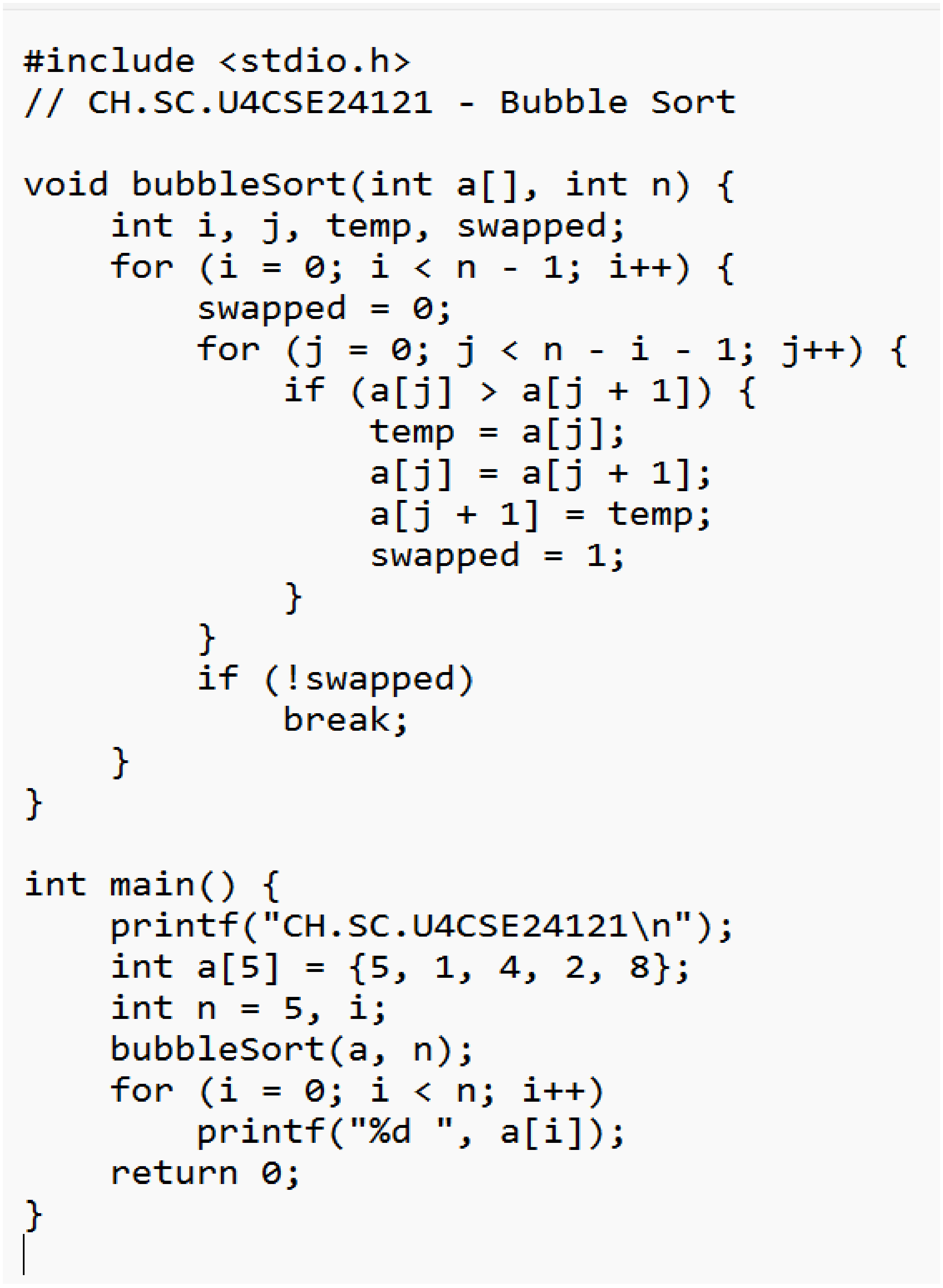
NAME: KURRA VENKATA GOKUL

CH.SC.U4CSE24121

CSE-B

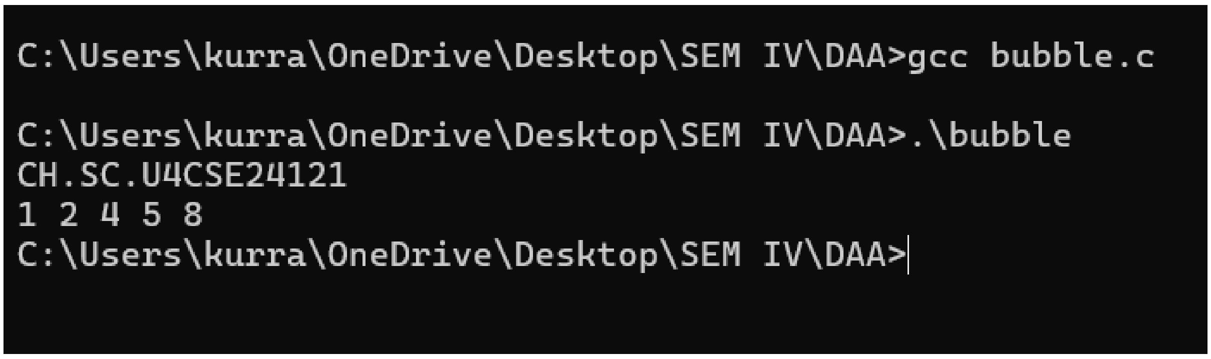
1.BUBBLE SORT

CODE:



```c
#include <stdio.h>
// CH.SC.U4CSE24121 - Bubble Sort

void bubbleSort(int a[], int n) {
    int i, j, temp, swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = 0;
        for (j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
                swapped = 1;
            }
        }
        if (!swapped)
            break;
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[5] = {5, 1, 4, 2, 8};
    int n = 5, i;
    bubbleSort(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

OUTPUT:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc bubble.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\bubble
CH.SC.U4CSE24121
1 2 4 5 8
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Time Complexity:

• Best Case: O(n)
• Average Case: O(n^2)
• Worst Case: O(n^2)

Justification:

The algorithm uses two nested loops.

- The outer loop runs (n − 1) times.
- The inner loop compares adjacent elements.
- In the best case (already sorted), the loop terminates early.
