ASSIGMENT 2

QUESTION 1

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

int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key) {
            return mid; // found
        }
        else if (arr[mid] < key) {
            low = mid + 1; // search right half
        }
        else {
            high = mid - 1; // search left half
        }
    }
    return -1; // not found
}

int main() {
```

```
    int arr[100], n, key;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements in sorted order:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Enter the element to search: ";
    cin >> key;

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

    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in the array." << endl;

    return 0;
}
```

QUESTION 2

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

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Last i elements are already sorted
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

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

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

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

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

    bubbleSort(arr, n);

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

    return 0;
}
```

QUESTION 3

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

// Linear Time Method
int findMissingLinear(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] != i + 1) {
            return i + 1;  // Missing number found
        }
    }
    return n; // If no mismatch, missing number is last
}

// Binary Search Method
int findMissingBinary(int arr[], int n) {
    int left = 0, right = n - 2; // because array size = n-1

    while (left <= right) {
        int mid = (left + right) / 2;

        if (arr[mid] == mid + 1) {
            left = mid + 1;  // Missing number is on right side
        } else {
            right = mid - 1; // Missing number is on left side
        }
    }
```

```
    return left + 1;
}

int main() {
    int arr[] = {1, 2, 3, 5, 6, 7, 8}; // Example: n=8, missing=4
    int n = 8;

    cout << "Missing Number (Linear): " << findMissingLinear(arr, n) << endl;
    cout << "Missing Number (Binary Search): " << findMissingBinary(arr, n) << endl;

