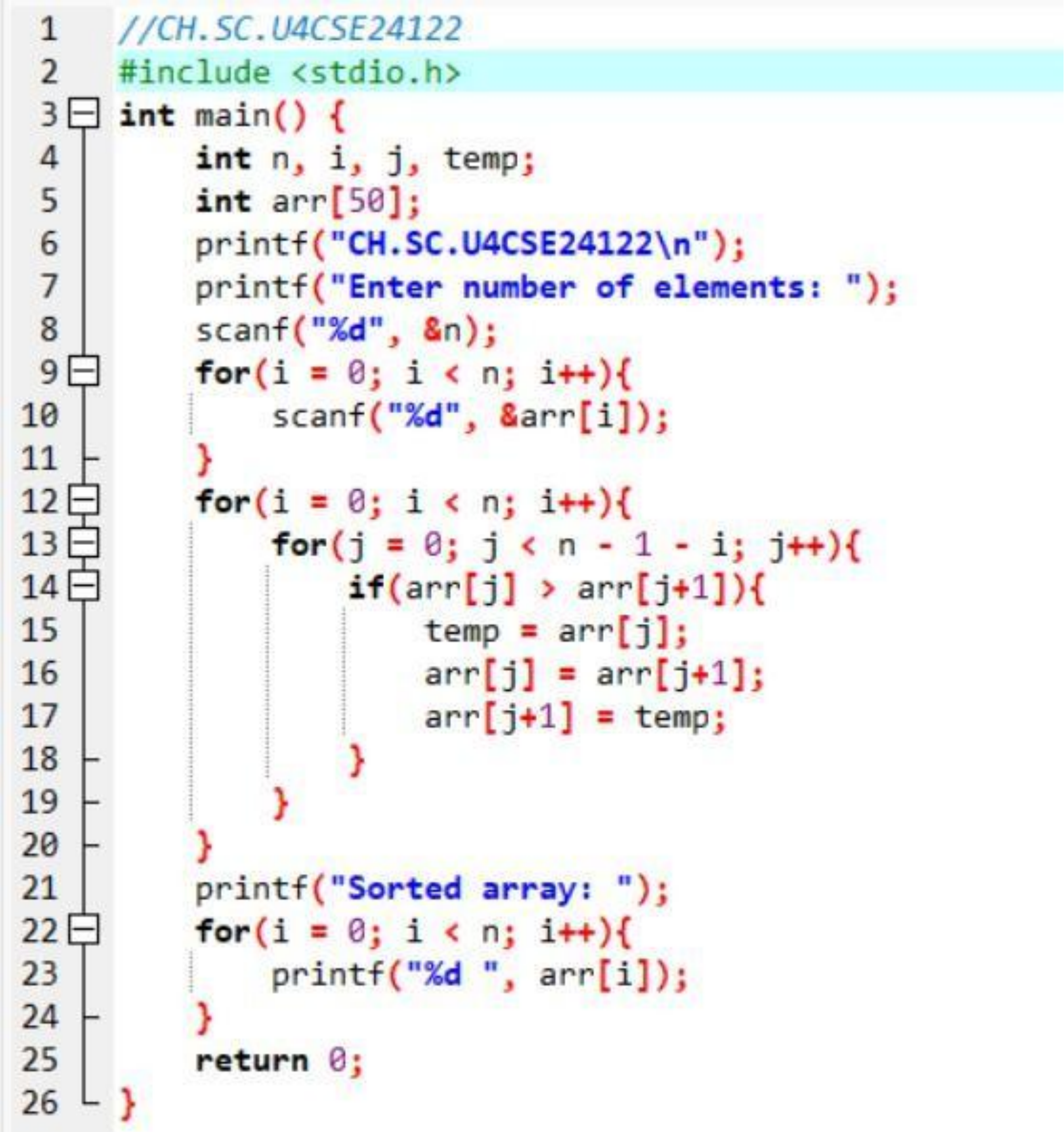
LEENA K CH.SC.U4CSE24122 CSE-B DAA LAB WORK

DESIGN ANALYSIS AND ALGORITHM-LAB WORK

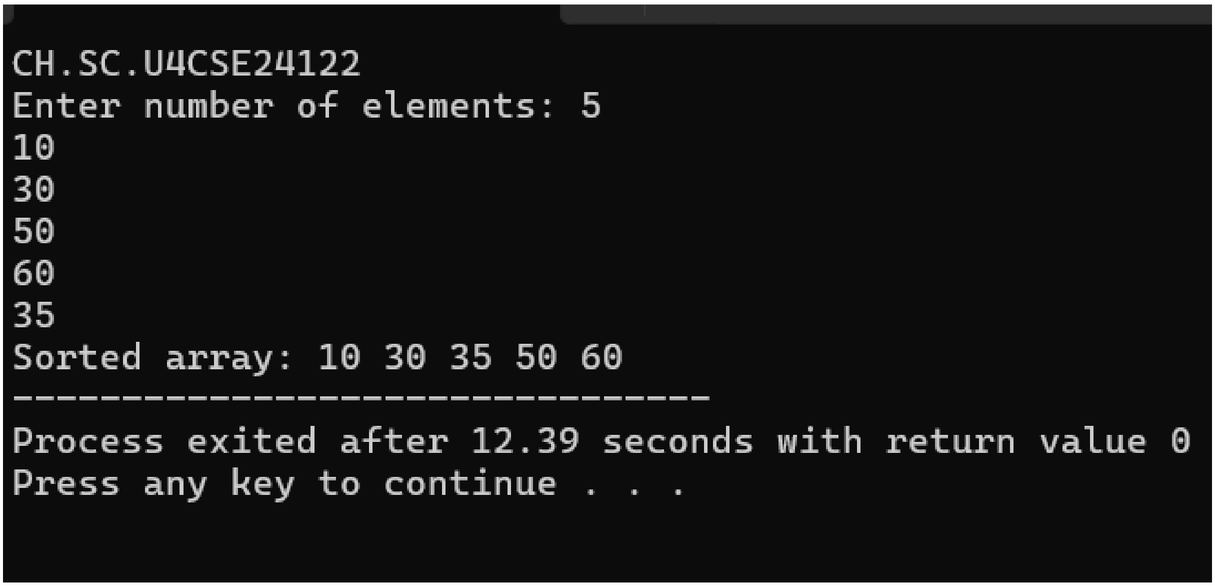
1)Write a program for Bubble Sorting

Code:



```c
//CH.SC.U4CSE24122
#include <stdio.h>
int main() {
    int n, i, j, temp;
    int arr[50];
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < n; i++){
        for(j = 0; j < n - 1 - i; j++){
            if(arr[j] > arr[j+1]){
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:



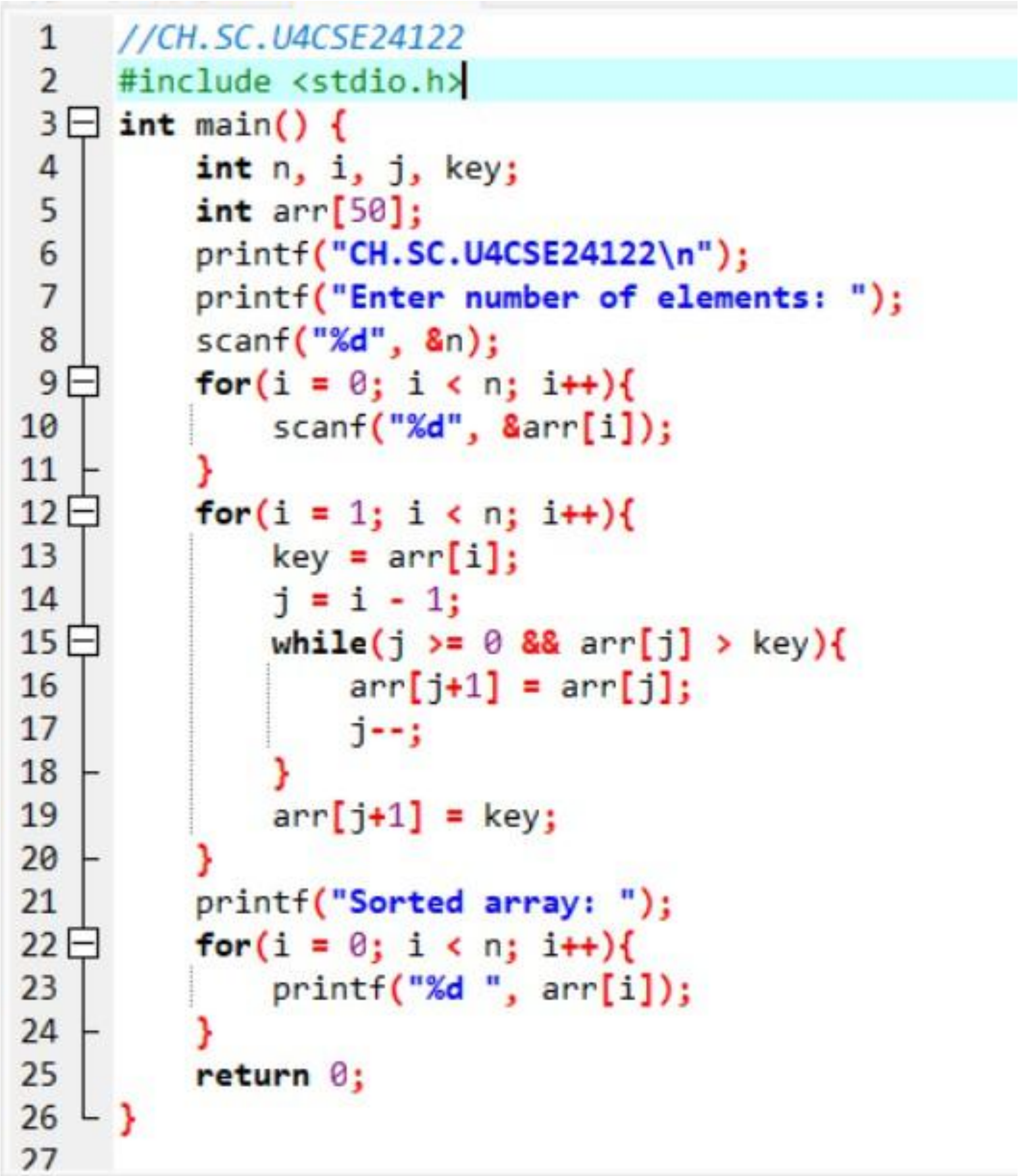
```
CH.SC.U4CSE24122
Enter number of elements: 5
10
30
50
60
35
Sorted array: 10 30 35 50 60
--------------------------------
Process exited after 12.39 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(n^2)
Space Complexity
O(1)

