# Data Analytics Algorithm

Task -2

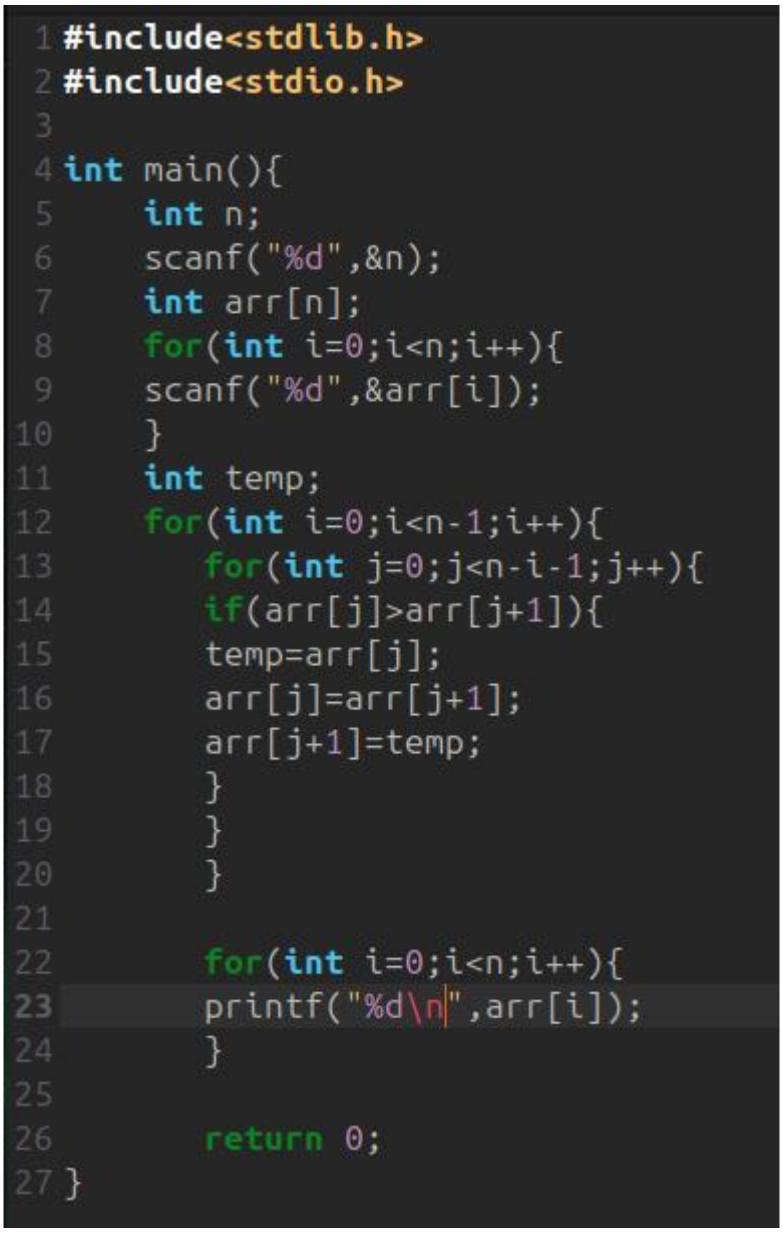
Dhusyanth R

CH.SC.U4CSE24161

CSE 2nd YEAR

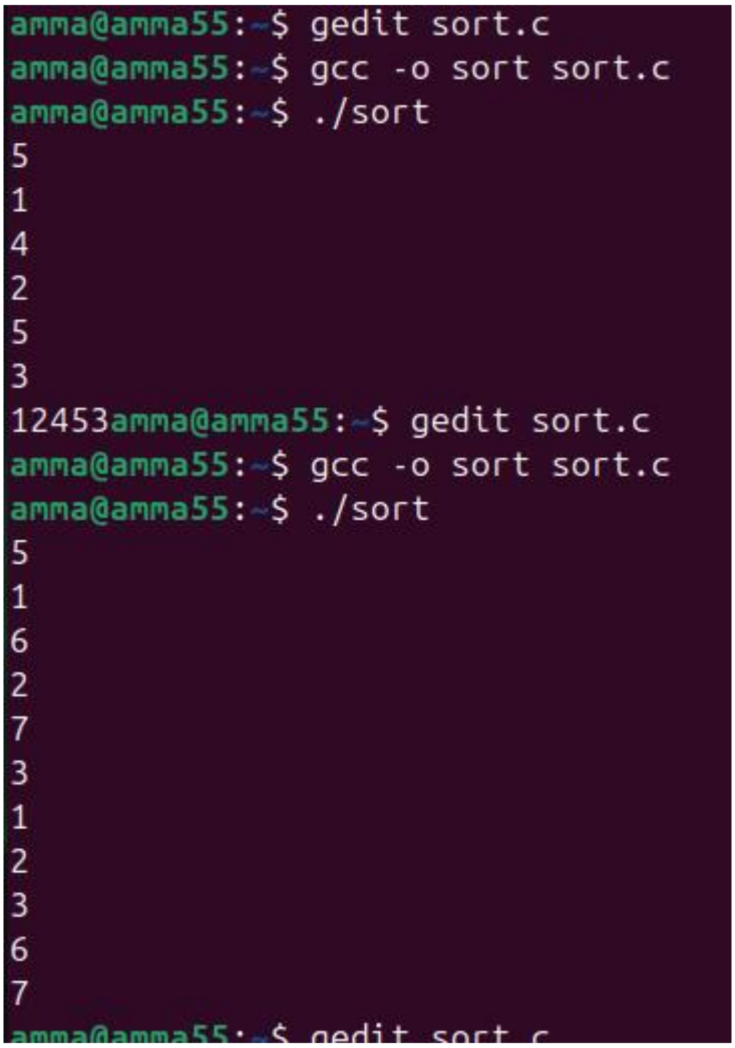
## 1.)Bubble Sort:

Code:



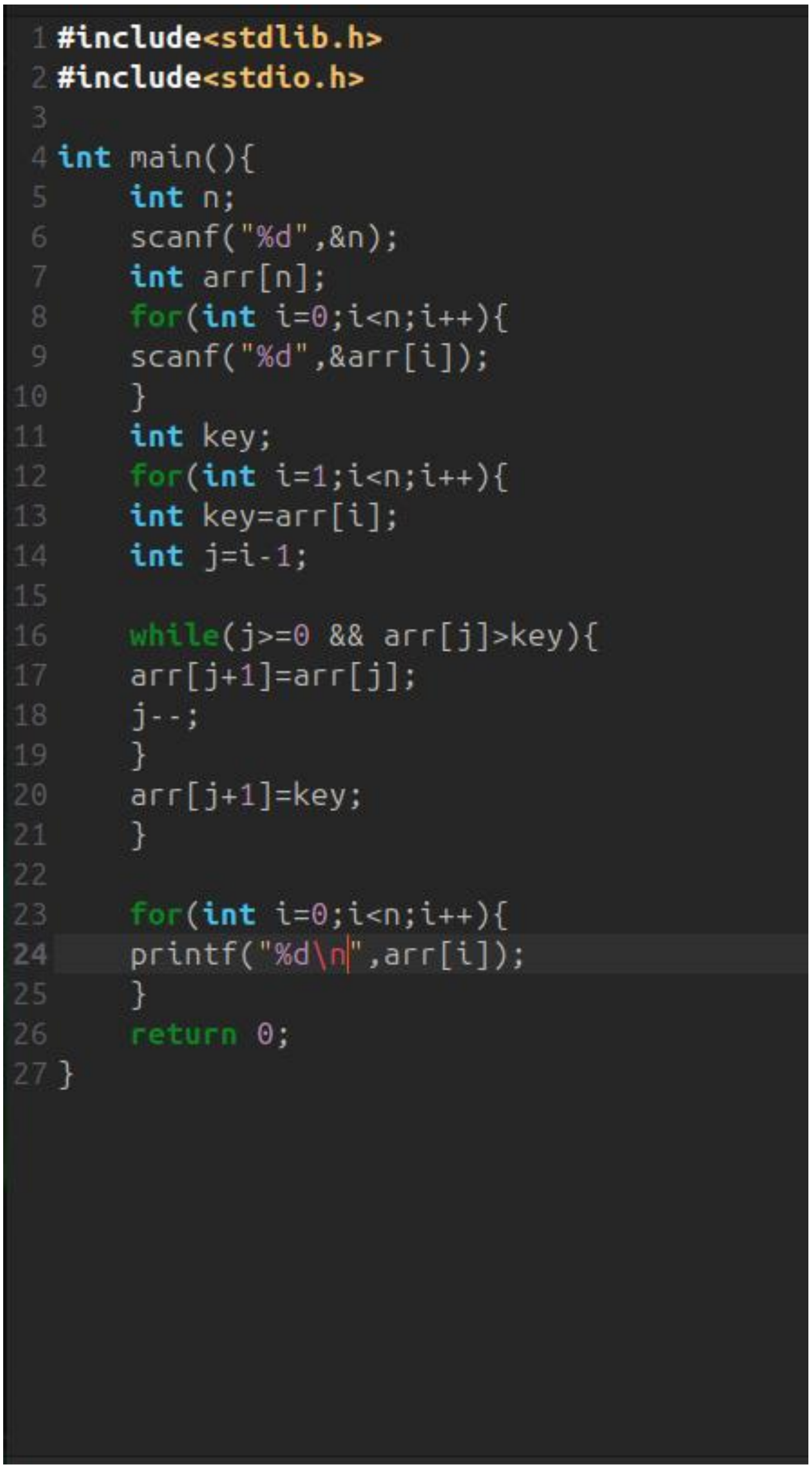
```c
#include<stdlib.h>
#include<stdio.h>

int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
    }
    int temp;
    for(int i=0;i<n-1;i++){
       for(int j=0;j<n-i-1;j++){
       if(arr[j]>arr[j+1]){
       temp=arr[j];
       arr[j]=arr[j+1];
       arr[j+1]=temp;
       }
       }
       }

       for(int i=0;i<n;i++){
       printf("%d\n",arr[i]);
       }

       return 0;
}
```

Output:



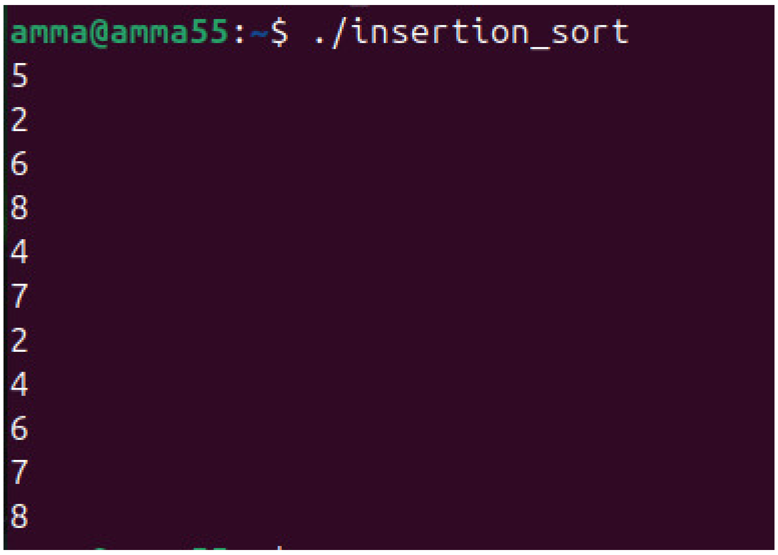
```
amma@amma55:~$ gedit sort.c
amma@amma55:~$ gcc -o sort sort.c
amma@amma55:~$ ./sort
5
1
4
2
5
3
12453amma@amma55:~$ gedit sort.c
amma@amma55:~$ gcc -o sort sort.c
amma@amma55:~$ ./sort
5
1
6
2
7
3
1
2
3
6
7
amma@amma55:~$ gedit sort.c
```

