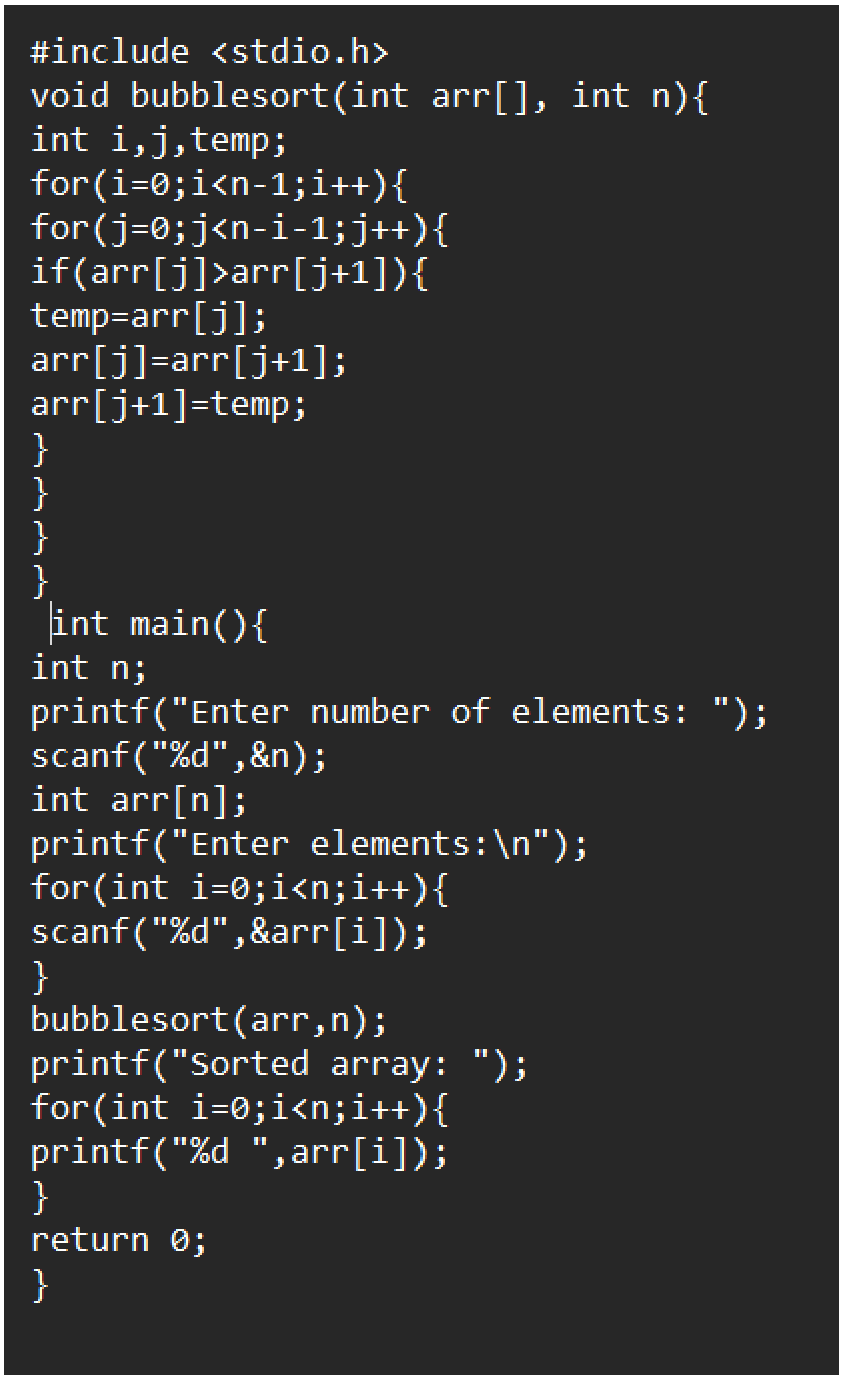
23CSE211
LAB
WEEK: 2

CH.SC.U4CSE24164

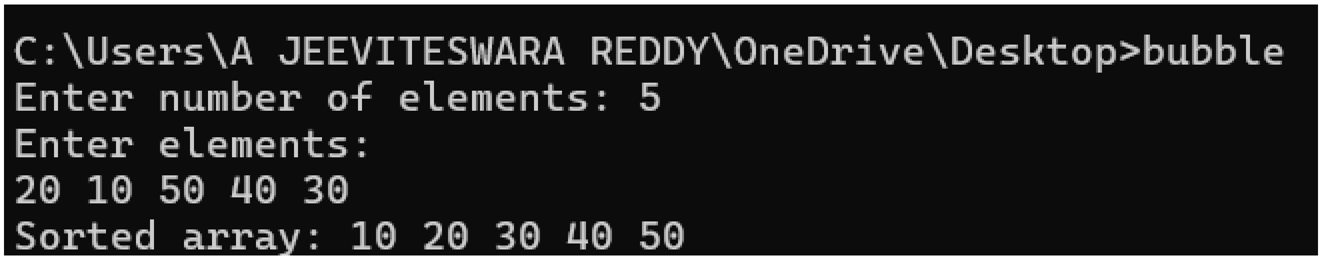
A.Jeeviteswara Reddy

1. Bubble sorting
   Code:



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

OUTPUT:



