 1) {

                compactArray[k] = matrix[i][j];

                k++;

            }

        }

    }

    cout << "Compact 1D array: ";

    for(int i = 0; i < Size; i++) cout << compactArray[i] << " ";

    cout << endl;

    return 0;

}

c)

#include <iostream>

using namespace std;

int main() {

    int n = 4;

    int matrix[4][4] = {

        {1, 0, 0, 0},

        {2, 3, 0, 0},

        {4, 5, 6, 0},

        {7, 8, 9, 10}

    };

    int Size = n \* (n + 1) / 2 ;

    int compactArray[Size];

    int k = 0;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if (i >= j) {

                compactArray[k] = matrix[i][j];

                k++;

            }

        }

    }

    cout << "Compact 1D array: ";

    for(int i = 0; i < Size; i++){

cout << compactArray[i] << " ";

}

    cout << endl;

    return 0;

}

d)

#include <iostream>

using namespace std;

int main() {

    int n = 4;

    int matrix[4][4] = {

        {1, 2, 3, 4},

        {0, 5, 6, 7},

        {0, 0, 8, 9},

        {0, 0, 0, 10}

    };

    Int Size = n \* (n + 1) / 2 ;

    int compactArray[size];

    int k = 0;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if (j >= i) {

                compactArray[k] = matrix[i][j];

                k++;

            }

        }

    }

    cout << "Compact 1D array: ";

    for(int i = 0; i < Size; i++) cout << compactArray[i] << " ";

    cout << endl;

    return 0;

}

e)

#include <iostream>

using namespace std;

int main() {

    int n = 4;

    int matrix[4][4] = {

        {1, 2, 3, 4},

        {2, 5, 6, 7},

        {3, 6, 8, 9},

        {4, 7, 9, 10}

    };

    int Size = n \* (n + 1) / 2 ;

    int compactArray[Size];

    int k = 0;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if (i >= j) {

                compactArray[k] = matrix[i][j];

                k++;

            }

        }

    }

    cout << "Compact 1D array: ";

    for(int i = 0; i < Size; i++) cout << compactArray[i] << " ";

    cout << endl;

    return 0;

}

7) Let A[1 .... n] be an array of n real numbers. A pair (A[i], A[j ]) is said to be an

inversion if these numbers are out of order, i.e., i < j but A[i]>A[j ]. Write a program to

count the number of inversions in an array.

Solution: -

#include <iostream>

using namespace std;

int main() {

    int arr[] = {8, 4, 2, 1};

    int n = sizeof(arr) / sizeof(arr[0]);

    int inversionCount = 0;

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (arr[i] > arr[j]) {

                inversionCount++;

            }

        }

    }

    cout << "Total number of inversions: " << inversionCount << endl;

    return 0;

}

8) Write a program to count the total number of distinct elements in an array of length

n.

#include <iostream>

using namespace std;

int main() {

    int arr[] = {10, 20, 20, 10, 30, 10, 40};

    int n = sizeof(arr) / sizeof(arr[0]);

    int distinctCount = 0;

    for (int i = 0; i < n; i++) {

        bool isDuplicate = false;

        for (int j = 0; j < i; j++) {

            if (arr[i] == arr[j]) {

                isDuplicate = true;

                break;

            }

        }

        if (!isDuplicate) {

            distinctCount++;

        }

    }

    cout << "Total number of distinct elements: " << distinctCount << endl;

    return 0;

}

Write a program to find a saddle point in a two-dimensional array. A saddle

point in a numerical array is a number that is larger than or equal to every

number in its column, and smaller than or equal to every number in its row.

Solution:-

#include <iostream>

using namespace std;

int main() {

    int matrix[3][3] = {

        {9, 8, 7},

        {5, 3, 2},

        {6, 6, 7}

    };

    bool saddlePointFound = false;

    for (int i = 0; i < 3; i++) {

        int minInRow = matrix[i][0];

        int minColIndex = 0;

        for (int j = 1; j < 3; j++) {

            if (matrix[i][j] < minInRow) {

                minInRow = matrix[i][j];

                minColIndex = j;

            }

        }

        bool isMaxInCol = true;

        for (int k = 0; k < 3; k++) {

            if (matrix[k][minColIndex] > minInRow) {

                isMaxInCol = false;

                break;

            }

        }

        if (isMaxInCol) {

            cout << "\n>>> Saddle Point Found <<<" << endl;

            cout << "Value: " << minInRow << endl;

            saddlePointFound = true;

        }

    }

    if (!saddlePointFound) {

        cout << "No saddle point found." << endl;

    }

    return 0;

}