## 2.)Insertion Sort:



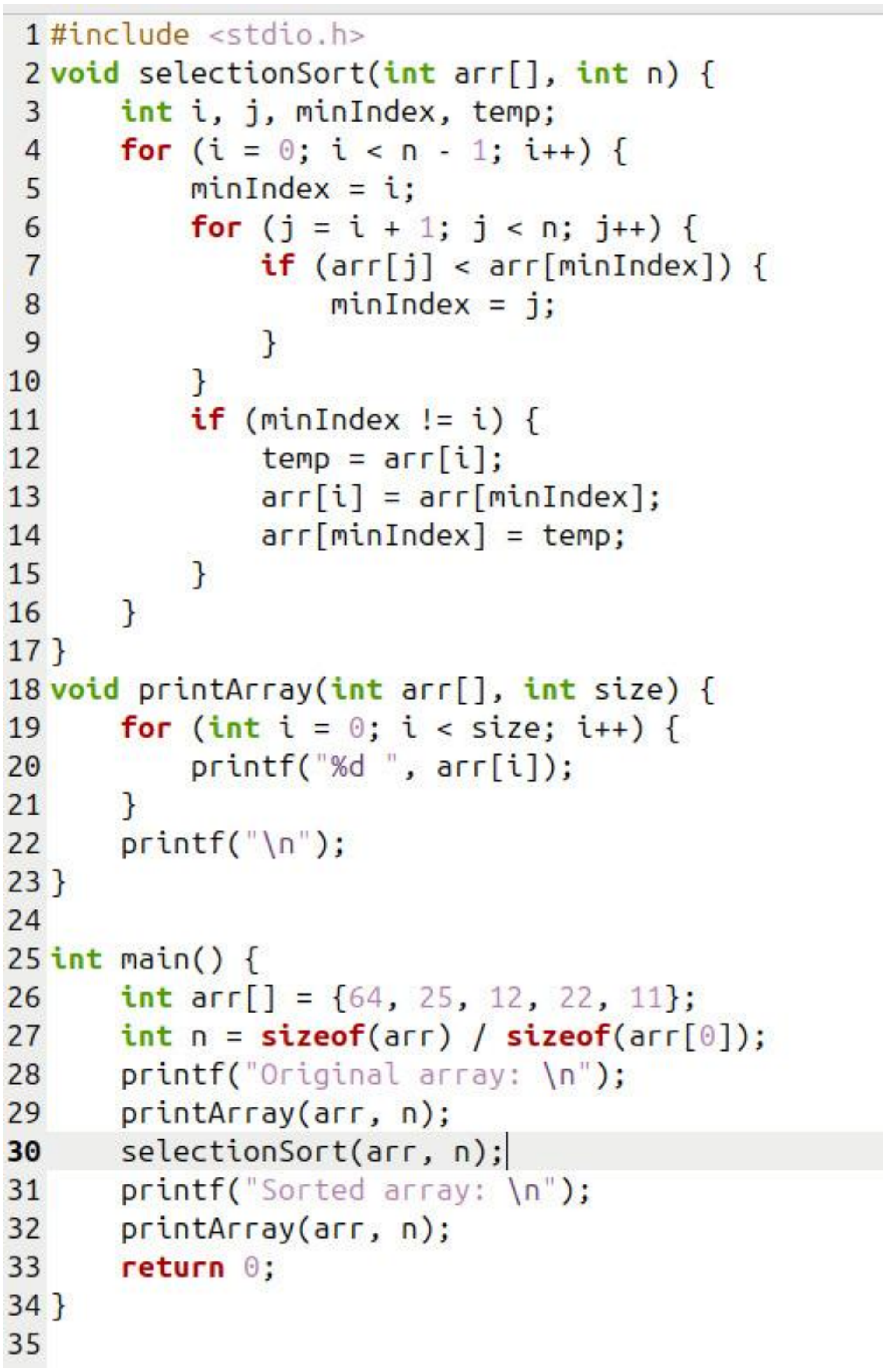
```c
#include<stdlib.h>
#include<stdio.h>

int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
    }
    int key;
    for(int i=1;i<n;i++){
    int key=arr[i];
    int j=i-1;

    while(j>=0 && arr[j]>key){
    arr[j+1]=arr[j];
    j--;
    }
    arr[j+1]=key;
    }

    for(int i=0;i<n;i++){
    printf("%d\n",arr[i]);
    }
    return 0;
}
```

Output:



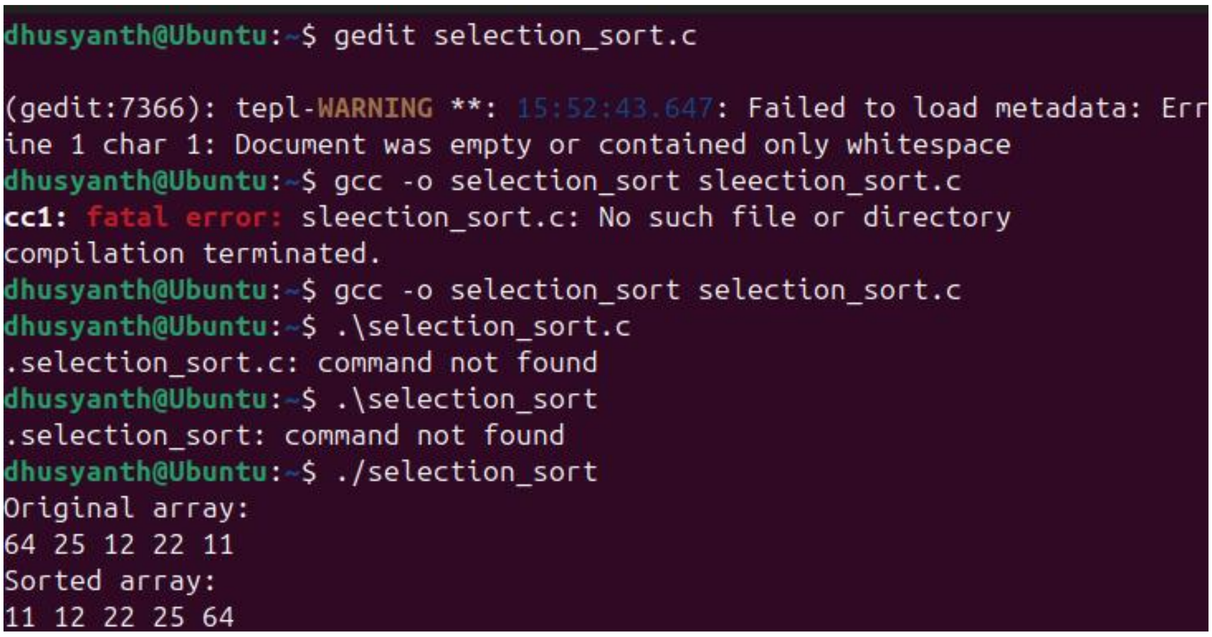
```
amma@amma55:~$ ./insertion_sort
5
2
6
8
4
7
2
4
6
7
8
```