```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>bubble
Enter number of elements: 5
Enter elements:
20 10 50 40 30
Sorted array: 10 20 30 40 50
```

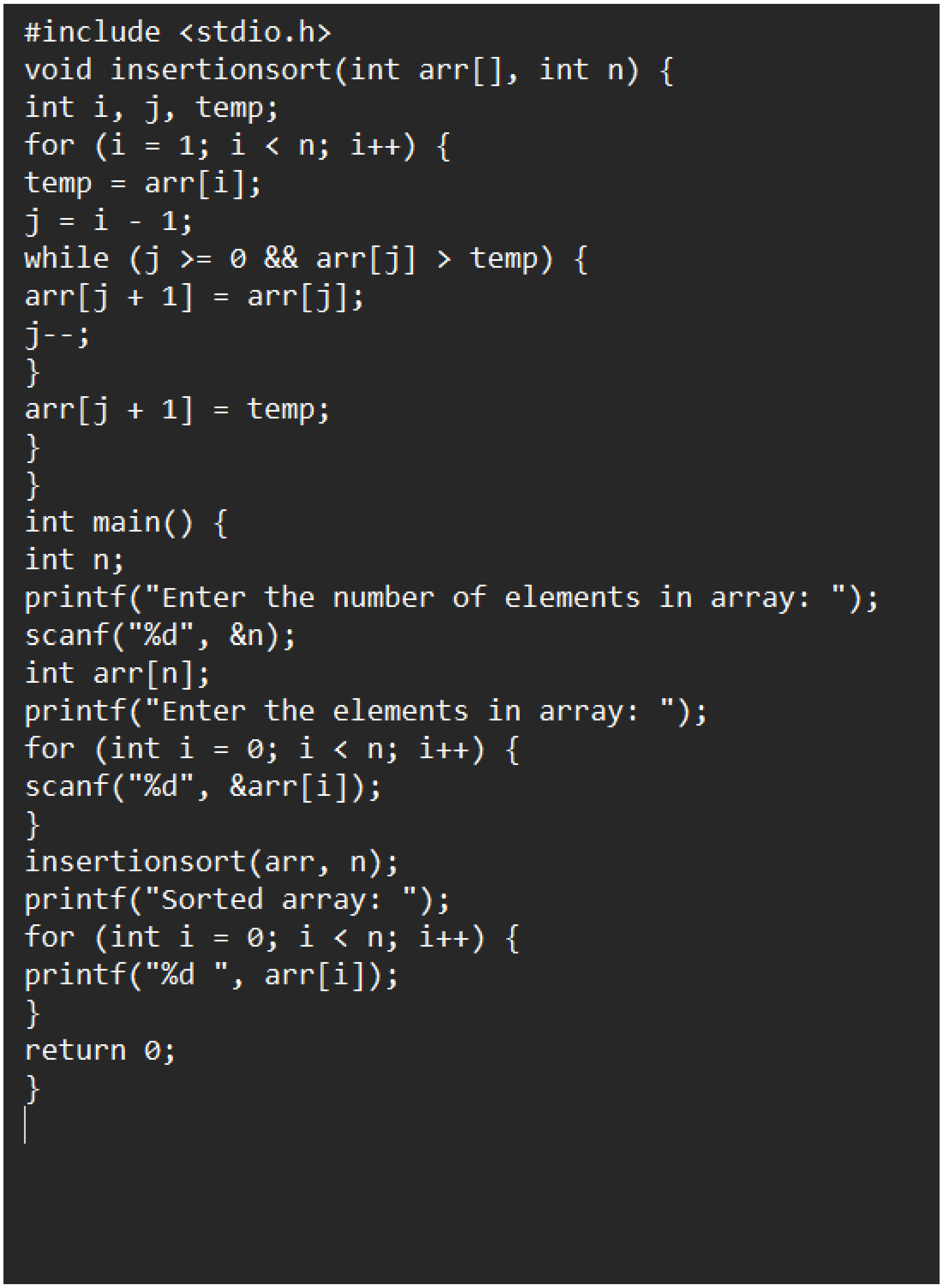
space and time complexity for:

time: O(n^2) will be the worst because every element will be compared to every other element. O(n^2) will be the best case because even if the list is sorted the comparision of elements still take place.

Space : O(1) because the sorting is done using only one temporary variable.

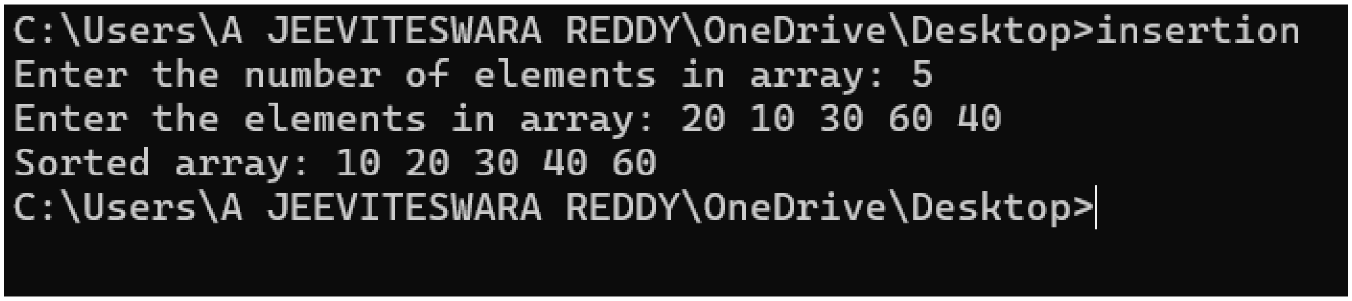
2. Insertion sort

C Code:



```c
#include <stdio.h>
void insertionsort(int arr[], int n) {
int i, j, 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;
}
}
int main() {
int n;
printf("Enter the number of elements in array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements in array: ");
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
insertionsort(arr, n);
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
return 0;
}
```

OUTPUT:



```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>insertion
Enter the number of elements in array: 5
Enter the elements in array: 20 10 30 60 40
Sorted array: 10 20 30 40 60
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>
```

Time Complexity:

Best Case: O(n) happens when the array is already sorted the inner loop runs only once per element.