Time Complexity: O(n²) – This is because the algorithm uses nested loops to compare and swap adjacent elements repeatedly, going through the list multiple times until sorted.
Space Complexity: O(1) – It sorts in place and only uses a constant amount of extra memory for temporary variables.

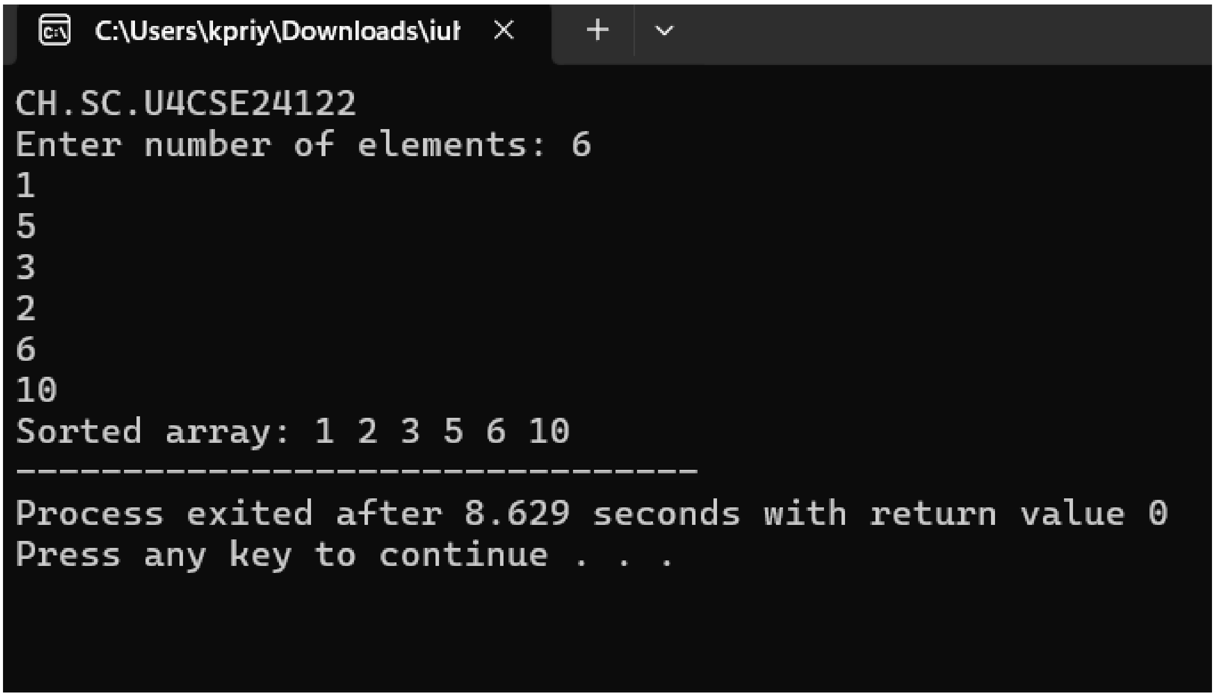
2)Write a program for Insertion Sorting

Code:



```c
//CH.SC.U4CSE24122
#include <stdio.h>
int main() {
    int n, i, j, key;
    int arr[50];
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    for(i = 1; i < n; i++){
        key = arr[i];
        j = i - 1;
        while(j >= 0 && arr[j] > key){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
    printf("Sorted array: ");
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}

```

Output:



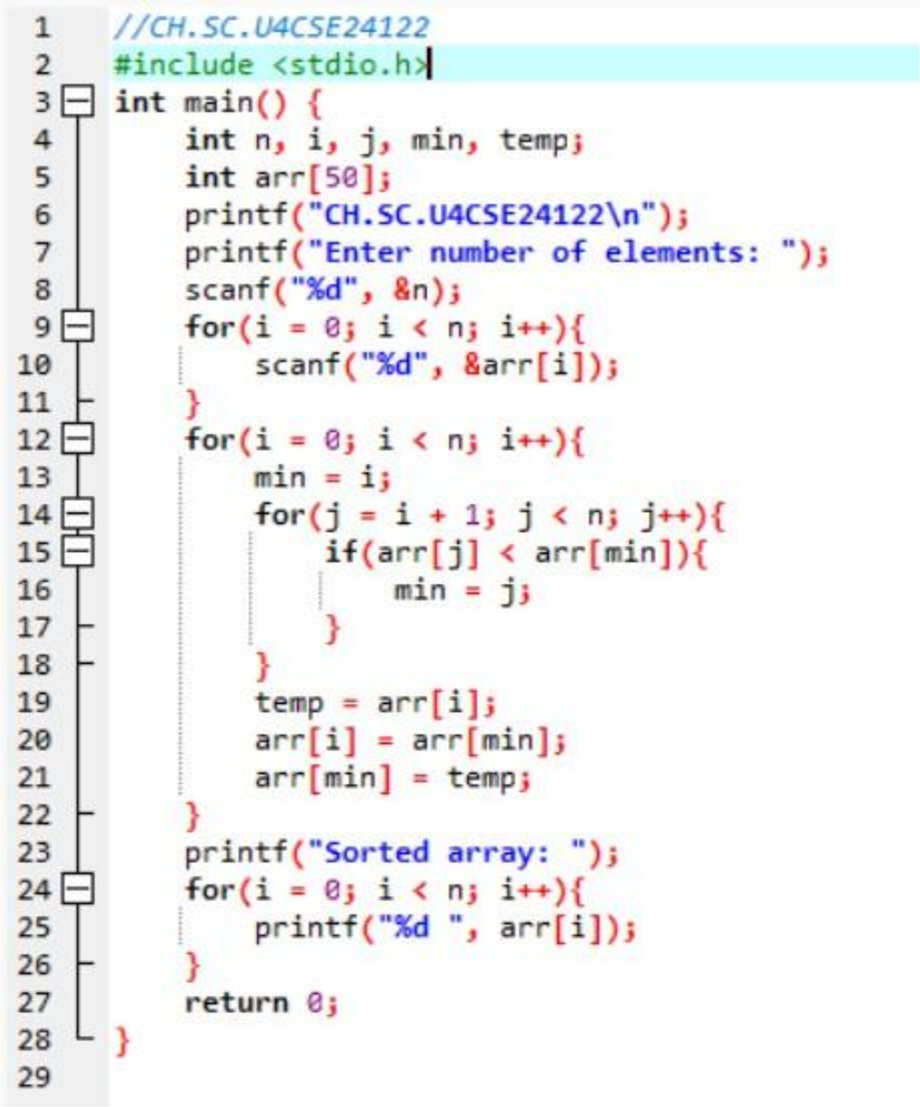
```
CH.SC.U4CSE24122
Enter number of elements: 6
1
5
3
2
6
10
Sorted array: 1 2 3 5 6 10
--------------------------------
Process exited after 8.629 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(n^2)
Space Complexity
O(1)

Time Complexity: O(n²) – In the worst case, each element may need to be compared and shifted with all previous elements, leading to quadratic time.
Space Complexity: O(1) – The algorithm operates in place without needing significant additional memory.

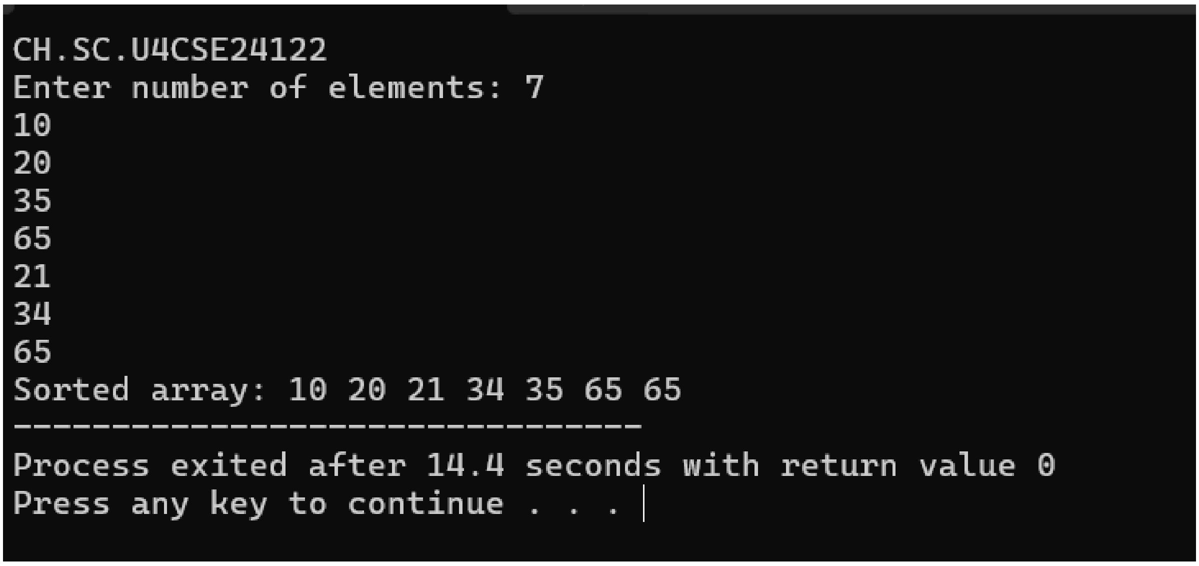
3) Write a program for Selection Sorting

Code:



```c
//CH.SC.U4CSE24122
#include <stdio.h>
int main() {
    int n, i, j, min, temp;
    int arr[50];
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < n; i++){
        min = i;
        for(j = i + 1; j < n; j++){
            if(arr[j] < arr[min]){
                min = j;
            }
        }
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
    printf("Sorted array: ");
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:



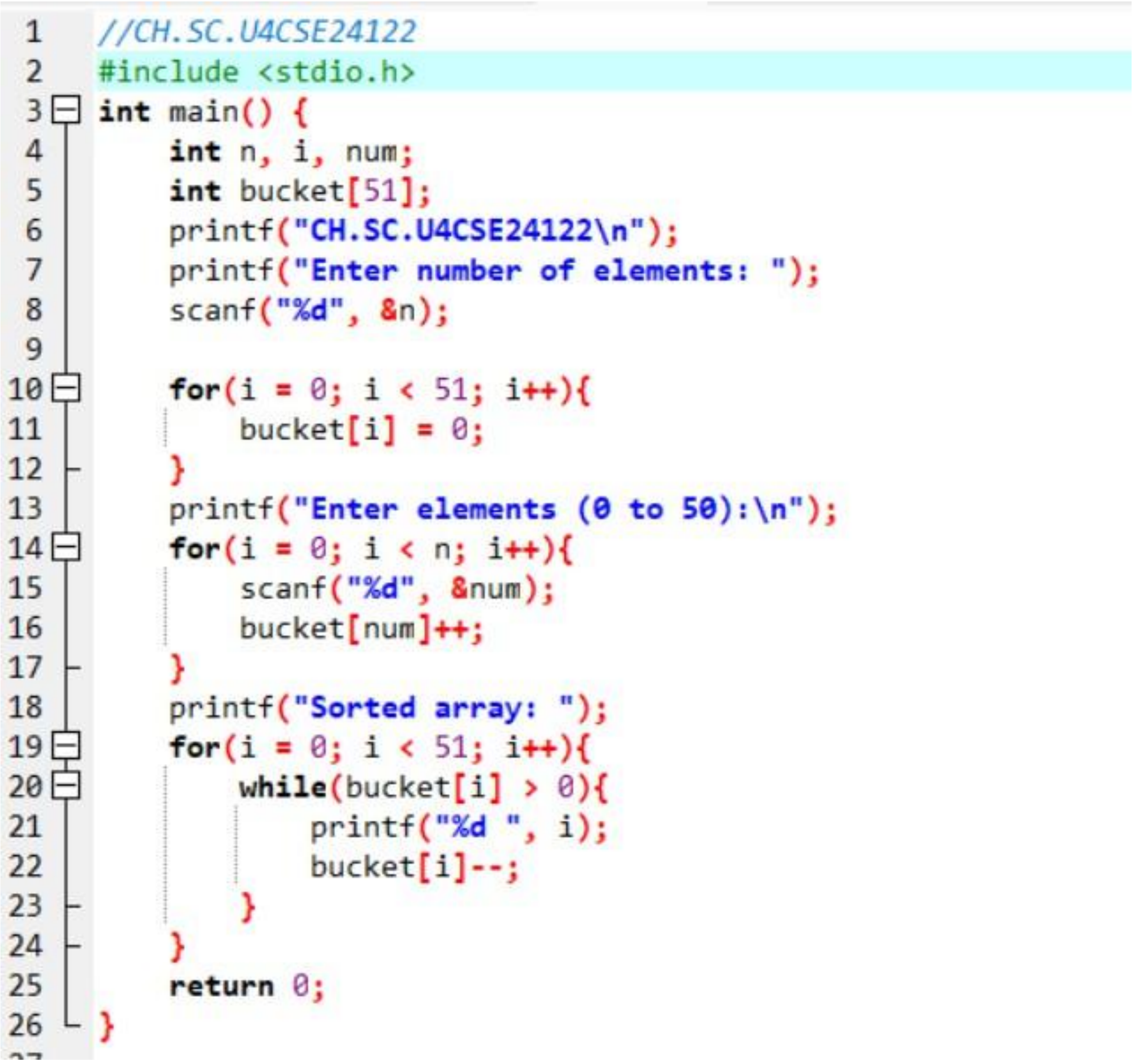
```
CH.SC.U4CSE24122
Enter number of elements: 7
10
20
35
65
21
34
65
Sorted array: 10 20 21 34 35 65 65
--------------------------------
Process exited after 14.4 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(n^2)
Space Complexity
O(1)

Time Complexity: $O(n^2)$ – It repeatedly finds the smallest (or largest) element from the unsorted portion and moves it to the sorted portion, requiring nested loops.
Space Complexity: O(1) – Like bubble and insertion sort, it uses constant extra space.

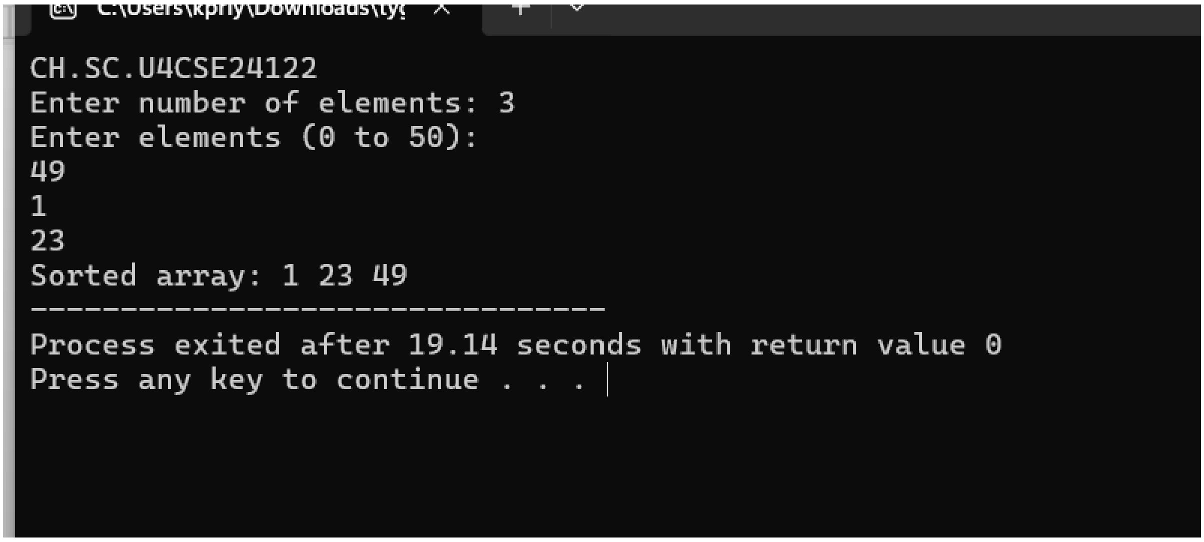
4) Write a program for Bucket Sorting

Code:



```c
//CH.SC.U4CSE24122
#include <stdio.h>
int main() {
    int n, i, num;
    int bucket[51];
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of elements: ");
    scanf("%d", &n);

    for(i = 0; i < 51; i++){
        bucket[i] = 0;
    }
    printf("Enter elements (0 to 50):\n");
    for(i = 0; i < n; i++){
        scanf("%d", &num);
        bucket[num]++;
    }
    printf("Sorted array: ");
    for(i = 0; i < 51; i++){
        while(bucket[i] > 0){
            printf("%d ", i);
            bucket[i]--;
        }
    }
    return 0;
}
```

Output:



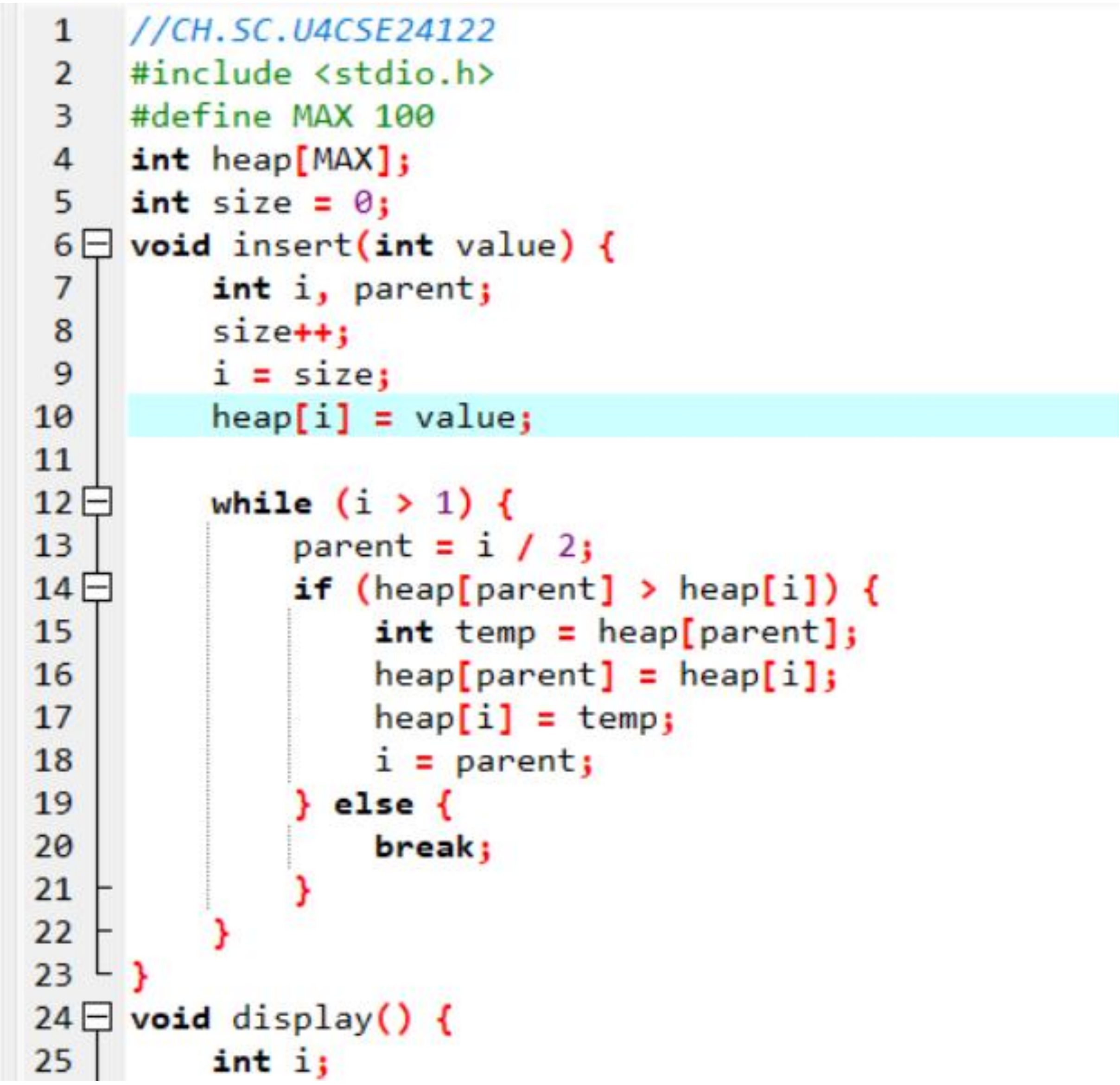
```
CH.SC.U4CSE24122
Enter number of elements: 3
Enter elements (0 to 50):
49
1
23
Sorted array: 1 23 49
--------------------------------
Process exited after 19.14 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(n^)
Space Complexity
O(1)

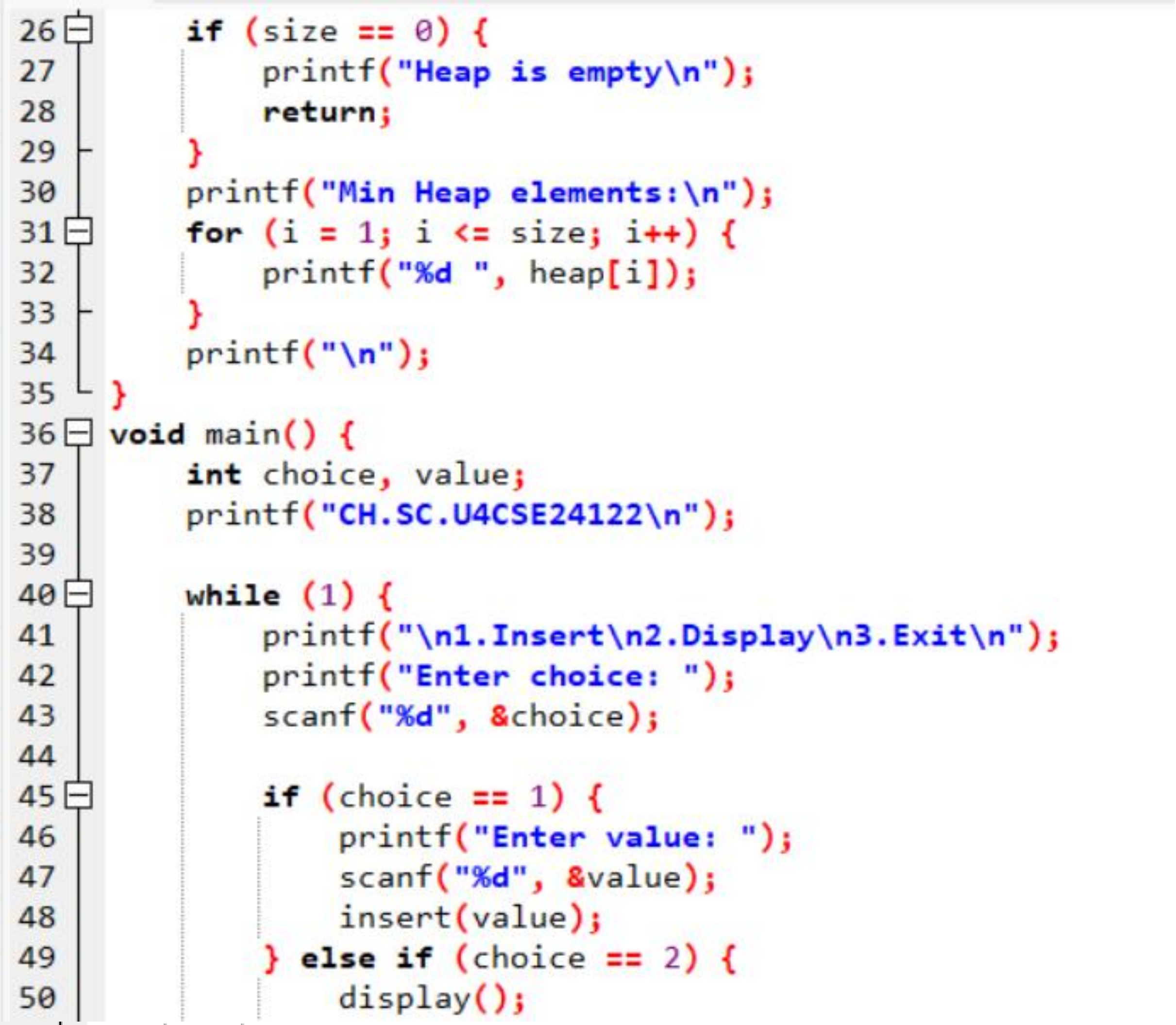
Time Complexity: $O(n^2)$ in worst case, but average is O(n+k) – It depends on distribution. Here, it's listed as $O(n^2)$, possibly due to uneven bucket distribution.
Space Complexity: O(1) – Usually O(n+k) for buckets, but if implemented in place or with fixed buckets, extra space can be minimal.