## 3.)Selection Sort:



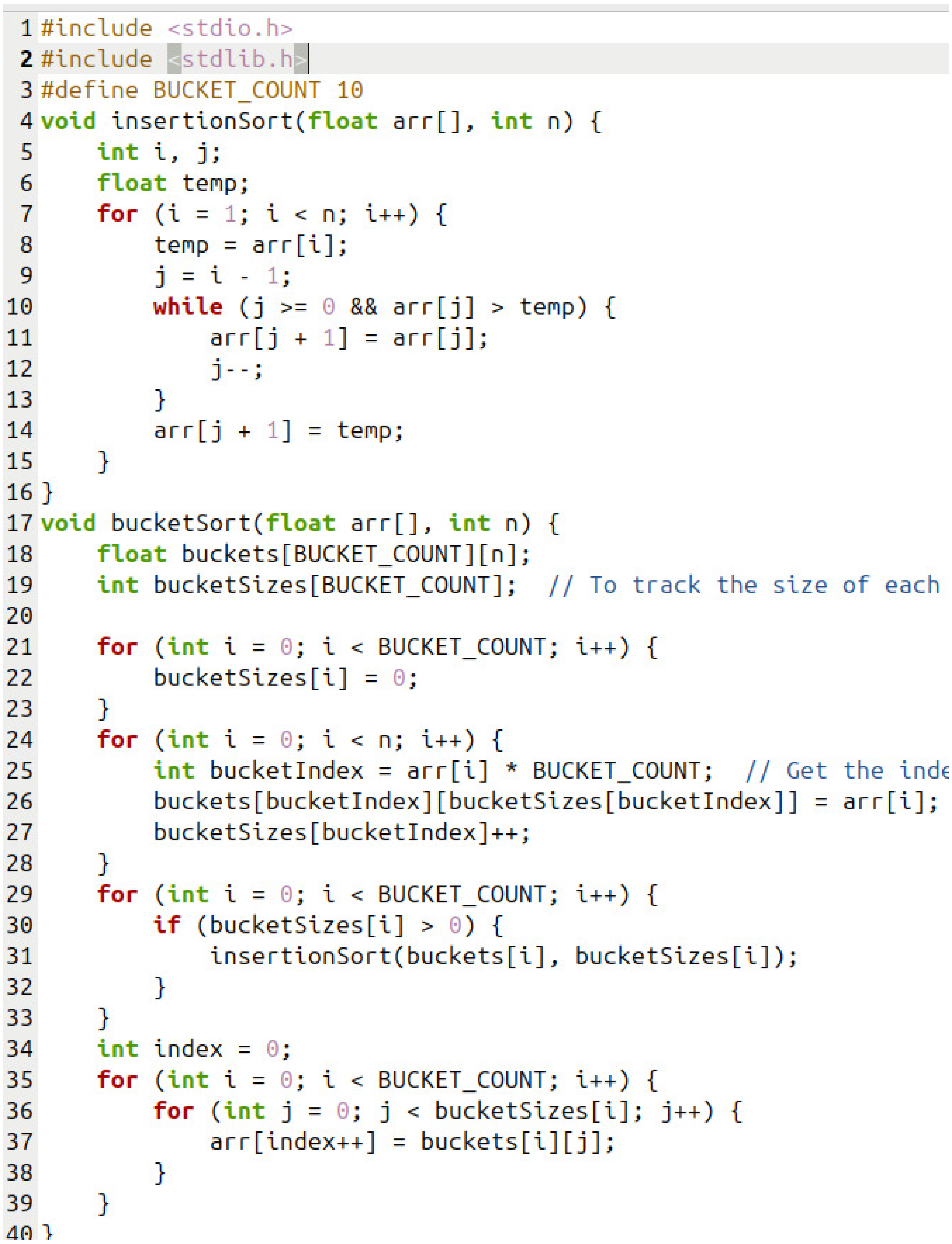
```c
#include <stdio.h>
void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: \n");
    printArray(arr, n);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

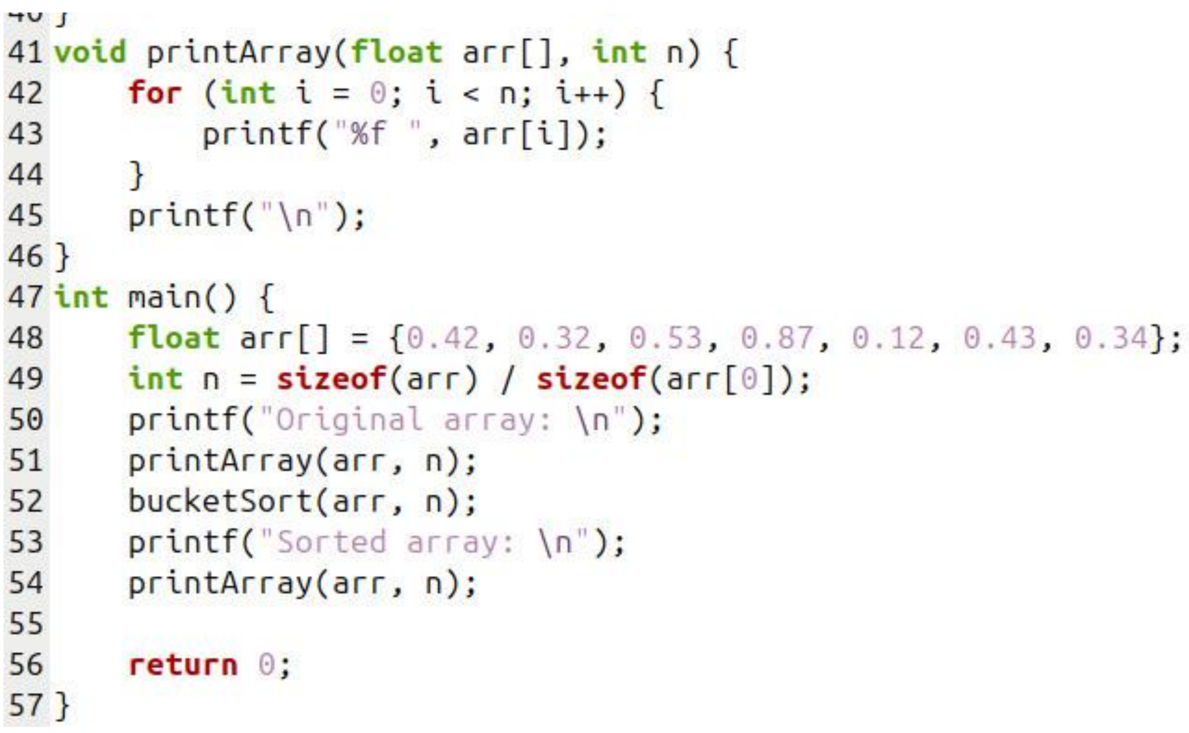