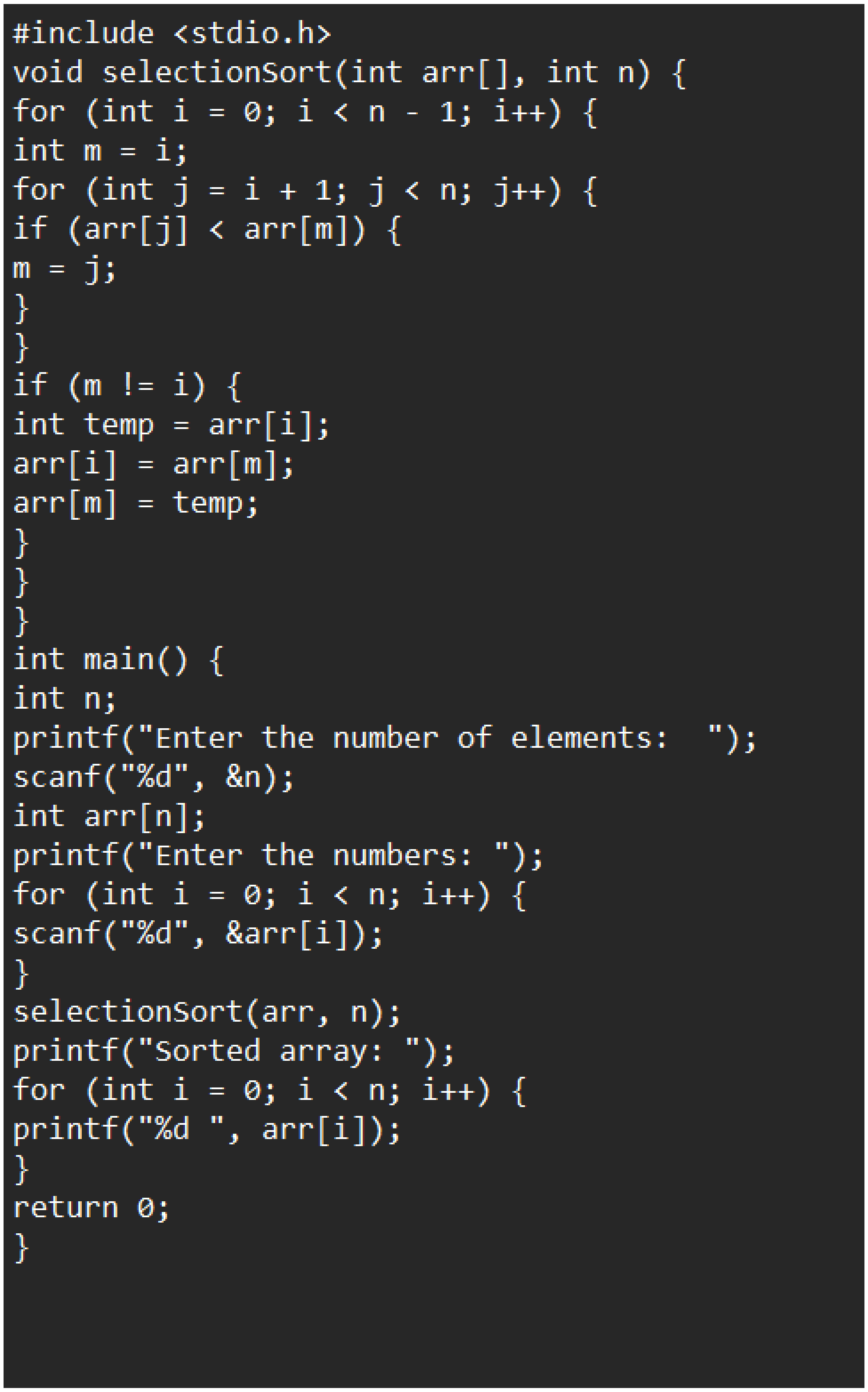
Worst Case: O(n^2) happens when the array is reverse sorted each element shifts all its previous elements.

Space Complexity:

O(1) same array is being used with few extra elements.

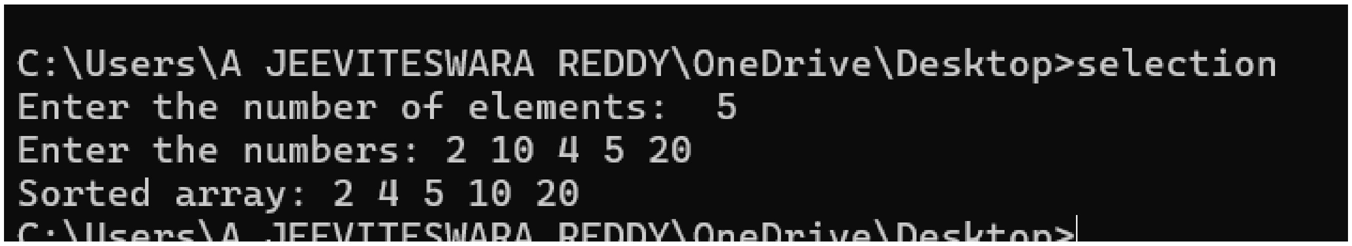
3. Selection sort:

C Code:



```c
#include <stdio.h>
void selectionSort(int arr[], int n) {
for (int i = 0; i < n - 1; i++) {
int m = i;
for (int j = i + 1; j < n; j++) {
if (arr[j] < arr[m]) {
m = j;
}
}
if (m != i) {
int temp = arr[i];
arr[i] = arr[m];
arr[m] = temp;
}
}
}
int main() {
int n;
printf("Enter the number of elements:  ");
scanf("%d", &n);
int arr[n];
printf("Enter the numbers: ");
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
selectionSort(arr, n);
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
}
return 0;
}
```

OUTPUT:



```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>selection
Enter the number of elements:  5
Enter the numbers: 2 10 4 5 20
Sorted array: 2 4 5 10 20
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>
```

Time complexity:

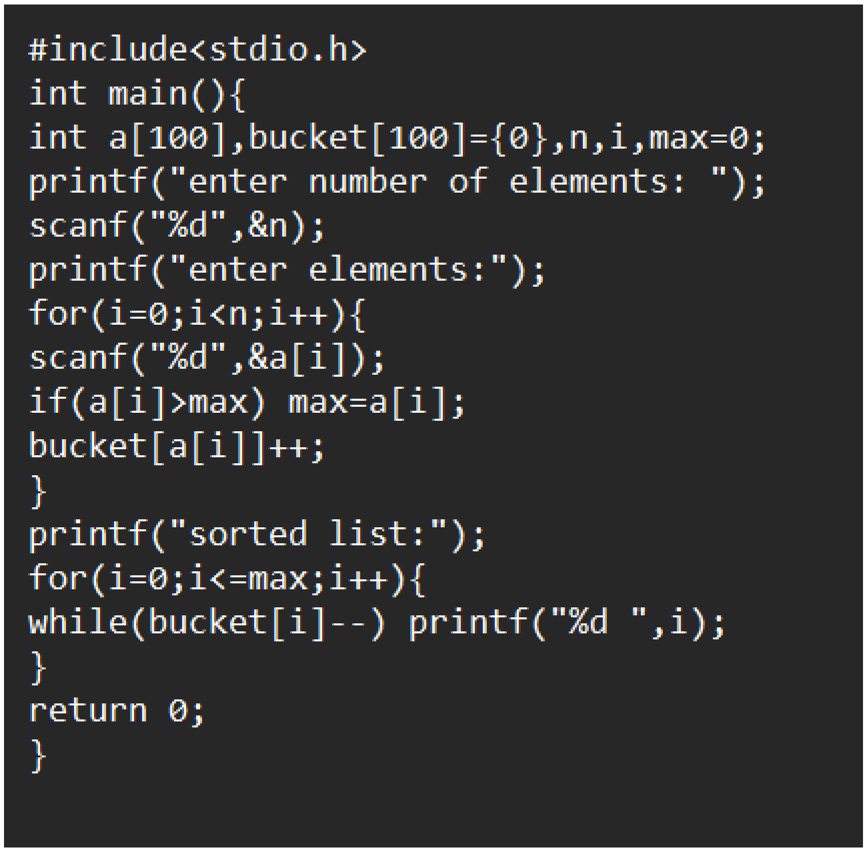
Best case: O(n^2) even if array is sorted it still compares eith every element.
Worst Case: O(n^2) if the array is reverse sorted all comparisions happen.

Space complexity:

O(1) : sorting is done using only a single variable for swapping.

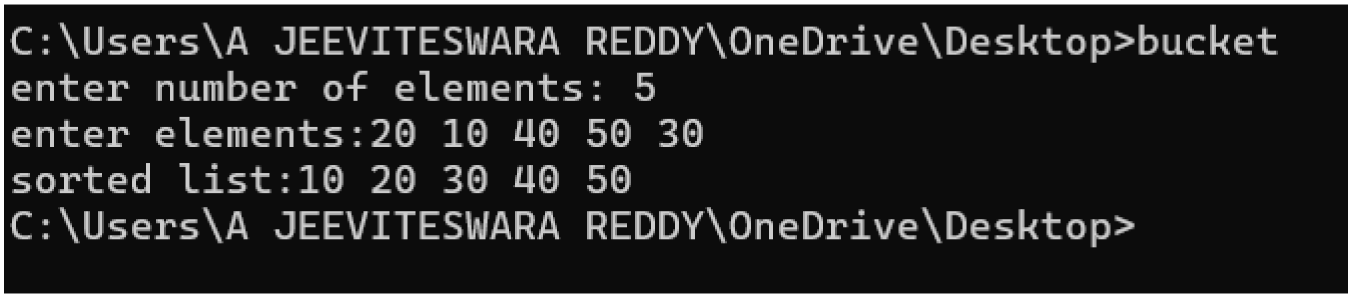
4. Bucket sort:

C Code:



```c
#include<stdio.h>
int main(){
int a[100],bucket[100]={0},n,i,max=0;
printf("enter number of elements: ");
scanf("%d",&n);
printf("enter elements:");
for(i=0;i<n;i++){
scanf("%d",&a[i]);
if(a[i]>max) max=a[i];
bucket[a[i]]++;
}
printf("sorted list:");
for(i=0;i<=max;i++){
while(bucket[i]--) printf("%d ",i);
}
return 0;
}
```

OUTPUT:



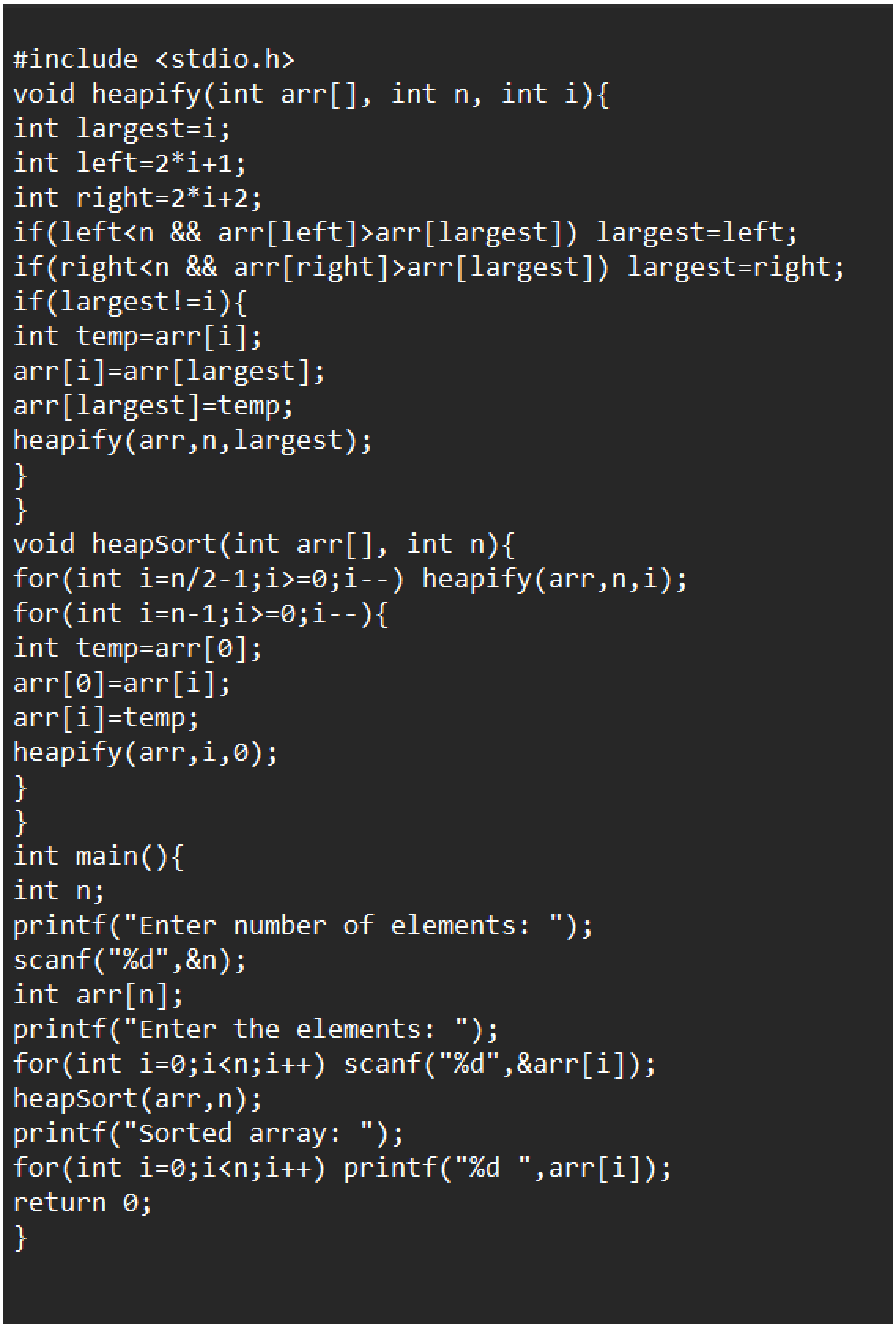
```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>bucket
enter number of elements: 5
enter elements:20 10 40 50 30
sorted list:10 20 30 40 50
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>
```

Time complexity:

Best Case: O(n + k) occurs when the elements are uniformly distributed; n is the number of elements, k is the maximum element.
Worst Case: O(n + k) even if all elements are the same or concentrated, we still need to iterate over the bucket array.