5)Write a program for Heap Sorting

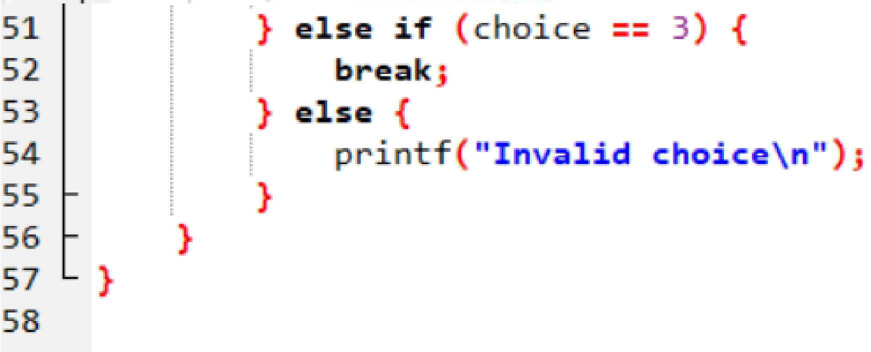
Min Heap Code:



```c
//CH.SC.U4CSE24122
#include <stdio.h>
#define MAX 100
int heap[MAX];
int size = 0;
void insert(int value) {
    int i, parent;
    size++;
    i = size;
    heap[i] = value;

    while (i > 1) {
        parent = i / 2;
        if (heap[parent] > heap[i]) {
            int temp = heap[parent];
            heap[parent] = heap[i];
            heap[i] = temp;
            i = parent;
        } else {
            break;
        }
    }
}
void display() {
    int i;
```

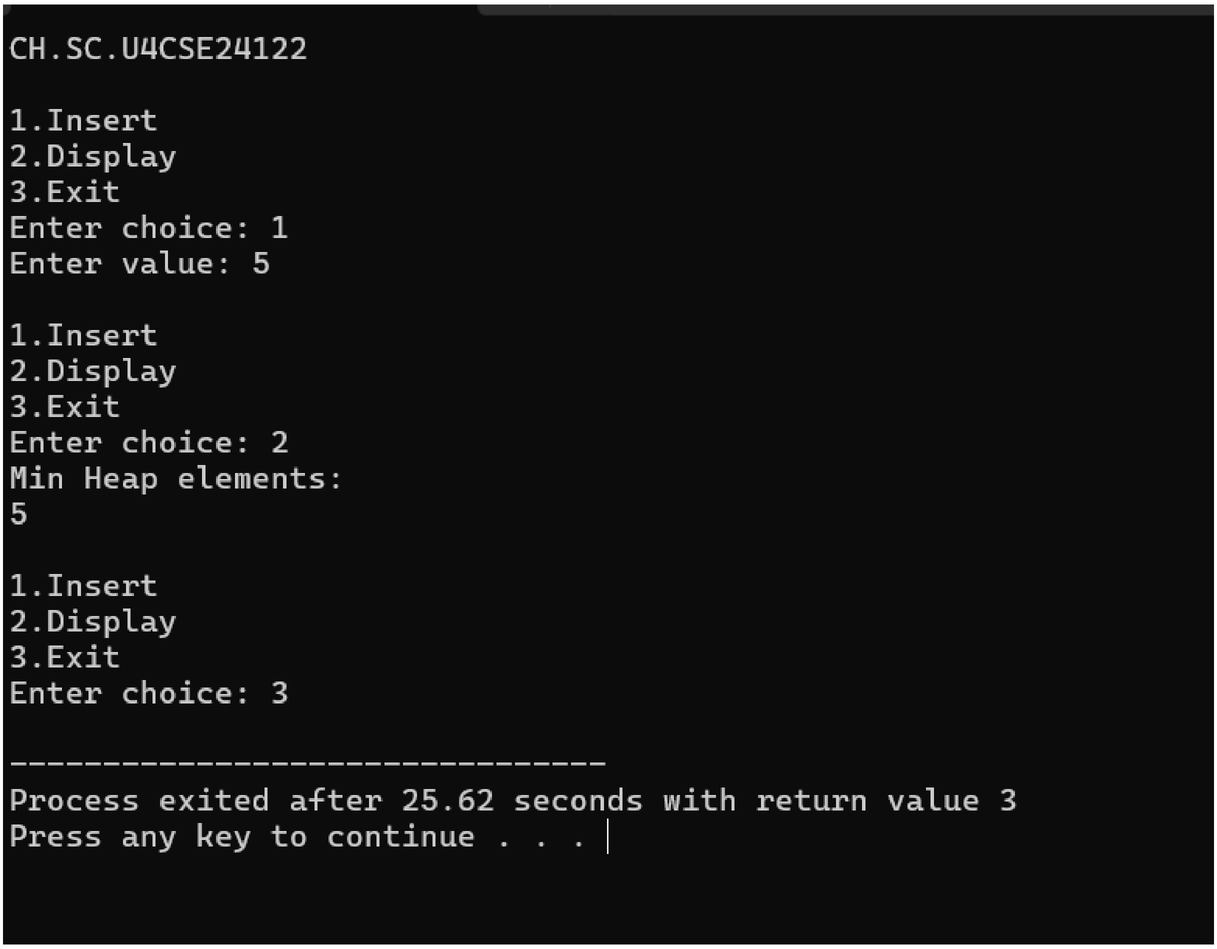


```c
    if (size == 0) {
        printf("Heap is empty\n");
        return;
    }
    printf("Min Heap elements:\n");
    for (i = 1; i <= size; i++) {
        printf("%d ", heap[i]);
    }
    printf("\n");
}
void main() {
    int choice, value;
    printf("CH.SC.U4CSE24122\n");

    while (1) {
        printf("\n1.Insert\n2.Display\n3.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter value: ");
            scanf("%d", &value);
            insert(value);
        } else if (choice == 2) {
            display();
```



```c
        } else if (choice == 3) {
            break;
        } else {
            printf("Invalid choice\n");
        }
    }
}

```

Output:

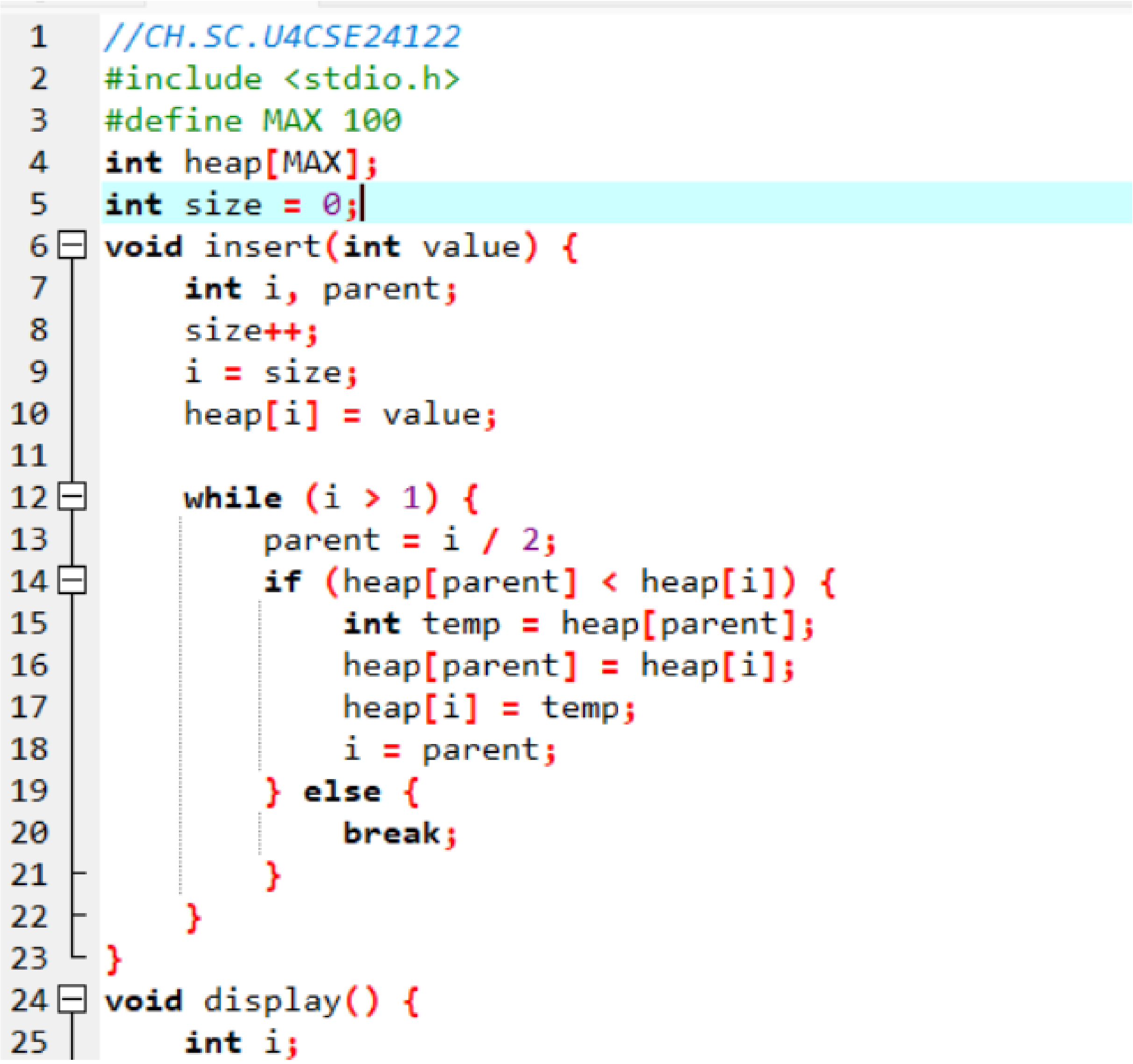


```
CH.SC.U4CSE24122

1.Insert
2.Display
3.Exit
Enter choice: 1
Enter value: 5

1.Insert
2.Display
3.Exit
Enter choice: 2
Min Heap elements:
5

1.Insert
2.Display
3.Exit
Enter choice: 3

--------------------------------
Process exited after 25.62 seconds with return value 3
Press any key to continue . . .
```