  Hence, time complexity is O(n^2) in average and worst cases.

Space Complexity:

• Space Complexity: O(n)

Justification:

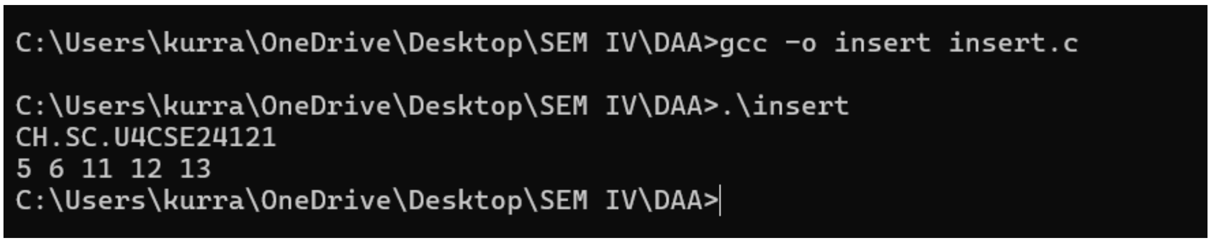
The program uses an array of size n to store elements.
Only a few extra variables (i, j, temp, swapped) are used.
No additional memory is allocated.
Therefore, space complexity is O(n).

2. Insertion Sort

Code:

```c
#include <stdio.h>
// CH.SC.U4CSE24121 - Insertion Sort

void insertionSort(int a[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[5] = {12, 11, 13, 5, 6};
    int n = 5, i;
    insertionSort(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o insert insert.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\insert
CH.SC.U4CSE24121
5 6 11 12 13
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Time Complexity:

• Best Case: O(n)
• Average Case: $O(n^2)$
• Worst Case: $O(n^2)$

Justification:

The outer loop runs (n − 1) times.

- In the best case, the array is already sorted and no shifting is needed.
- In the worst case, each element is compared with all previous elements.
  Hence, time complexity is $O(n^2)$.