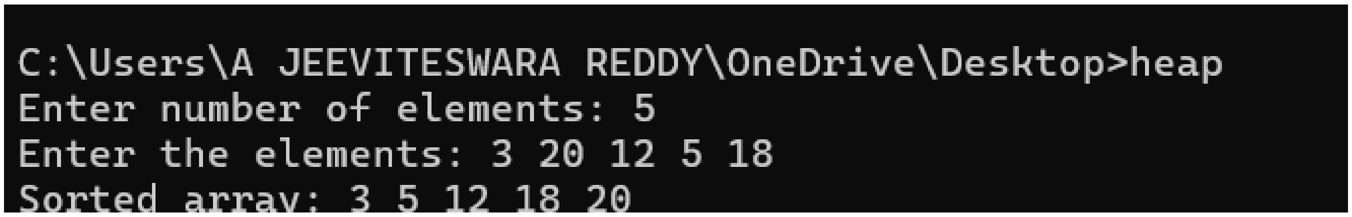
Space complexity:
O(n+k) : O(n) for storing input elements
bucket[100] O(k) for bucket array to count occurrences

5. Heap sort:
   C Code:



```
#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);
}
}
int main(){
int n;
printf("Enter number of elements: ");
scanf("%d",&n);
int arr[n];
printf("Enter the elements: ");
for(int i=0;i<n;i++) scanf("%d",&arr[i]);
heapSort(arr,n);
printf("Sorted array: ");
for(int i=0;i<n;i++) printf("%d ",arr[i]);
return 0;
}
```

OUTPUT:



```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>heap
Enter number of elements: 5
Enter the elements: 3 20 12 5 18
Sorted array: 3 5 12 18 20
```

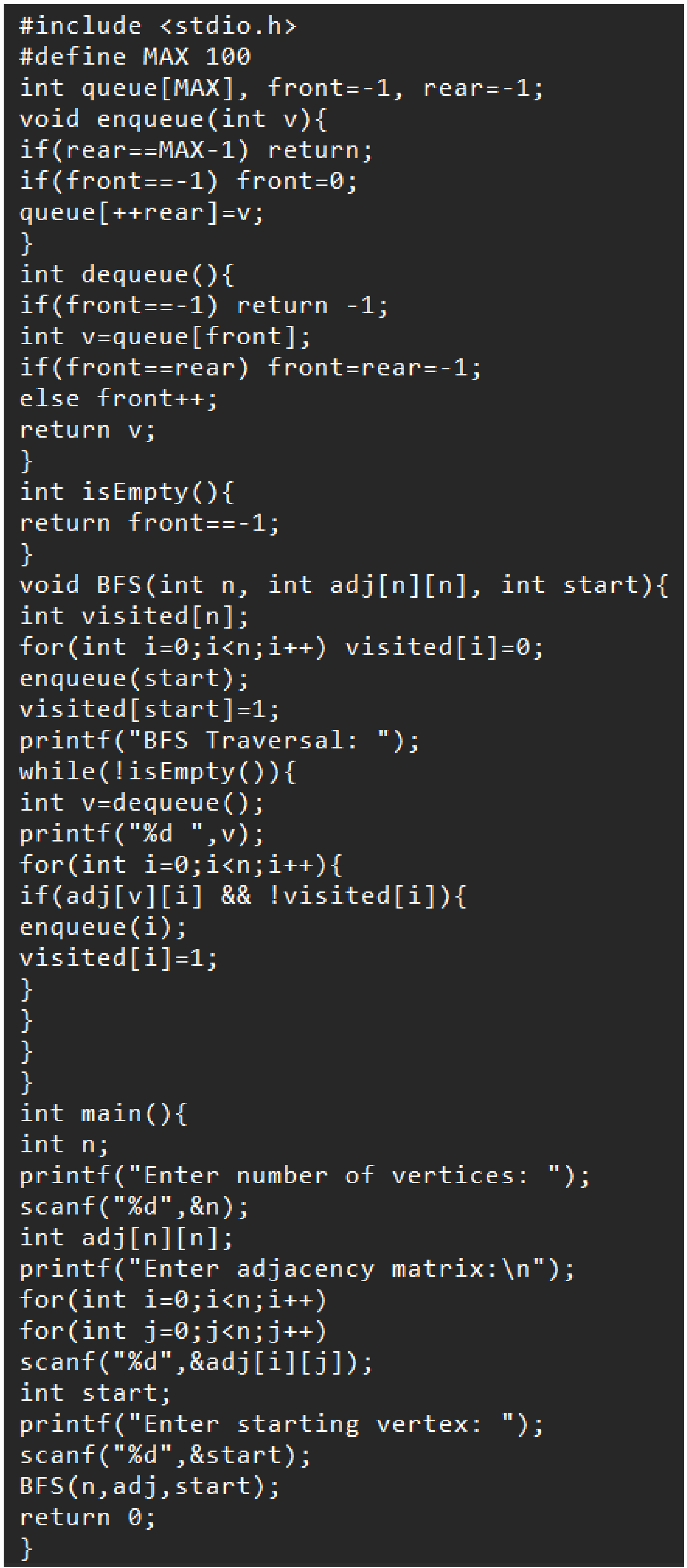
Time complexity:

Best Case: O(n log n) building the heap and performing heapify operations take logarithmic time per element.
Worst Case: O(n log n) same as best case; heap operations do not depend on input order.

