Experiment 1

Student Name:Harsh Kumar **UID:22BCS15754**
Branch:BE-CSE **Section/Group:FL_IOT_603(B)**
Semester:5th **Date of Performance:22/07/24**
Subject Name:Advanced Programming **Subject Code:22CSP-314**

1. **Aim: (A)** To compute the absolute difference between the sums of the left and right diagonals of a 3x3 matrix using two different time complexities:$O(n^2)$ and $O(n)$.

   (B) To calculate the total number of hourglasses in a given m×n integer matrix and find the minimum sum of the elements of an hourglass

2. **Objective:(A)**

1. Implement a function to input a 3x3 matrix.
2. Implement a function to calculate the sums of the primary and secondary diagonals with time complexity $O(n^2)$
3. Implement a function to calculate the sums of the primary and secondary diagonals with time complexity $O(n)$
4. Print the absolute difference between the sums of the diagonals.

(B)

1. Implement a function to count the total number of hourglasses in an m×n matrix.
2. Implement a function to find the minimum sum of the elements of an hourglass in the matrix.
3. Print the total number of hourglasses and the minimum hourglass sum.

3. **Implementation/Code:**

Approach with Time Complexity $O(n^2)$

```
#include <iostream>
#include <cmath>
using namespace std;
void getMatrixInput(int matrix[3][3]) {
  cout << "Enter the elements of the 3x3 matrix, row by row:\n";
  for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cin >> matrix[i][j];
        } }}
void calculateDiagonalSumsOn2(int matrix[3][3], int &primaryDiagonalSum, int
&secondaryDiagonalSum) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (i == j) primaryDiagonalSum += matrix[i][j];
            if (i + j == 2) secondaryDiagonalSum += matrix[i][j];
        }
    }
}

void printAbsoluteDifference(int primaryDiagonalSum, int secondaryDiagonalSum) {
    int difference = abs(primaryDiagonalSum - secondaryDiagonalSum);
    cout << "The absolute difference between the sums of the diagonals is: " << difference
<< endl;
}
int main() {
    int matrix[3][3];
    int primaryDiagonalSum = 0;
    int secondaryDiagonalSum = 0;
getMatrixInput(matrix);
    calculateDiagonalSumsOn2(matrix, primaryDiagonalSum, secondaryDiagonalSum);
    printAbsoluteDifference(primaryDiagonalSum, secondaryDiagonalSum);
  return 0;
}
```

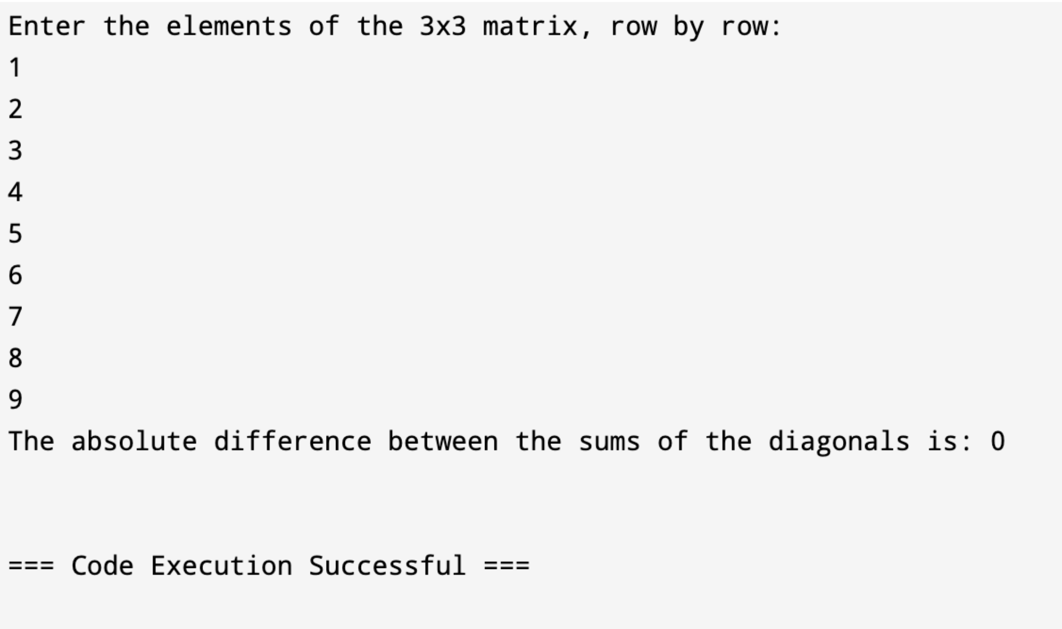
Approach with Time Complexity O(n)