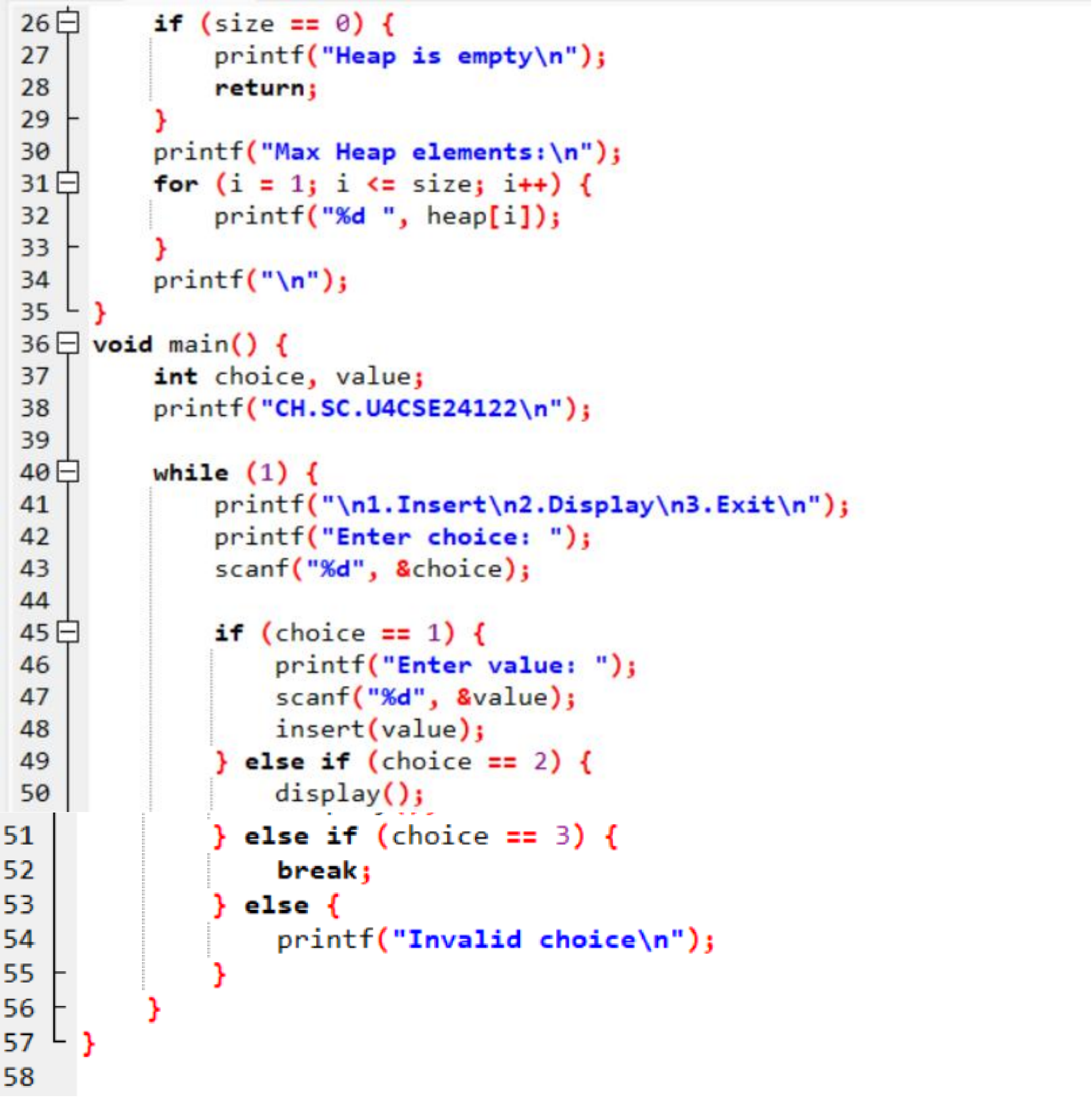
Time Complexity:
O(logn)
Space Complexity
O(n)

Time Complexity: O(n log n) – Each heap insertion or removal takes O(log n), repeated for n elements.
Space Complexity: O(n) – Additional space is used to store the heap structure, especially if implemented with an array.

Max Heap Code:

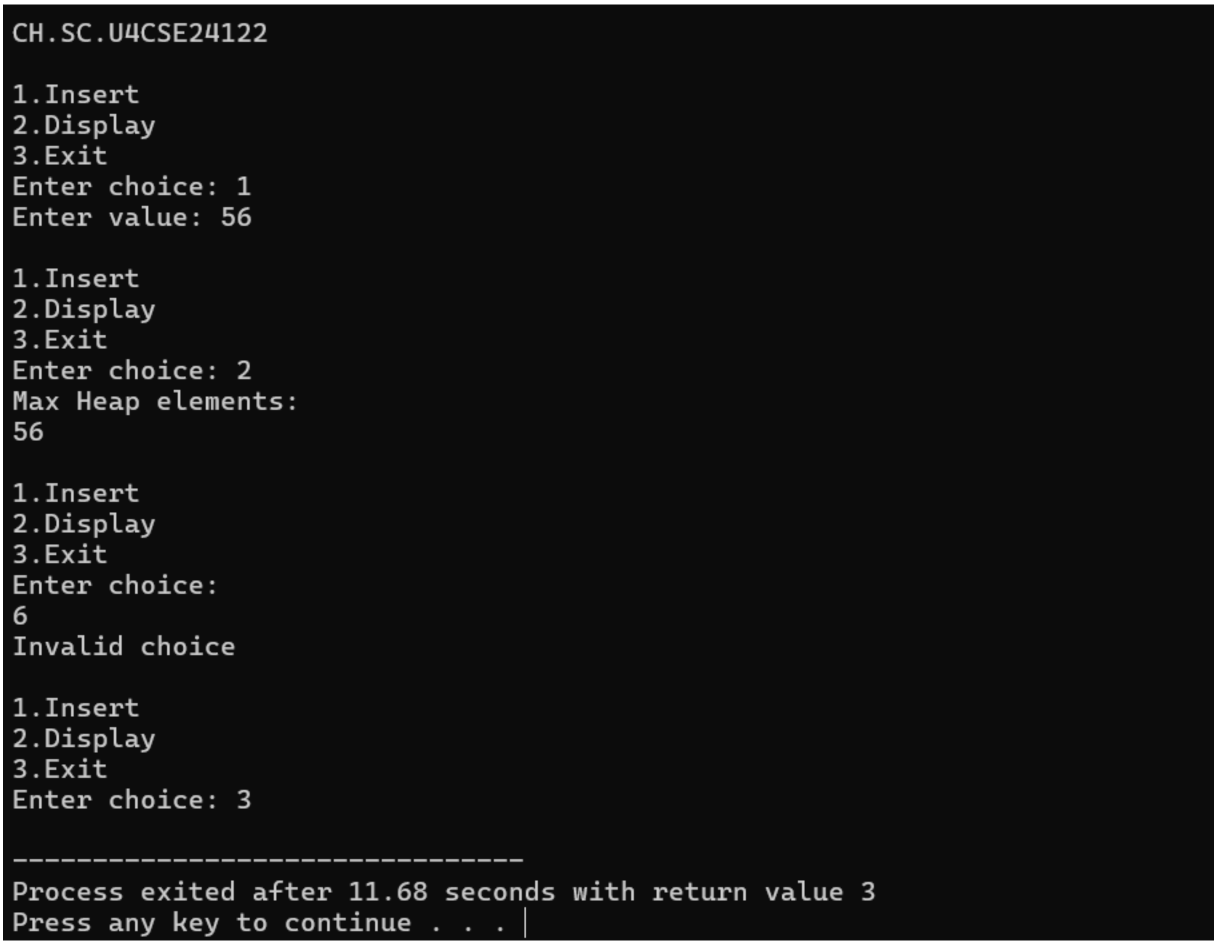


```c
//CH.SC.U4CSE24122
#include <stdio.h>
#define MAX 100
int heap[MAX];
int size = 0;
void insert(int value) {
    int i, parent;
    size++;
    i = size;
    heap[i] = value;

    while (i > 1) {
        parent = i / 2;
        if (heap[parent] < heap[i]) {
            int temp = heap[parent];
            heap[parent] = heap[i];
            heap[i] = temp;
            i = parent;
        } else {
            break;
        }
    }
}
void display() {
    int i;
```



```c
    if (size == 0) {
        printf("Heap is empty\n");
        return;
    }
    printf("Max Heap elements:\n");
    for (i = 1; i <= size; i++) {
        printf("%d ", heap[i]);
    }
    printf("\n");
}
void main() {
    int choice, value;
    printf("CH.SC.U4CSE24122\n");

    while (1) {
        printf("\n1.Insert\n2.Display\n3.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter value: ");
            scanf("%d", &value);
            insert(value);
        } else if (choice == 2) {
            display();
        } else if (choice == 3) {
            break;
        } else {
            printf("Invalid choice\n");
        }
    }
}
```