Space Complexity:

• Space Complexity: O(n)

Justification:

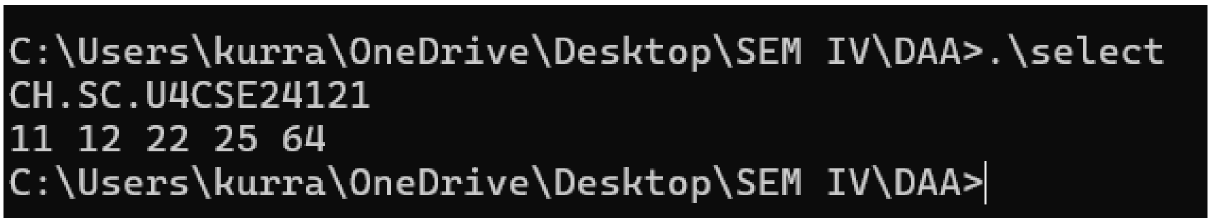
The program stores elements in an array of size n.
Only constant extra variables (i, j, key) are used.
No extra data structures are created.
Therefore, space complexity is O(n).

3.selection sort

Code:

```
#include <stdio.h>
// CH.SC.U4CSE24121 - Selection Sort

void selectionSort(int a[], int n) {
    int i, j, min, temp;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (a[j] < a[min])
                min = j;
        }
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[5] = {64, 25, 12, 22, 11};
    int n = 5, i;
    selectionSort(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\select
CH.SC.U4CSE24121
11 12 22 25 64
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Time Complexity:

• Best Case: O(n^2)

• Average Case: O(n^2)

• Worst Case: O(n^2)

Justification:

The algorithm uses two nested loops.

- The inner loop always scans the unsorted part of the array.
- The number of comparisons remains the same in all cases.
  Hence, time complexity is O(n^2).

Space Complexity:

• Space Complexity: O(n)

Justification:

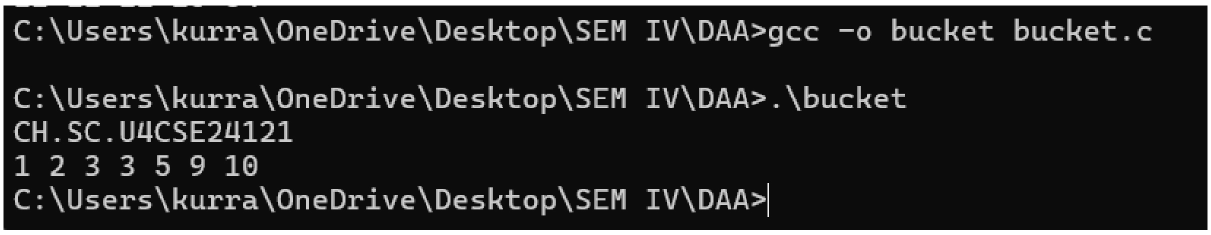
The input array of size n is used.
Only a few extra variables (i, j, min, temp) are used.
No additional memory is required.
So, space complexity is O(n).

4.bucket sort

Code:

```
#include <stdio.h>
// CH.SC.U4CSE24121 - Bucket Sort

void bucketSort(int a[], int n) {
    int bucket[100] = {0};
    int i;
    for (i = 0; i < n; i++)
        bucket[a[i]]++;

    int index = 0;
    for (i = 0; i < 100; i++) {
        while (bucket[i] > 0) {
            a[index++] = i;
            bucket[i]--;
        }
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[7] = {10, 3, 5, 2, 9, 3, 1};
    int n = 7, i;
    bucketSort(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o bucket bucket.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\bucket
CH.SC.U4CSE24121
1 2 3 3 5 9 10
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Time Complexity:

• Best Case: O(n + k)

• Average Case: O(n + k)

• Worst Case: O(n + k)

(where k is the number of buckets)

Justification:

- One loop inserts n elements into buckets.
- Another loop scans all k buckets.
  Since k is fixed, total time depends mainly on n.