```
dhusyanth@Ubuntu:~$ gedit selection_sort.c

(gedit:7366): tepl-WARNING **: 15:52:43.647: Failed to load metadata: Err
ine 1 char 1: Document was empty or contained only whitespace
dhusyanth@Ubuntu:~$ gcc -o selection_sort sleection_sort.c
cc1: fatal error: sleection_sort.c: No such file or directory
compilation terminated.
dhusyanth@Ubuntu:~$ gcc -o selection_sort selection_sort.c
dhusyanth@Ubuntu:~$ .\selection_sort.c
.selection_sort.c: command not found
dhusyanth@Ubuntu:~$ .\selection_sort
.selection_sort: command not found
dhusyanth@Ubuntu:~$ ./selection_sort
Original array:
64 25 12 22 11
Sorted array:
11 12 22 25 64
```

## 4.)Bucket Sort:

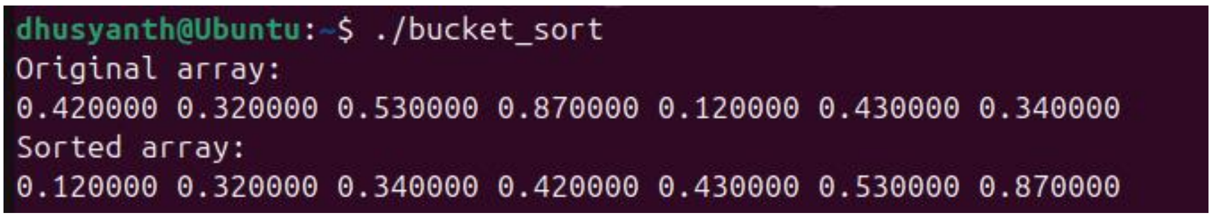


```c
#include <stdio.h>
#include <stdlib.h>
#define BUCKET_COUNT 10
void insertionSort(float arr[], int n) {
    int i, j;
    float temp;
    for (i = 1; i < n; i++) {
        temp = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}
void bucketSort(float arr[], int n) {
    float buckets[BUCKET_COUNT][n];
    int bucketSizes[BUCKET_COUNT];  // To track the size of each

    for (int i = 0; i < BUCKET_COUNT; i++) {
        bucketSizes[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        int bucketIndex = arr[i] * BUCKET_COUNT;  // Get the inde
        buckets[bucketIndex][bucketSizes[bucketIndex]] = arr[i];
        bucketSizes[bucketIndex]++;
    }
    for (int i = 0; i < BUCKET_COUNT; i++) {
        if (bucketSizes[i] > 0) {
            insertionSort(buckets[i], bucketSizes[i]);
        }
    }
    int index = 0;
    for (int i = 0; i < BUCKET_COUNT; i++) {
        for (int j = 0; j < bucketSizes[i]; j++) {
            arr[index++] = buckets[i][j];
        }
    }
}
```



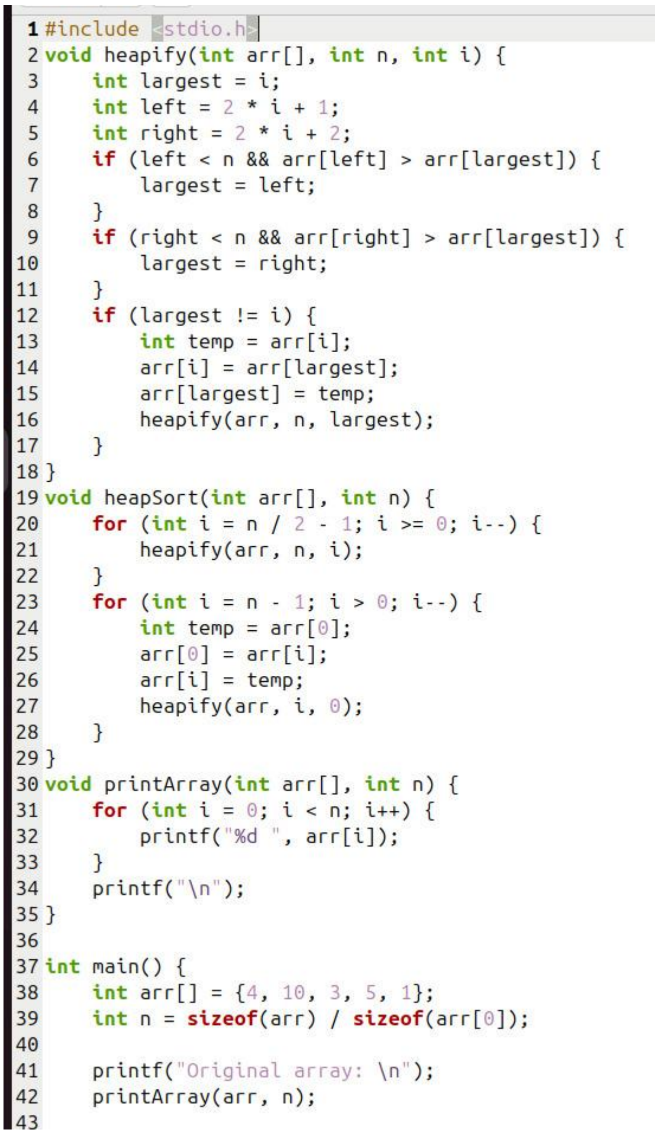
```c
}
void printArray(float arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%f ", arr[i]);
    }
    printf("\n");
}
int main() {
    float arr[] = {0.42, 0.32, 0.53, 0.87, 0.12, 0.43, 0.34};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: \n");
    printArray(arr, n);
    bucketSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

Output:

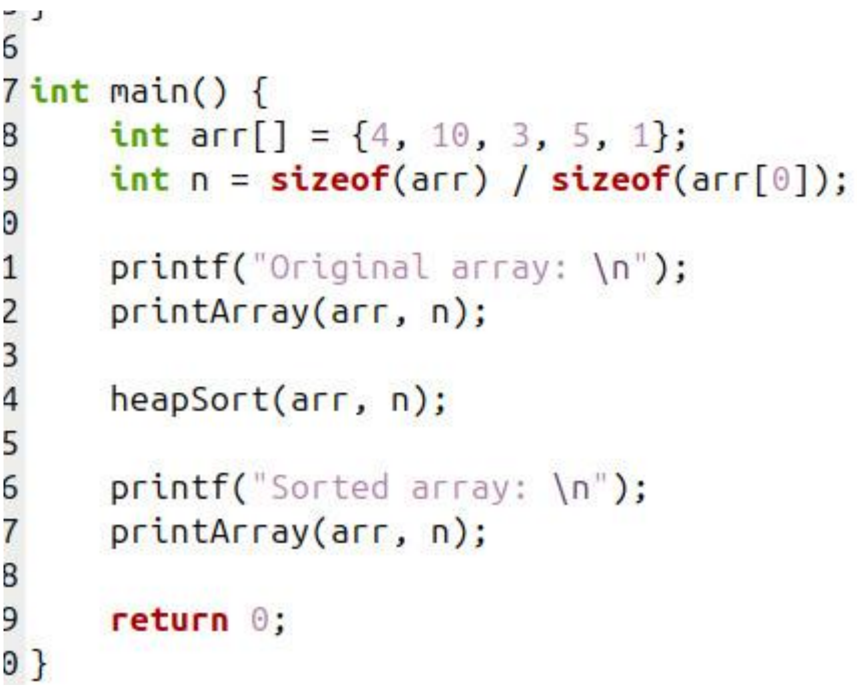


```
dhusyanth@Ubuntu:~$ ./bucket_sort
Original array:
0.420000 0.320000 0.530000 0.870000 0.120000 0.430000 0.340000
Sorted array:
0.120000 0.320000 0.340000 0.420000 0.430000 0.530000 0.870000
```