    return 0;
}
```

QUESTION 4

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

// (a) Concatenate one string to another
void concatenateStrings() {
    char str1[100], str2[100];
    cout << "Enter first string: ";
    cin.ignore(); // to clear buffer
    cin.getline(str1, 100);
    cout << "Enter second string: ";
    cin.getline(str2, 100);
    strcat(str1, str2);
    cout << "Concatenated String: " << str1 << endl;
}

// (b) Reverse a string
void reverseString() {
    char str[100];
    cout << "Enter a string: ";
    cin.ignore();
    cin.getline(str, 100);
    int len = strlen(str);
    for (int i = len - 1; i >= 0; i--)
        cout << str[i];
```

```
    cout << endl;
}

// (c) Delete all vowels from string
void deleteVowels() {
    char str[100], result[100];
    cout << "Enter a string: ";
    cin.ignore();
    cin.getline(str, 100);

    int j = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        char ch = tolower(str[i]);
        if (ch != 'a' && ch != 'e' && ch != 'i' && ch != 'o' && ch != 'u') {
            result[j++] = str[i];
        }
    }
    result[j] = '\0';
    cout << "String without vowels: " << result << endl;
}

// (d) Sort strings in alphabetical order
void sortStrings() {
    int n;
    cout << "How many strings do you want to sort? ";
    cin >> n;
    cin.ignore();
```

```
    char arr[20][100];  // max 20 strings, each up to 100 chars
    cout << "Enter " << n << " strings:\n";
    for (int i = 0; i < n; i++) {
        cin.getline(arr[i], 100);
    }

    // Bubble sort on strings
    char temp[100];
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (strcmp(arr[i], arr[j]) > 0) {
                strcpy(temp, arr[i]);
                strcpy(arr[i], arr[j]);
                strcpy(arr[j], temp);
            }
        }
    }

    cout << "Strings in alphabetical order:\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << endl;
    }
}

// (e) Convert uppercase to lowercase
void convertCase() {
    char str[100];
    cout << "Enter a string (with uppercase letters): ";
```

```
    cin.ignore();
    cin.getline(str, 100);

    for (int i = 0; str[i] != '\0'; i++) {
        str[i] = tolower(str[i]);
    }
    cout << "Converted string (lowercase): " << str << endl;
}

// Main menu
int main() {
    int choice;
    do {
        cout << "\n--- String Operations Menu ---\n";
        cout << "1. Concatenate Strings\n";
        cout << "2. Reverse String\n";
        cout << "3. Delete Vowels\n";
        cout << "4. Sort Strings Alphabetically\n";
        cout << "5. Convert Uppercase to Lowercase\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1: concatenateStrings(); break;
        case 2: reverseString(); break;
        case 3: deleteVowels(); break;
        case 4: sortStrings(); break;
```

```
        case 5: convertCase(); break;
        case 6: cout << "Exiting...\n"; break;
        default: cout << "Invalid choice! Try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

QUESTION 5

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

// 1. Diagonal Matrix
void diagonalMatrix() {
    int n;
    cout << "Enter size of diagonal matrix: ";
    cin >> n;
    int D[n];

    cout << "Enter diagonal elements:\n";
    for (int i = 0; i < n; i++) cin >> D[i];

    cout << "Full Diagonal Matrix:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) cout << D[i] << " ";
            else cout << 0 << " ";
        }
        cout << endl;
    }
}

// 2. Tri-diagonal Matrix
void triDiagonalMatrix() {
```

```
    int n;
    cout << "Enter size of tri-diagonal matrix: ";
    cin >> n;
    int T[3*n-2];

    cout << "Enter lower diagonal elements (n-1 elements):\n";
    for(int i = 1; i < n; i++) cin >> T[i-1];

    cout << "Enter main diagonal elements (n elements):\n";
    for(int i = 0; i < n; i++) cin >> T[n-1 + i];

    cout << "Enter upper diagonal elements (n-1 elements):\n";
    for(int i = 0; i < n-1; i++) cin >> T[2*n-1 + i];

    cout << "Full Tri-diagonal Matrix:\n";
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            if(i-j == 1) cout << T[i-1] << " ";
            else if(i == j) cout << T[n-1 + i] << " ";
            else if(j-i == 1) cout << T[2*n-1 + i] << " ";
            else cout << 0 << " ";
        }
        cout << endl;
    }
}

// 3. Lower Triangular Matrix
void lowerTriangularMatrix() {
```

```
    int n;
    cout << "Enter size of lower triangular matrix: ";
    cin >> n;
    int LT[n*(n+1)/2];

    cout << "Enter lower triangular elements row-wise:\n";
    for(int i = 0; i < n; i++)
        for(int j = 0; j <= i; j++)
            cin >> LT[i*(i+1)/2 + j];

    cout << "Full Lower Triangular Matrix:\n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i >= j) cout << LT[i*(i+1)/2 + j] << " ";
            else cout << 0 << " ";
        }
        cout << endl;
    }
}

// 4. Upper Triangular Matrix
void upperTriangularMatrix() {
    int n;
    cout << "Enter size of upper triangular matrix: ";
    cin >> n;
    int UT[n*(n+1)/2];

    cout << "Enter upper triangular elements row-wise:\n";
```

```
    for(int i = 0; i < n; i++)
        for(int j = i; j < n; j++)
            cin >> UT[i*n - i*(i-1)/2 + (j-i)];

    cout << "Full Upper Triangular Matrix:\n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i <= j) cout << UT[i*n - i*(i-1)/2 + (j-i)] << " ";
            else cout << 0 << " ";
        }
        cout << endl;
    }
}

// 5. Symmetric Matrix
void symmetricMatrix() {
    int n;
    cout << "Enter size of symmetric matrix: ";
    cin >> n;
    int S[n*(n+1)/2]; // store lower triangle

    cout << "Enter lower triangular elements row-wise:\n";
    for(int i = 0; i < n; i++)
        for(int j = 0; j <= i; j++)
            cin >> S[i*(i+1)/2 + j];

    cout << "Full Symmetric Matrix:\n";
    for(int i = 0; i < n; i++){
```

```
        for(int j = 0; j < n; j++){
            if(i >= j) cout << S[i*(i+1)/2 + j] << " ";
            else cout << S[j*(j+1)/2 + i] << " "; // symmetric
        }
        cout << endl;
    }
}

int main() {
    int choice;
    do {
        cout << "\n--- Efficient Matrix Storage Menu ---\n";
        cout << "1. Diagonal Matrix\n";
        cout << "2. Tri-diagonal Matrix\n";
        cout << "3. Lower Triangular Matrix\n";
        cout << "4. Upper Triangular Matrix\n";
        cout << "5. Symmetric Matrix\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1: diagonalMatrix(); break;
            case 2: triDiagonalMatrix(); break;
            case 3: lowerTriangularMatrix(); break;
            case 4: upperTriangularMatrix(); break;
            case 5: symmetricMatrix(); break;
            case 6: cout << "Exiting...\n"; break;
```

```
            default: cout << "Invalid choice! Try again.\n";
        }
    } while(choice != 6);


    return 0;
}
```

QUESTION 6

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

struct Triplet {
    int row;
    int col;
    int val;
};

// Function to display a sparse matrix
void displaySparse(Triplet t[], int size) {
    cout << "Row  Column  Value\n";
    for (int i = 0; i < size; i++) {
        cout << t[i].row << "      " << t[i].col << "      " << t[i].val << endl;
    }
}

// (a) Transpose of a sparse matrix
void transposeSparse(Triplet t[], int size) {
    Triplet trans[size];
    for (int i = 0; i < size; i++) {
        trans[i].row = t[i].col;
        trans[i].col = t[i].row;
        trans[i].val = t[i].val;
    }
    cout << "\nTransposed Sparse Matrix:\n";
    displaySparse(trans, size);
}

