ASSIGNMENT 2&3[DSA]

Submitted by :Disha(1024030180)

Q1

LINEAR SEARCH

```cpp
#include <iostream>
using namespace std;

int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target)
            return i;
    }
    return -1;
}

int main() {
    int arr[] = {34, 78, 12, 56, 89, 23};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 56;
    int idx = linearSearch(arr, size, target);
    if (idx != -1)
        cout << "Element found at index " << idx << endl;
    else
        cout << "Element not found in the array." << endl;
    return 0;
}

```

BINARY SEARCH

```cpp
#include <iostream> //binary search
using namespace std;

int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;  // safe against overflow
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {12, 23, 34, 56, 78, 89};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 56;
    int idx = binarySearch(arr, size, target);
    if (idx != -1)
```

```
    while (left <= right) {
        int mid = left + (right - left) / 2;  // safe against overflow
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {12, 23, 34, 56, 78, 89};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 56;
    int idx = binarySearch(arr, size, target);
    if (idx != -1)
        cout << "Element found at index " << idx << endl;
    else
        cout << "Element not found in the array." << endl;
    return 0;
}
```

Q2

```
#include <iostream>
using namespace std;

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

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

```
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    printArray(arr, size);

    bubbleSort(arr, size);

    cout << "Sorted array:   ";
    printArray(arr, size);

    return 0;
}
```

Q3

```
#include <iostream>
using namespace std;

int findMissingLinear(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        if (arr[i] != i + 1) {
            return i + 1;
        }
    }
    return size + 1;  // Missing number is n
}

int main() {
    int arr[] = {1, 2, 3, 4, 6, 7, 8};  // n = 8, size = 7
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Missing number (linear): " << findMissingLinear(arr, size) <<
        endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int findMissingBinary(int arr[], int size) {
    int left = 0, right = size - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        // Found first mismatch:
        if (arr[mid] != mid + 1 &&
            (mid == 0 || arr[mid - 1] == mid)) {
            return mid + 1;
        }
        if (arr[mid] == mid + 1) {
            // Everything up to mid is correct—go right
            left = mid + 1;
        } else {
            // Mismatch occurs on or before mid—go left
            right = mid - 1;
        }
    }
    return -1;  // Shouldn't happen if input guarantees one missing
}

int main() {
```

```cpp
        if (arr[mid] != mid + 1 &&
            (mid == 0 || arr[mid - 1] == mid)) {
            return mid + 1;
        }
        if (arr[mid] == mid + 1) {
            // Everything up to mid is correct—go right
            left = mid + 1;
        } else {
            // Mismatch occurs on or before mid—go left
            right = mid - 1;
        }
    }
    return -1;  // Shouldn't happen if input guarantees one missing
}

int main() {
    int arr[] = {1, 2, 3, 4, 6, 7, 8};  // n = 8, size = 7
    int size = sizeof(arr) / sizeof(arr[0]);
    cout << "Missing number (binary): " << findMissingBinary(arr, size) <<
        endl;
    return 0;
}
```

Q4 a

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string s1 = "Hello, ";
    string s2 = "World!";
    string result = s1 + s2;  // or s1.append(s2);
    cout << "Concatenated string: " << result << endl;
    return 0;
}
```

B

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main() {
    string s = "Hello, world!";
    reverse(s.begin(), s.end());
    cout << "Reversed string: " << s << endl;
    return 0;
}
```

C

main.cpp Share Run

```cpp
#include <iostream>
using namespace std;

bool isVowel(char c) {
    c = tolower(c);
    return c=='a' || c=='e' || c=='i' || c=='o' || c=='u';
}

int main() {
    char str[100];
    cout << "Enter string: ";
    cin.getline(str, 100);

    int j = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (!isVowel(str[i])) {
            str[j++] = str[i];
        }
    }
    str[j] = '\0';

    cout << "After deleting vowels: " << str << endl;
    return 0;
}
```

D

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main() {
    vector<string> arr = {"banana", "apple", "cherry", "date"};
    sort(arr.begin(), arr.end());
    cout << "Sorted strings:\n";
    for (const auto &s : arr) {
        cout << s << endl;
    }
    return 0;
}
```

E

```cpp
#include <iostream>
using namespace std;

int main() {
    string s = "HeLLo, WoRLD!";
    for (char &c : s) {
        if (c >= 'A' && c <= 'Z') {
            c = c + ('a' - 'A');  // Add 32
        }
    }
    cout << "Lowercase string: " << s << endl;
    return 0;
}
```

Q5

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 4; // Example size
    int diag[4] = {1, 2, 3, 4}; // Diagonal Matrix -> only n elements

    int tri[10]; // Tri-diagonal Matrix -> 3n-2 elements
    int lower[10]; // Lower triangular -> n(n+1)/2 elements
    int upper[10]; // Upper triangular -> n(n+1)/2 elements
    int sym[10];   // Symmetric -> n(n+1)/2 elements

    cout << "Space Efficient Storage Demonstrated" << endl;
    return 0;
}
```

Q6

```cpp
int main() {
    int r1=3, c1=3, r2=3, c2=3;

    // Triplet form: row col value
    int sparse1[4][3] = {{0,0,1},{0,2,2},{1,1,3},{2,0,4}};
    int sparse2[3][3] = {{0,1,5},{1,2,6},{2,2,7}};

    cout << "Transpose of sparse1:" << endl;
    for(int i=0;i<4;i++){
        cout << sparse1[i][1] << " " << sparse1[i][0] << " " <<
            sparse1[i][2] << endl;
    }

    cout << "Addition of sparse1 and sparse2:" << endl;
    for(int i=0;i<4;i++) cout << sparse1[i][0] << " " << sparse1[i][1] << "
        " << sparse1[i][2] << endl;
    for(int i=0;i<3;i++) cout << sparse2[i][0] << " " << sparse2[i][1] << "
        " << sparse2[i][2] << endl;

    cout << "Multiplication not shown fully (too complex), but idea:
        multiply triplets row-wise * column-wise" << endl;

    return 0;
}
```

Q7

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 5;
    int A[5] = {2, 4, 1, 3, 5};
    int count = 0;

    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(A[i] > A[j]) count++;
        }
    }

    cout << "Number of inversions = " << count << endl;
    return 0;
}
```