## 5.)Heap Sort:

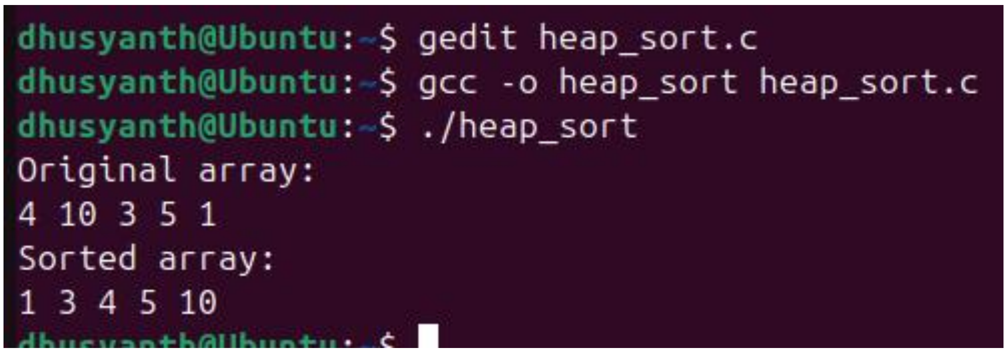


```c
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    for (int i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

```



```c

int main() {
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

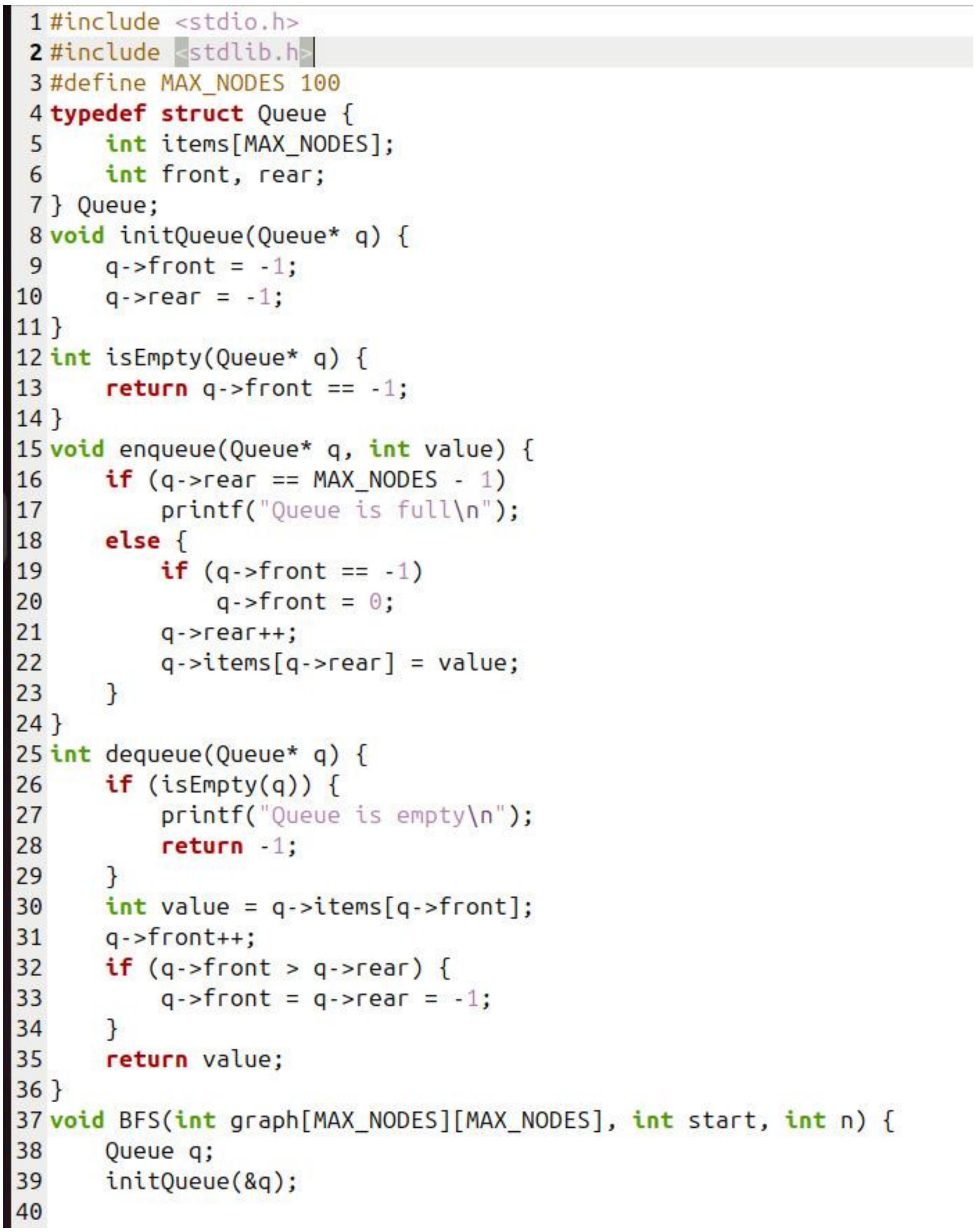
Output:



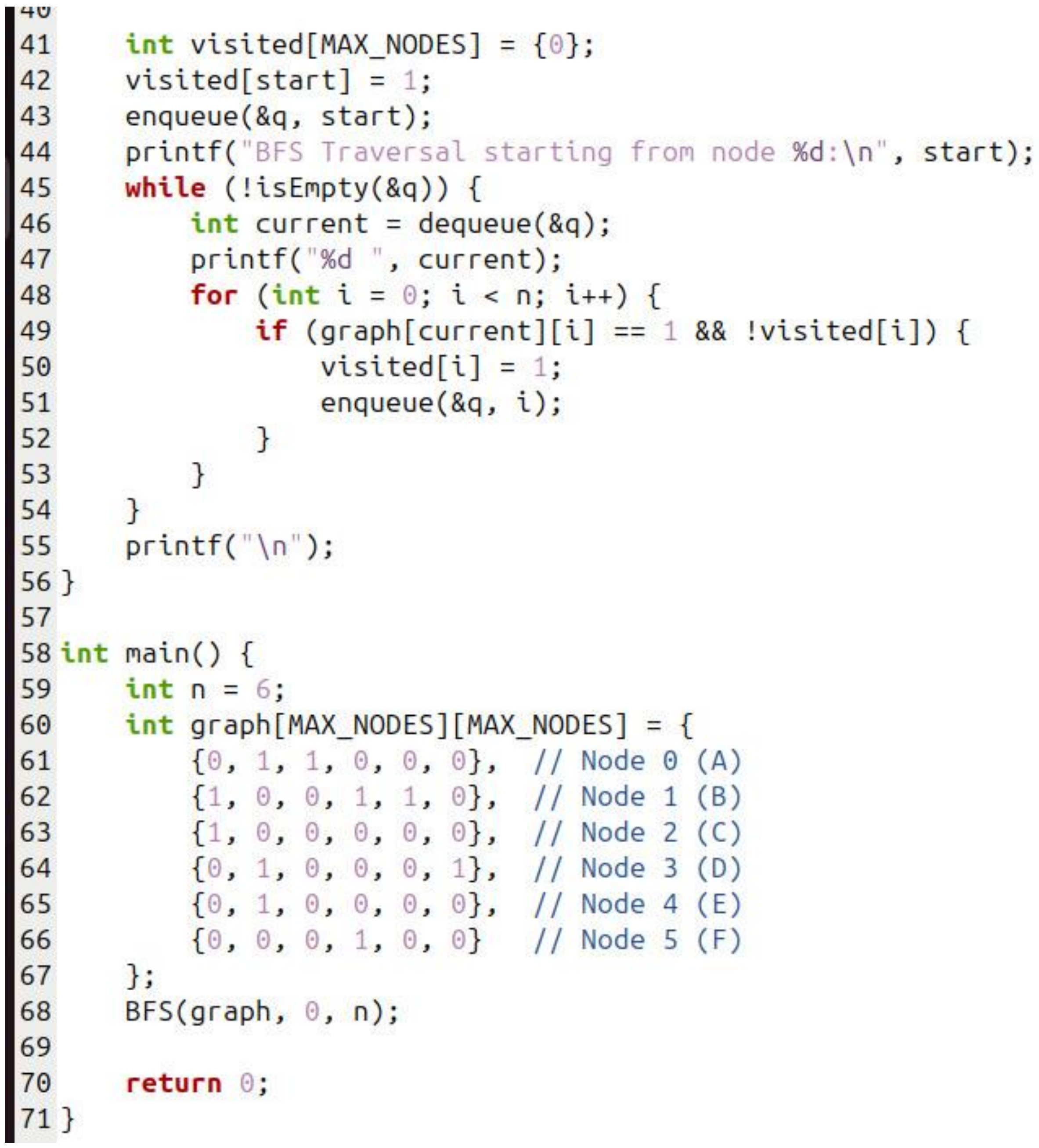
```
dhusyanth@Ubuntu:~$ gedit heap_sort.c
dhusyanth@Ubuntu:~$ gcc -o heap_sort heap_sort.c
dhusyanth@Ubuntu:~$ ./heap_sort
Original array:
4 10 3 5 1
Sorted array:
1 3 4 5 10
dhusyanth@Ubuntu:~$
```

6.)Bfs:

CH.SC.U4CSE24161



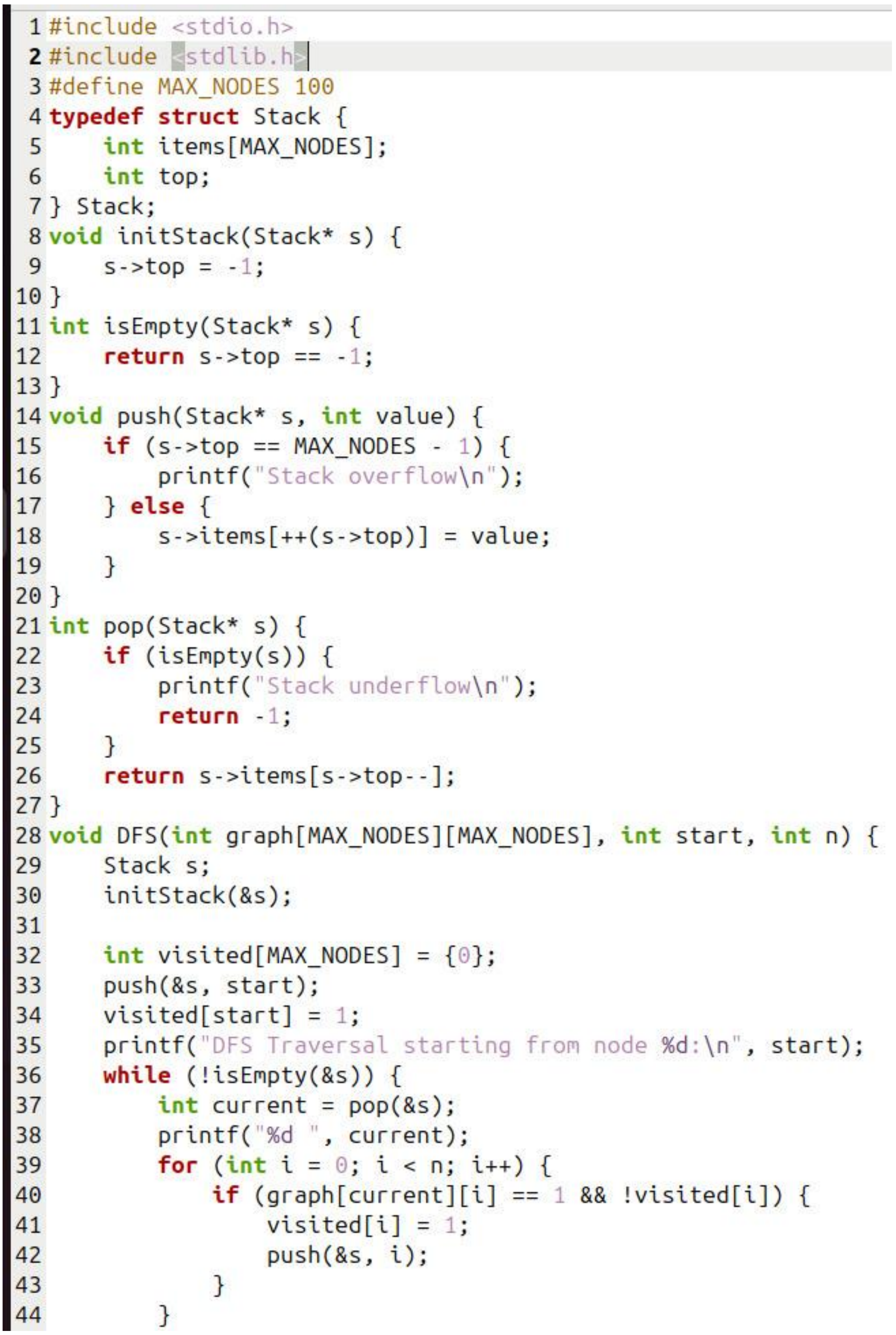
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100
typedef struct Queue {
    int items[MAX_NODES];
    int front, rear;
} Queue;
void initQueue(Queue* q) {
    q->front = -1;
    q->rear = -1;
}
int isEmpty(Queue* q) {
    return q->front == -1;
}
void enqueue(Queue* q, int value) {
    if (q->rear == MAX_NODES - 1)
        printf("Queue is full\n");
    else {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int value = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return value;
}
void BFS(int graph[MAX_NODES][MAX_NODES], int start, int n) {
    Queue q;
    initQueue(&q);