Output:



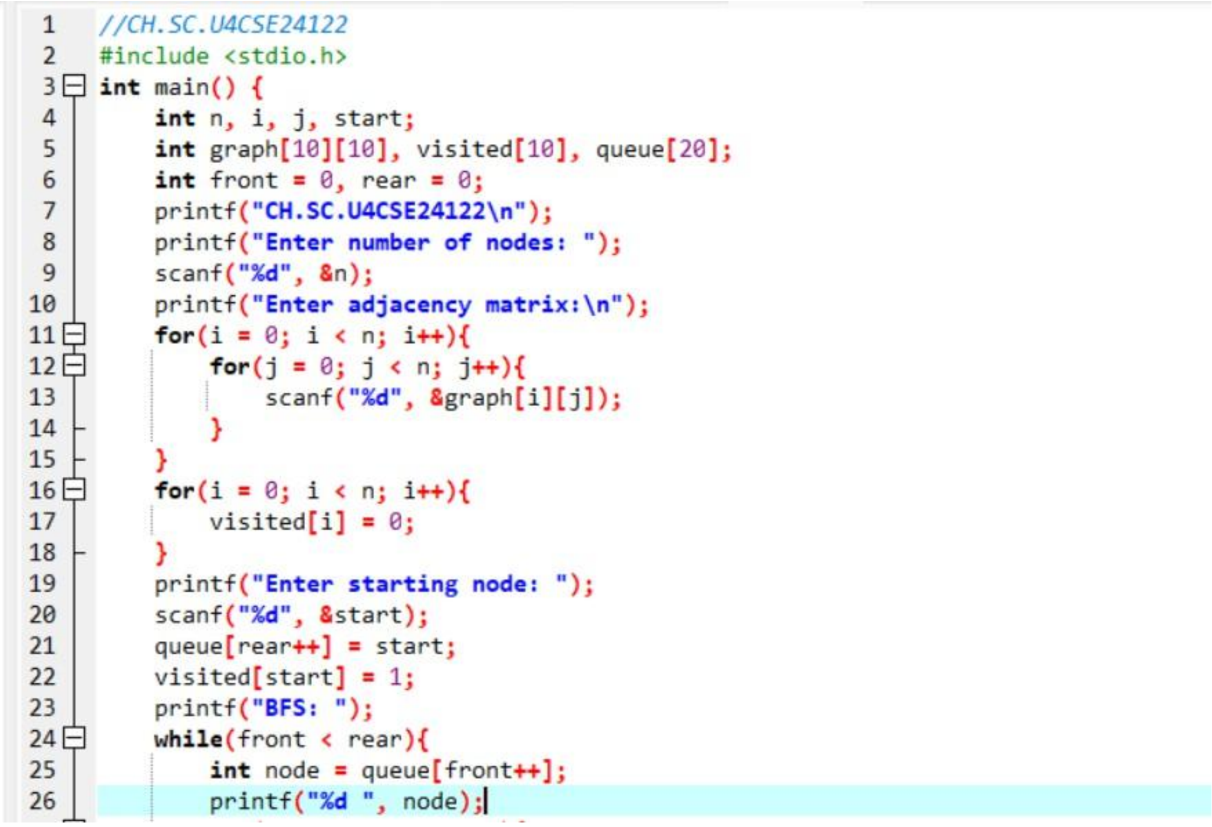
```
CH.SC.U4CSE24122

1.Insert
2.Display
3.Exit
Enter choice: 1
Enter value: 56

1.Insert
2.Display
3.Exit
Enter choice: 2
Max Heap elements:
56

1.Insert
2.Display
3.Exit
Enter choice:
6
Invalid choice

1.Insert
2.Display
3.Exit
Enter choice: 3

--------------------------------
Process exited after 11.68 seconds with return value 3
Press any key to continue . . .
```

Time Complexity:
O(logn)
Space Complexity
O(n)

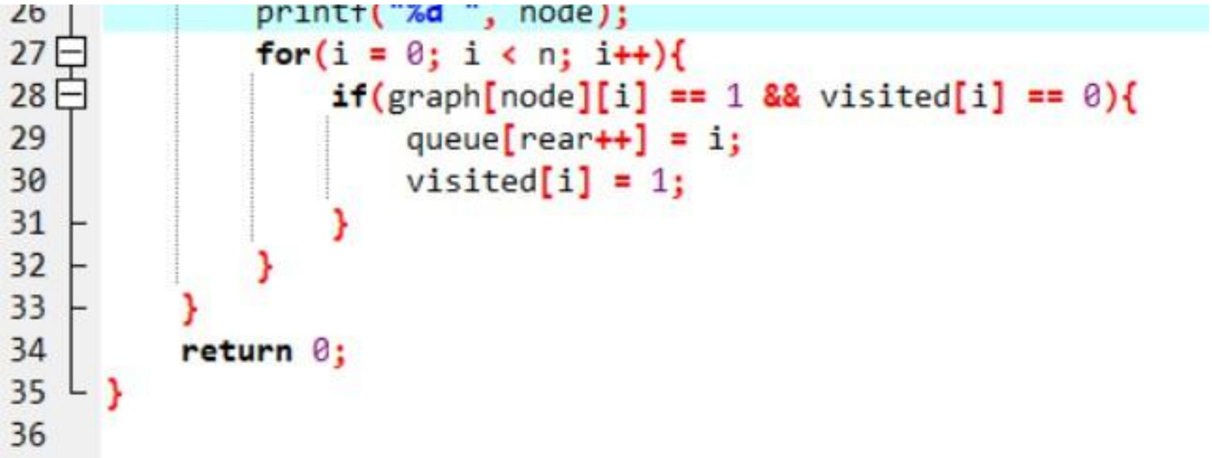
Time Complexity: O(n log n) – Each heap insertion or removal takes O(log n), repeated for n elements.
Space Complexity: O(n) – Additional space is used to store the heap structure, especially if implemented with an array.

6) Write a program for BFS

Code:

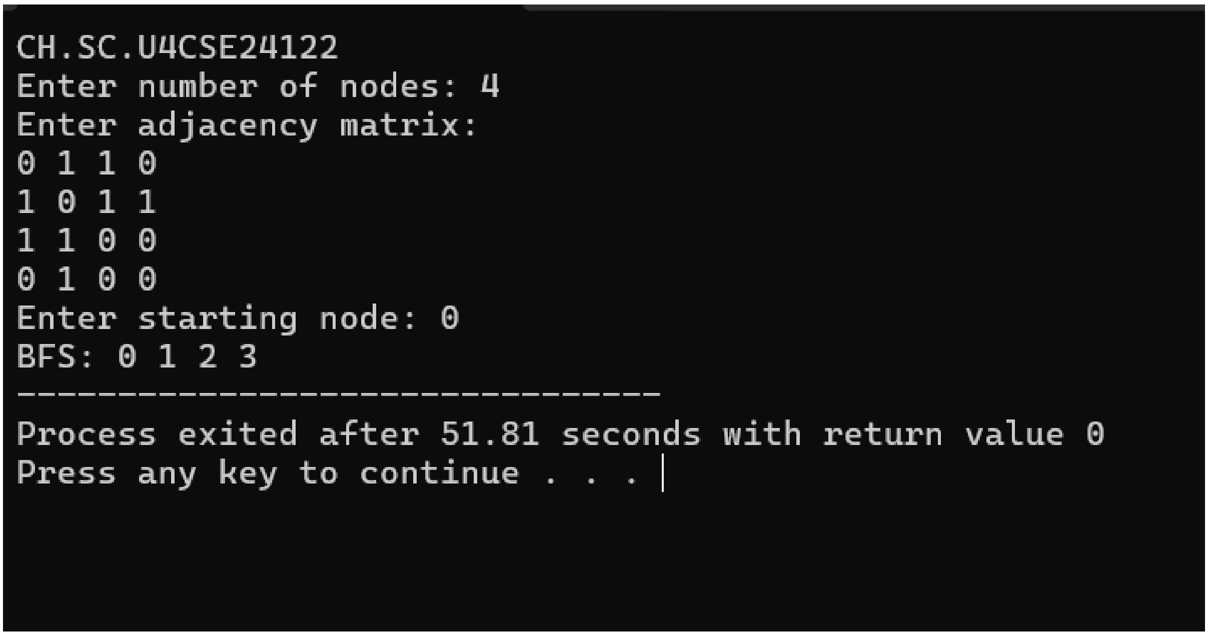


```c
//CH.SC.U4CSE24122
#include <stdio.h>
int main() {
    int n, i, j, start;
    int graph[10][10], visited[10], queue[20];
    int front = 0, rear = 0;
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    for(i = 0; i < n; i++){
        visited[i] = 0;
    }
    printf("Enter starting node: ");
    scanf("%d", &start);
    queue[rear++] = start;
    visited[start] = 1;
    printf("BFS: ");
    while(front < rear){
        int node = queue[front++];
        printf("%d ", node);
```



```c
        for(i = 0; i < n; i++){
            if(graph[node][i] == 1 && visited[i] == 0){
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
    return 0;
}
```

Output:



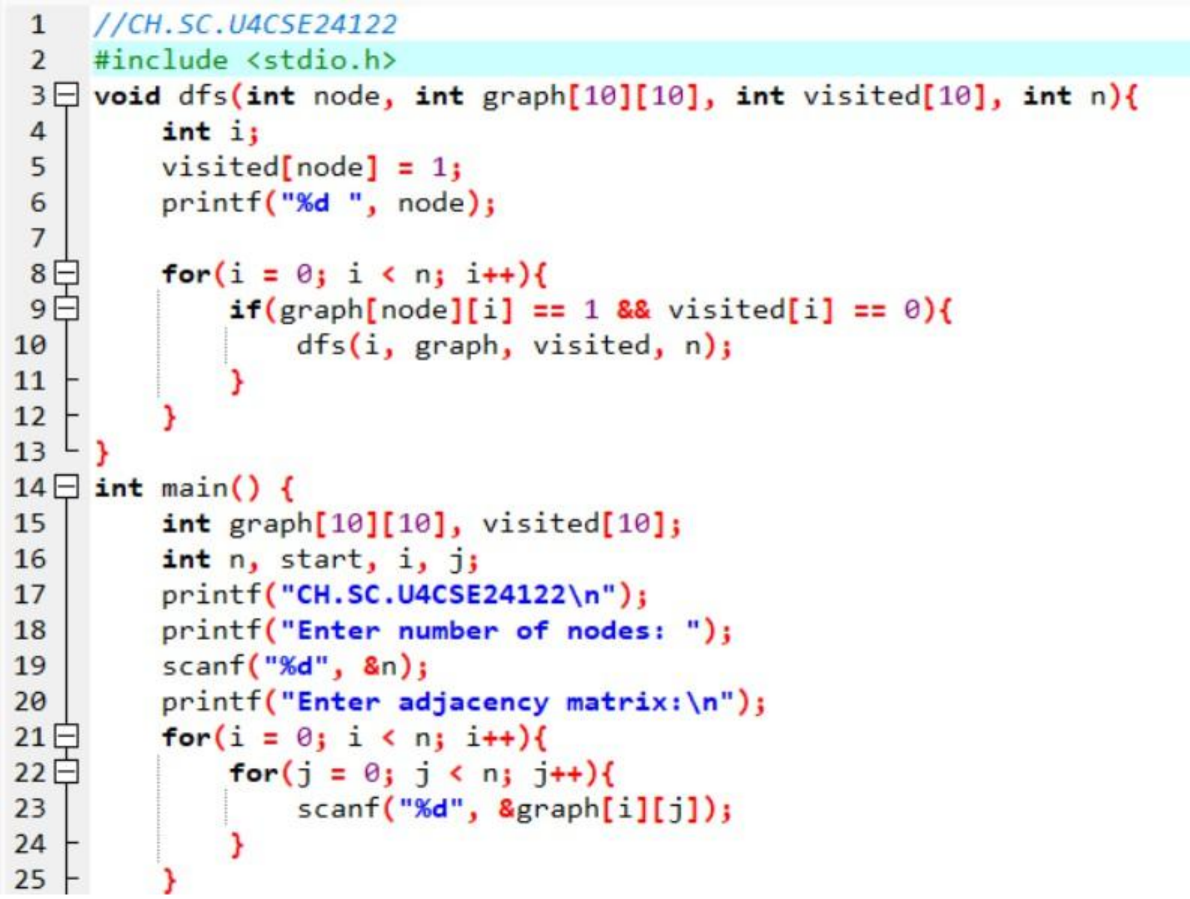
```
CH.SC.U4CSE24122
Enter number of nodes: 4
Enter adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
Enter starting node: 0
BFS: 0 1 2 3
--------------------------------
Process exited after 51.81 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(v^2)
Space Complexity
O(v^2)

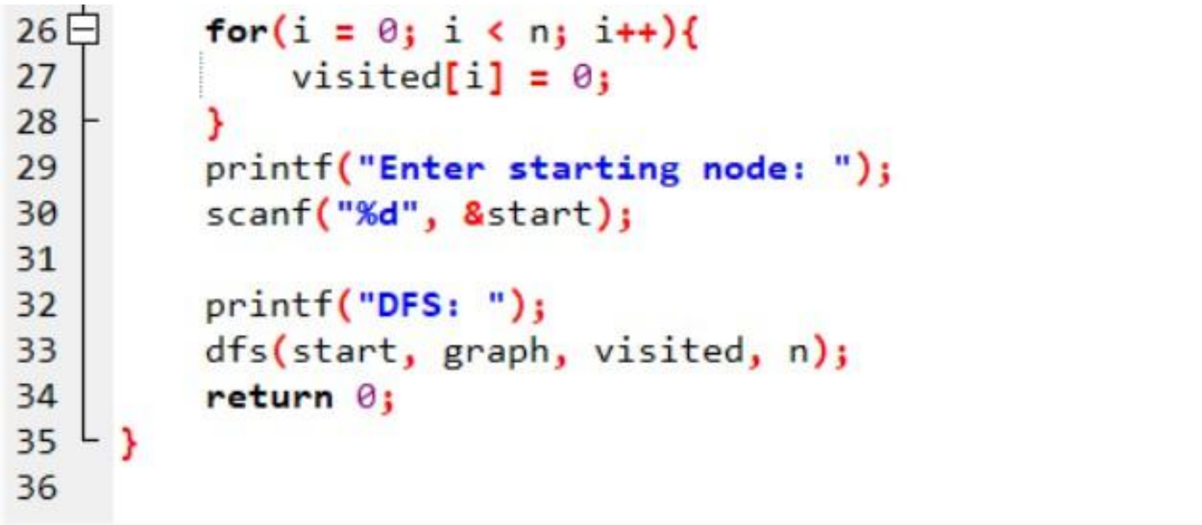
Time Complexity: O(v²) – In adjacency matrix representation, each node checks all others, leading to v² operations.
Space Complexity: O(v²) – It stores the adjacency matrix and a queue that can hold up to all vertices.

7)Write a program for DFS

Code:

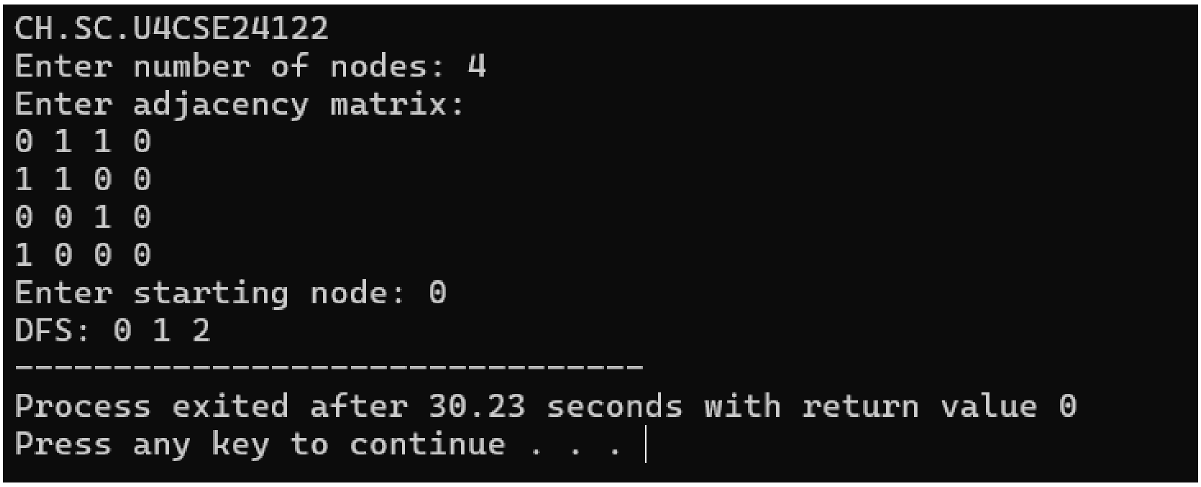


```c
//CH.SC.U4CSE24122
#include <stdio.h>
void dfs(int node, int graph[10][10], int visited[10], int n){
    int i;
    visited[node] = 1;
    printf("%d ", node);

    for(i = 0; i < n; i++){
        if(graph[node][i] == 1 && visited[i] == 0){
            dfs(i, graph, visited, n);
        }
    }
}
int main() {
    int graph[10][10], visited[10];
    int n, start, i, j;
    printf("CH.SC.U4CSE24122\n");
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
```



```c
    for(i = 0; i < n; i++){
        visited[i] = 0;
    }
    printf("Enter starting node: ");
    scanf("%d", &start);

    printf("DFS: ");
    dfs(start, graph, visited, n);
    return 0;
}

```

Output:



```
CH.SC.U4CSE24122
Enter number of nodes: 4
Enter adjacency matrix:
0 1 1 0
1 1 0 0
0 0 1 0
1 0 0 0
Enter starting node: 0
DFS: 0 1 2
--------------------------------
Process exited after 30.23 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:
O(v^2)
Space Complexity
O(v^2)

Time Complexity: O(v^2) – Similarly, with adjacency matrix, each vertex may explore all others.
Space Complexity: O(v^2) – Due to matrix storage and recursion stack that can go as deep as the number of vertices.