Space Complexity:

• Space Complexity: O(n + k)

Justification:

The program uses:

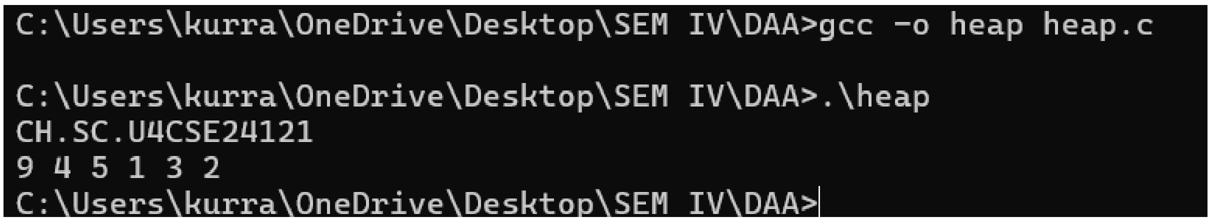
- Input array of size n
- Bucket array of size k
  Therefore, total space used is O(n + k).

5. Max heap

Code:

```c
#include <stdio.h>
// CH.SC.U4CSE24121 - Max Heap

void maxHeapify(int a[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    int temp;

    if (left < n && a[left] > a[largest])
        largest = left;
    if (right < n && a[right] > a[largest])
        largest = right;

    if (largest != i) {
        temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;
        maxHeapify(a, n, largest);
    }
}

void buildMaxHeap(int a[], int n) {
    int i;
    for (i = n / 2 - 1; i >= 0; i--)
        maxHeapify(a, n, i);
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[6] = {3, 9, 2, 1, 4, 5};
    int n = 6, i;
    buildMaxHeap(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o heap heap.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\heap
CH.SC.U4CSE24121
9 4 5 1 3 2
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>
```

Time Complexity:

• Best Case: O(n)

• Average Case: O(n)

• Worst Case: O(n)

Justification:

Heap is built by calling heapify from n/2 − 1 to 0.
Heapify takes logarithmic time but fewer operations are needed at lower levels.
Overall heap construction takes O(n) time.

Space Complexity:

• Space Complexity: O(n)

Justification:

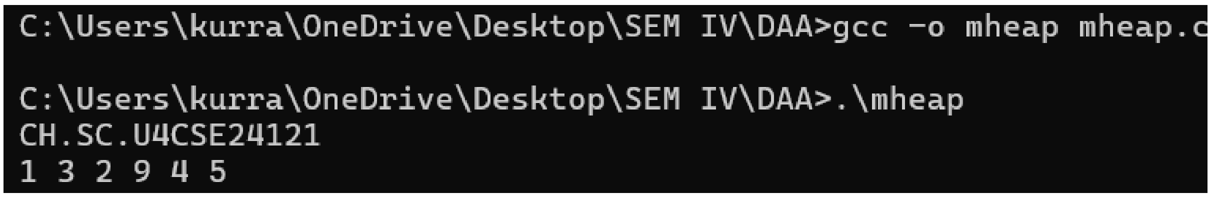
The heap is stored in an array of size n.
Only recursive calls and a few variables are used.
Hence, space complexity is O(n).

6.Min Heap

Code:

```
#include <stdio.h>
// CH.SC.U4CSE24121 - Min Heap

void minHeapify(int a[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    int temp;

    if (left < n && a[left] < a[smallest])
        smallest = left;
    if (right < n && a[right] < a[smallest])
        smallest = right;

    if (smallest != i) {
        temp = a[i];
        a[i] = a[smallest];
        a[smallest] = temp;
        minHeapify(a, n, smallest);
    }
}

void buildMinHeap(int a[], int n) {
    int i;
    for (i = n / 2 - 1; i >= 0; i--)
        minHeapify(a, n, i);
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int a[6] = {3, 9, 2, 1, 4, 5};
    int n = 6, i;
    buildMinHeap(a, n);
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:


```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o mheap mheap.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\mheap
CH.SC.U4CSE24121
1 3 2 9 4 5
```

Time Complexity:

• Best Case: O(n)

• Average Case: O(n)

• Worst Case: O(n)

Justification:

Similar to max heap.
Heapify is applied to non-leaf nodes only.
Overall heap construction takes linear time O(n).

Space Complexity:

• Space Complexity: O(n)

Justification:

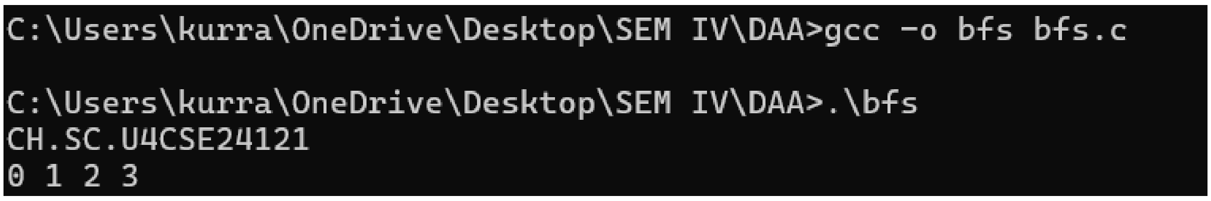
The array of size n stores heap elements.
Only recursion stack and few variables are used.
So, space complexity is O(n).

7.BFS

Code:

```c
#include <stdio.h>
#define MAX 100
// CH.SC.U4CSE24121 - BFS

int queue[MAX], front = 0, rear = 0;

void enqueue(int x) {
    queue[rear++] = x;
}

int dequeue() {
    return queue[front++];
}

void bfs(int graph[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};
    int i, node;

    enqueue(start);
    visited[start] = 1;

    while (front != rear) {
        node = dequeue();
        printf("%d ", node);
        for (i = 0; i < n; i++) {
            if (graph[node][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(i);
            }
        }
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int n = 4;
    int graph[MAX][MAX] = {
        {0,1,1,0},
        {1,0,1,1},
        {1,1,0,0},
        {0,1,0,0}
    };
    bfs(graph, n, 0);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o bfs bfs.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\bfs
CH.SC.U4CSE24121
0 1 2 3
```

Time Complexity:

• Time Complexity: O(V^2)

Justification:

The graph is represented using an adjacency matrix.

- Each vertex is visited once.
- For every vertex, all V vertices are checked.
  Hence, time complexity is O(V^2).

Space Complexity:

• Space Complexity: O(V^2)

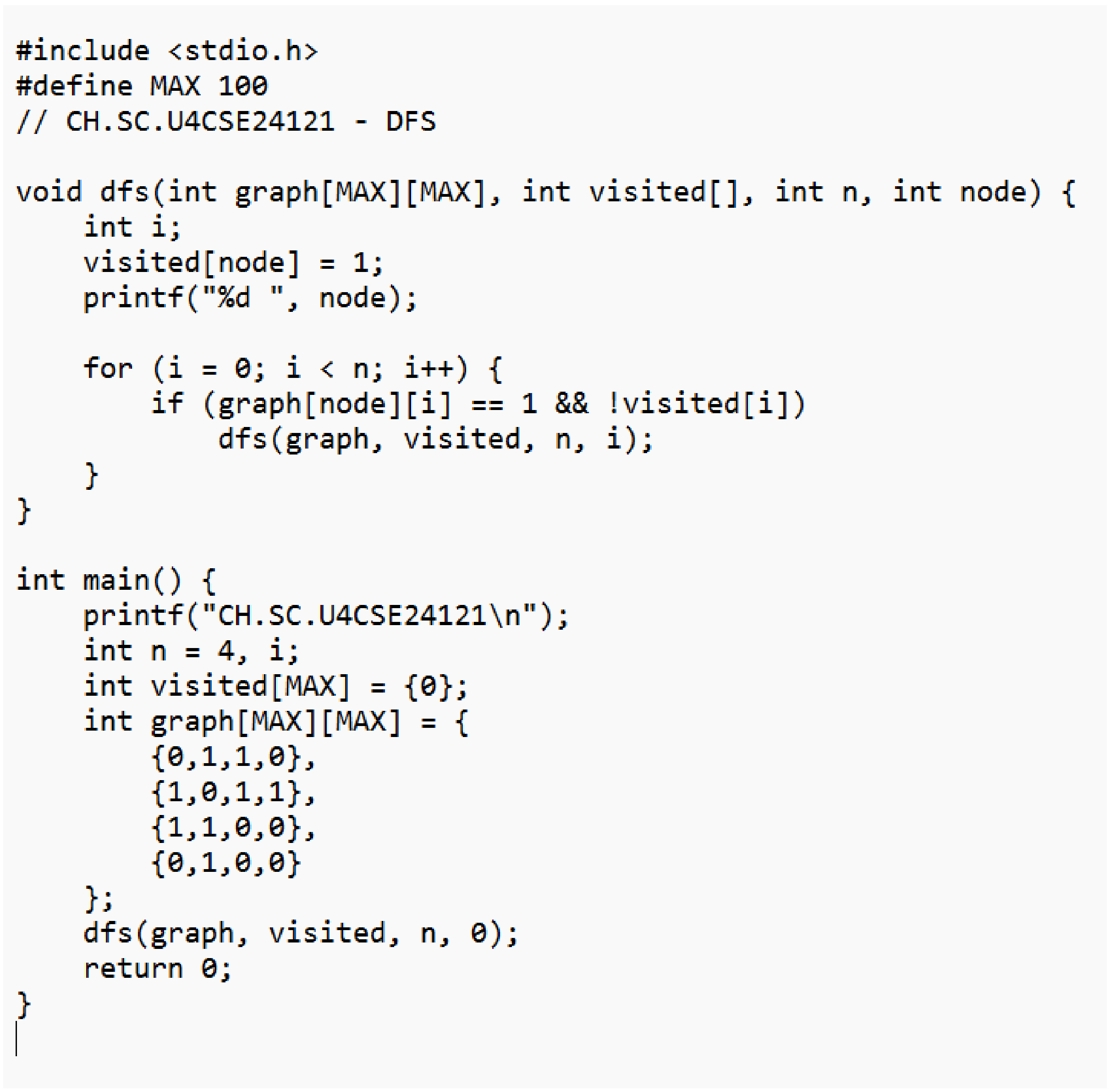
Justification:

The program uses:

- Adjacency matrix of size V × V
- Queue and visited array
  Therefore, space complexity is O(V^2).

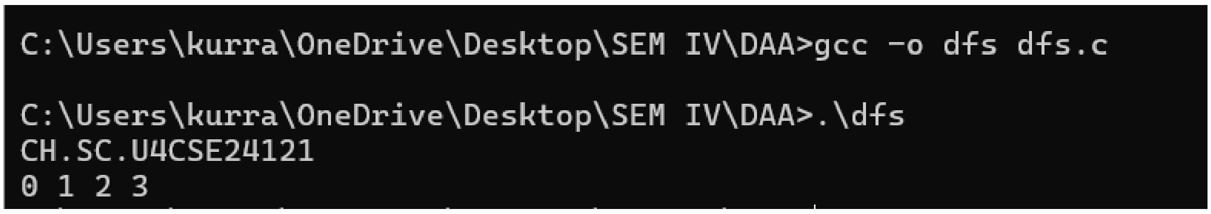
8.DFS

Code:



```c
#include <stdio.h>
#define MAX 100
// CH.SC.U4CSE24121 - DFS

void dfs(int graph[MAX][MAX], int visited[], int n, int node) {
    int i;
    visited[node] = 1;
    printf("%d ", node);

    for (i = 0; i < n; i++) {
        if (graph[node][i] == 1 && !visited[i])
            dfs(graph, visited, n, i);
    }
}

int main() {
    printf("CH.SC.U4CSE24121\n");
    int n = 4, i;
    int visited[MAX] = {0};
    int graph[MAX][MAX] = {
        {0,1,1,0},
        {1,0,1,1},
        {1,1,0,0},
        {0,1,0,0}
    };
    dfs(graph, visited, n, 0);
    return 0;
}
```

Output:



```
C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>gcc -o dfs dfs.c

C:\Users\kurra\OneDrive\Desktop\SEM IV\DAA>.\dfs
CH.SC.U4CSE24121
0 1 2 3
```

Time Complexity:

• Time Complexity: O(V^2)

Justification:

DFS uses an adjacency matrix.

- For each vertex, all other vertices are checked.
  Hence, time complexity is O(V^2).

Space Complexity:

• Space Complexity: O(V^2)

Justification:

The program uses:

- Adjacency matrix
- Visited array
- Recursion stack
  Thus, space complexity is O(V^2).