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

const int MAX_DIMENSION = 100; // Define the maximum size of the matrix
int computeDiagonalDifference(int grid[MAX_DIMENSION][MAX_DIMENSION], int
dimension) {
    int primarySum = 0;
    int secondarySum = 0;
    for (int i = 0; i < dimension; ++i) {
        primarySum += grid[i][i];
        secondarySum += grid[i][dimension - i - 1];
    }
    return abs(primarySum - secondarySum);
}

int main() {
    int dimension;
    cout << "Enter the size of the matrix: ";
    cin >> dimension;

    int grid[MAX_DIMENSION][MAX_DIMENSION];
    cout << "Enter the elements of the matrix:" << endl;
    for (int i = 0; i < dimension; ++i) {
        for (int j = 0; j < dimension; ++j) {
            cin >> grid[i][j];
        }
    }

    int result = computeDiagonalDifference(grid, dimension);
    cout << "Absolute difference between diagonal sums: " << result << endl;

    return 0;
}
```

(B) Code Implementation For Hour Glass

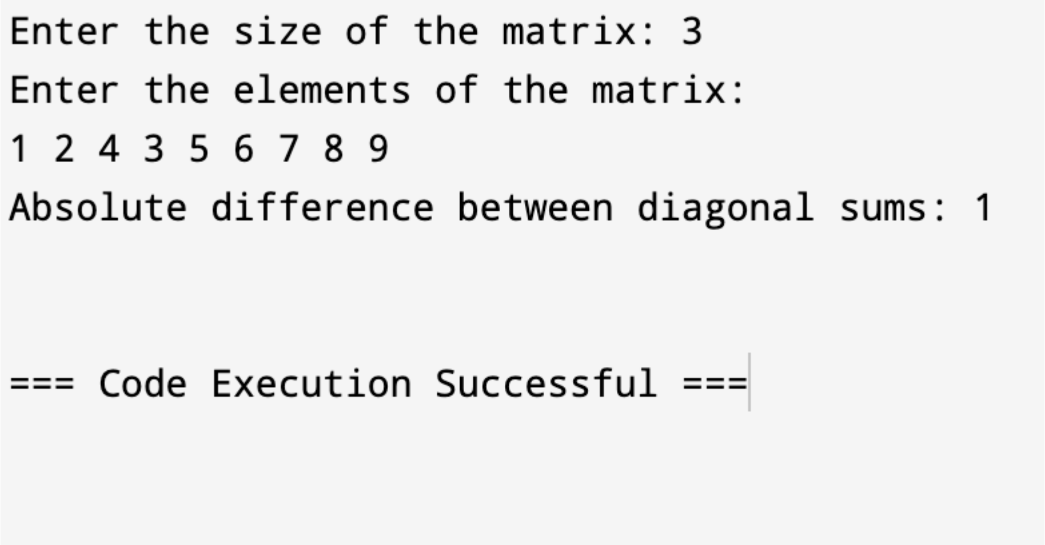
```
#include <iostream>
#include <climits>
using namespace std;
const int MAX_ROWS = 100; // Define maximum rows
const int MAX_COLS = 100; // Define maximum columns
int countHourglasses(int matrix[MAX_ROWS][MAX_COLS], int rows, int cols) {
    if (rows < 3 || cols < 3) return 0; // No hourglasses possible if matrix is smaller than 3x3
    return (rows - 2) * (cols - 2);
}
int minHourglassSum(int matrix[MAX_ROWS][MAX_COLS], int rows, int cols) {
    if (rows < 3 || cols < 3) return INT_MAX; // No hourglasses possible if matrix is smaller
than 3x3
    int minSum = INT_MAX;
    for (int i = 0; i <= rows - 3; ++i) {
        for (int j = 0; j <= cols - 3; ++j) {
            int sum = matrix[i][j] + matrix[i][j + 1] + matrix[i][j + 2]
                    + matrix[i + 1][j + 1]