// (b) Addition of two sparse matrices
```

```
void addSparse(Triplet A[], int sizeA, Triplet B[], int sizeB) {
    Triplet sum[sizeA + sizeB];
    int i = 0, j = 0, k = 0;

    while (i < sizeA && j < sizeB) {
        if (A[i].row < B[j].row || (A[i].row == B[j].row && A[i].col < B[j].col)) {
            sum[k++] = A[i++];
        } else if (A[i].row > B[j].row || (A[i].row == B[j].row && A[i].col > B[j].col)) {
            sum[k++] = B[j++];
        } else {
            // Same position
            sum[k] = A[i];
            sum[k].val += B[j].val;
            i++; j++;
            if (sum[k].val != 0) k++;
        }
    }

    while (i < sizeA) sum[k++] = A[i++];
    while (j < sizeB) sum[k++] = B[j++];

    cout << "\nSum of Sparse Matrices:\n";
    displaySparse(sum, k);
}

// (c) Multiplication of two sparse matrices
void multiplySparse(Triplet A[], int sizeA, int rA, int cA, Triplet B[], int sizeB, int rB, int cB) {
    if (cA != rB) {
        cout << "Matrix multiplication not possible.\n";
        return;
    }
```

```
    int result[rA][cB] = {0};

    // Convert A and B to normal 2D arrays temporarily
    for (int i = 0; i < sizeA; i++) result[A[i].row][A[i].col] += A[i].val;
    int Bmat[rB][cB] = {0};
    for (int i = 0; i < sizeB; i++) Bmat[B[i].row][B[i].col] = B[i].val;

    // Multiply
    for (int i = 0; i < rA; i++)
        for (int j = 0; j < cB; j++)
            for (int k = 0; k < cA; k++)
                result[i][j] += result[i][k] * Bmat[k][j];

    // Convert back to triplet
    Triplet resTriplet[rA*cB];
    int count = 0;
    for (int i = 0; i < rA; i++)
        for (int j = 0; j < cB; j++)
            if (result[i][j] != 0) {
                resTriplet[count].row = i;
                resTriplet[count].col = j;
                resTriplet[count].val = result[i][j];
                count++;
            }

    cout << "\nProduct of Sparse Matrices (Triplet Form):\n";
    displaySparse(resTriplet, count);
}

int main() {
```

```
int choice;

do {
    cout << "\n--- Sparse Matrix Operations Menu ---\n";
    cout << "1. Transpose\n2. Addition\n3. Multiplication\n4. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1: {
            int n, rows, cols;
            cout << "Enter number of non-zero elements: ";
            cin >> n;
            Triplet t[n];
            cout << "Enter row, column, value for each element:\n";
            for (int i = 0; i < n; i++) cin >> t[i].row >> t[i].col >> t[i].val;
            transposeSparse(t, n);
            break;
        }

        case 2: {
            int nA, nB;
            cout << "Enter number of non-zero elements in Matrix A: ";
            cin >> nA;
            Triplet A[nA];
            cout << "Enter row, column, value for each element of A:\n";
            for (int i = 0; i < nA; i++) cin >> A[i].row >> A[i].col >> A[i].val;

            cout << "Enter number of non-zero elements in Matrix B: ";
            cin >> nB;
            Triplet B[nB];
```

```
    cout << "Enter row, column, value for each element of B:\n";
    for (int i = 0; i < nB; i++) cin >> B[i].row >> B[i].col >> B[i].val;

    addSparse(A, nA, B, nB);
    break;
}

case 3: {
    int nA, nB, rA, cA, rB, cB;
    cout << "Enter rows and cols of Matrix A: ";
    cin >> rA >> cA;
    cout << "Enter number of non-zero elements in A: ";
    cin >> nA;
    Triplet A[nA];
    cout << "Enter row, column, value for each element of A:\n";
    for (int i = 0; i < nA; i++) cin >> A[i].row >> A[i].col >> A[i].val;

    cout << "Enter rows and cols of Matrix B: ";
    cin >> rB >> cB;
    cout << "Enter number of non-zero elements in B: ";
    cin >> nB;
    Triplet B[nB];
    cout << "Enter row, column, value for each element of B:\n";
    for (int i = 0; i < nB; i++) cin >> B[i].row >> B[i].col >> B[i].val;

    multiplySparse(A, nA, rA, cA, B, nB, rB, cB);
    break;
}

case 4: cout << "Exiting...\n"; break;
default: cout << "Invalid choice!\n";
```

```
        }

    } while(choice != 4);

    return 0;
}
```

QUESTION 7

```
#include <iostream>

using namespace std;


int countInversionsBrute(int arr[], int n) {
   int count = 0;
   for(int i = 0; i < n-1; i++)
      for(int j = i+1; j < n; j++)
         if(arr[i] > arr[j])
            count++;
   return count;
}


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


   int inversions = countInversionsBrute(arr, n);
   cout << "Number of inversions: " << inversions << endl;


   return 0;
}
```

QUESTION 8

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

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

    cout << "Enter elements:\n";
    for(int i = 0; i < n; i++) cin >> arr[i];

    // Sort the array
    sort(arr, arr + n);

    // Count distinct elements
    int count = 1; // first element is always distinct if n>0
    for(int i = 1; i < n; i++) {
        if(arr[i] != arr[i-1]) count++;
    }

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

    return 0;
}
```