```



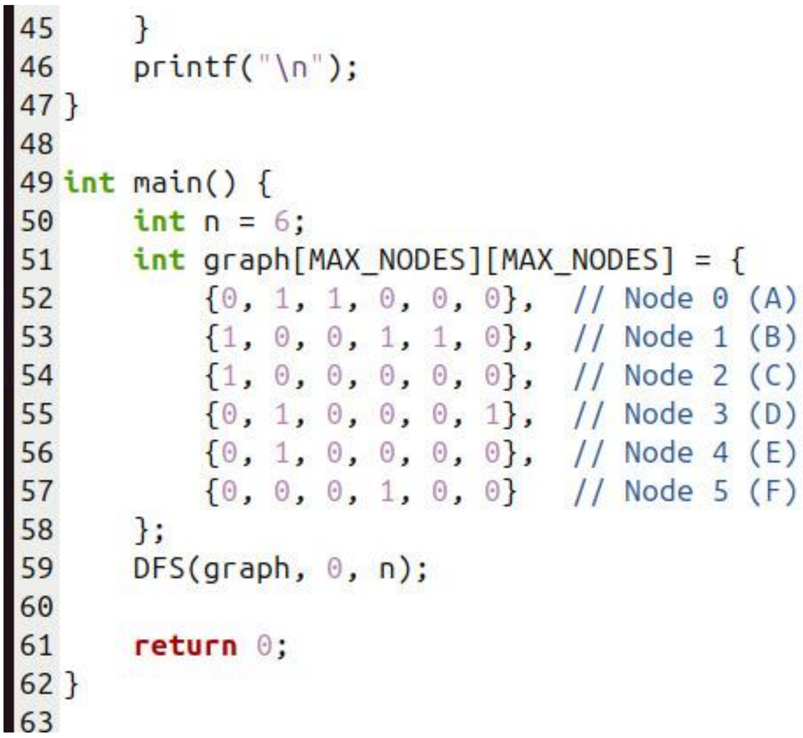
```c
    int visited[MAX_NODES] = {0};
    visited[start] = 1;
    enqueue(&q, start);
    printf("BFS Traversal starting from node %d:\n", start);
    while (!isEmpty(&q)) {
        int current = dequeue(&q);
        printf("%d ", current);
        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(&q, i);
            }
        }
    }
    printf("\n");
}

int main() {
    int n = 6;
    int graph[MAX_NODES][MAX_NODES] = {
        {0, 1, 1, 0, 0, 0},  // Node 0 (A)
        {1, 0, 0, 1, 1, 0},  // Node 1 (B)
        {1, 0, 0, 0, 0, 0},  // Node 2 (C)
        {0, 1, 0, 0, 0, 1},  // Node 3 (D)
        {0, 1, 0, 0, 0, 0},  // Node 4 (E)
        {0, 0, 0, 1, 0, 0}   // Node 5 (F)
    };
    BFS(graph, 0, n);

    return 0;
}
```

Output:

## 7.)Dfs:

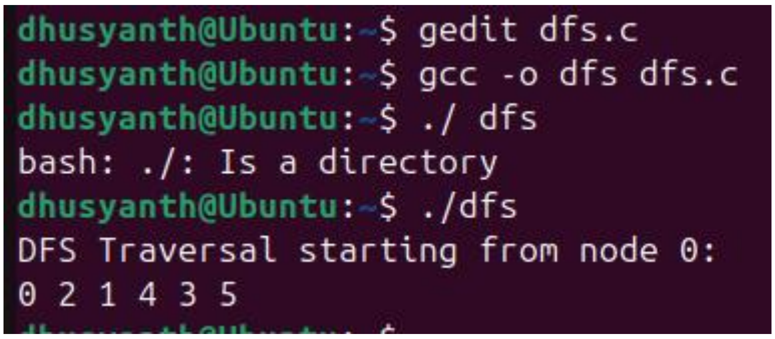


```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100
typedef struct Stack {
    int items[MAX_NODES];
    int top;
} Stack;
void initStack(Stack* s) {
    s->top = -1;
}
int isEmpty(Stack* s) {
    return s->top == -1;
}
void push(Stack* s, int value) {
    if (s->top == MAX_NODES - 1) {
        printf("Stack overflow\n");
    } else {
        s->items[++(s->top)] = value;
    }
}
int pop(Stack* s) {
    if (isEmpty(s)) {
        printf("Stack underflow\n");
        return -1;
    }
    return s->items[s->top--];
}
void DFS(int graph[MAX_NODES][MAX_NODES], int start, int n) {
    Stack s;
    initStack(&s);

    int visited[MAX_NODES] = {0};
    push(&s, start);
    visited[start] = 1;
    printf("DFS Traversal starting from node %d:\n", start);
    while (!isEmpty(&s)) {
        int current = pop(&s);
        printf("%d ", current);
        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                push(&s, i);
            }
        }
```



```c
    }
    printf("\n");
}

int main() {
    int n = 6;
    int graph[MAX_NODES][MAX_NODES] = {
        {0, 1, 1, 0, 0, 0},  // Node 0 (A)
        {1, 0, 0, 1, 1, 0},  // Node 1 (B)
        {1, 0, 0, 0, 0, 0},  // Node 2 (C)
        {0, 1, 0, 0, 0, 1},  // Node 3 (D)
        {0, 1, 0, 0, 0, 0},  // Node 4 (E)
        {0, 0, 0, 1, 0, 0}   // Node 5 (F)
    };
    DFS(graph, 0, n);

    return 0;
}
```

Output:



```
dhusyanth@Ubuntu:~$ gedit dfs.c
dhusyanth@Ubuntu:~$ gcc -o dfs dfs.c
dhusyanth@Ubuntu:~$ ./ dfs
bash: ./: Is a directory
dhusyanth@Ubuntu:~$ ./dfs
DFS Traversal starting from node 0:
0 2 1 4 3 5
```