                    + matrix[i + 2][j] + matrix[i + 2][j + 1] + matrix[i + 2][j + 2];
            if (sum < minSum) {
                minSum = sum;
            }
        }
    }
    return minSum;
}
int main() {
    int rows = 4, cols = 5;
    int matrix[MAX_ROWS][MAX_COLS] = {
        {1, 2, 3, 0, 0},
        {0, 0, 0, 0, 0},
        {2, 1, 4, 1, 0},
        {0, 0, 0, 0, 0}
    };
    int totalHourglasses = countHourglasses(matrix, rows, cols);
    int minimumHourglassSum = minHourglassSum(matrix, rows, cols);
    cout << "Total number of hourglasses: " << totalHourglasses << endl;
    cout << "Minimum hourglass sum: " << minimumHourglassSum << endl;
    return 0;
}
```

Output for $O(n^2)$



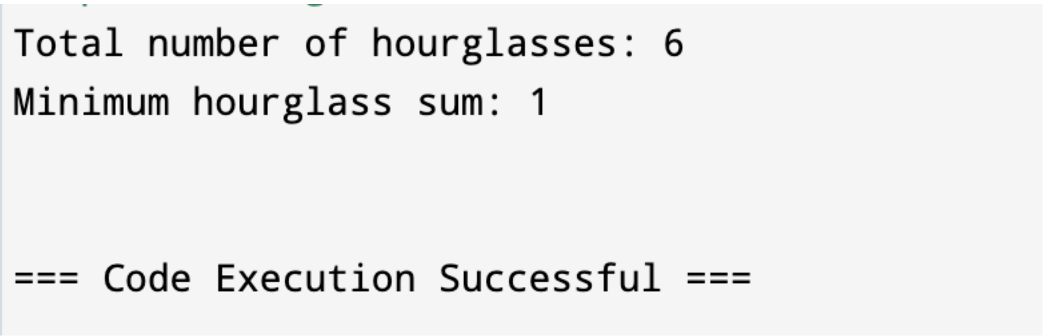
```
Enter the elements of the 3x3 matrix, row by row:
1
2
3
4
5
6
7
8
9
The absolute difference between the sums of the diagonals is: 0


=== Code Execution Successful ===
```

Output for O(n)



```
Enter the size of the matrix: 3
Enter the elements of the matrix:
1 2 4 3 5 6 7 8 9
Absolute difference between diagonal sums: 1


=== Code Execution Successful ===
```

OUTPUT (B)



```
Total number of hourglasses: 6
Minimum hourglass sum: 1


=== Code Execution Successful ===
```

4. Time Complexity

(A)

$O(n^2)$ approach: The function `calculateDiagonalSumsOn2` iterates over all elements of the matrix, leading to $O(n^2)$ time complexity.

The function `calculateDiagonalDifference` has a time complexity of $O(n)$ as it iterates through the primary and secondary diagonals of the matrix.

(B)

- The function `countHourglasses` has a time complexity of $O(1)$ as it involves simple arithmetic operations.
- The function `minHourglassSum` has a time complexity of $O(m \times n)$ as it iterates through each possible hourglass in the matrix.