Space complexity:
O(1) — Heap sort is done using a few extra variables.

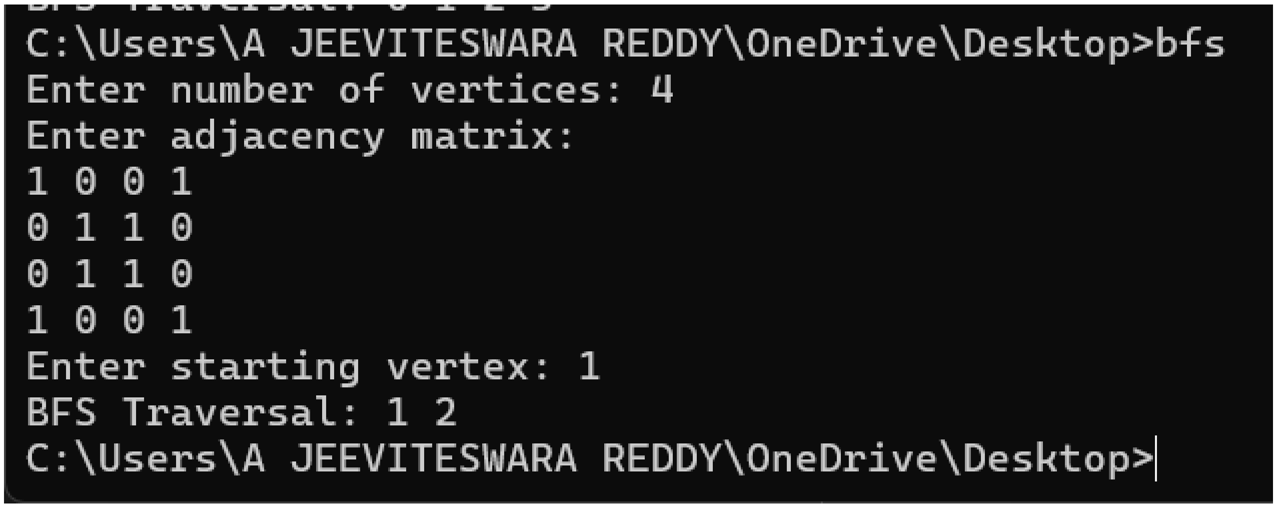
6.BFS traversal:

C Code:



```c
#include <stdio.h>
#define MAX 100
int queue[MAX], front=-1, rear=-1;
void enqueue(int v){
if(rear==MAX-1) return;
if(front==-1) front=0;
queue[++rear]=v;
}
int dequeue(){
if(front==-1) return -1;
int v=queue[front];
if(front==rear) front=rear=-1;
else front++;
return v;
}
int isEmpty(){
return front==-1;
}
void BFS(int n, int adj[n][n], int start){
int visited[n];
for(int i=0;i<n;i++) visited[i]=0;
enqueue(start);
visited[start]=1;
printf("BFS Traversal: ");
while(!isEmpty()){
int v=dequeue();
printf("%d ",v);
for(int i=0;i<n;i++){
if(adj[v][i] && !visited[i]){
enqueue(i);
visited[i]=1;
}
}
}
}
int main(){
int n;
printf("Enter number of vertices: ");
scanf("%d",&n);
int adj[n][n];
printf("Enter adjacency matrix:\n");
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
scanf("%d",&adj[i][j]);
int start;
printf("Enter starting vertex: ");
scanf("%d",&start);
BFS(n,adj,start);
return 0;
}
```

OUTPUT:



```
BFS Traversal: 0 1 2 3
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>bfs
Enter number of vertices: 4
Enter adjacency matrix:
1 0 0 1
0 1 1 0
0 1 1 0
1 0 0 1
Enter starting vertex: 1
BFS Traversal: 1 2
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>
```

Time complexity:

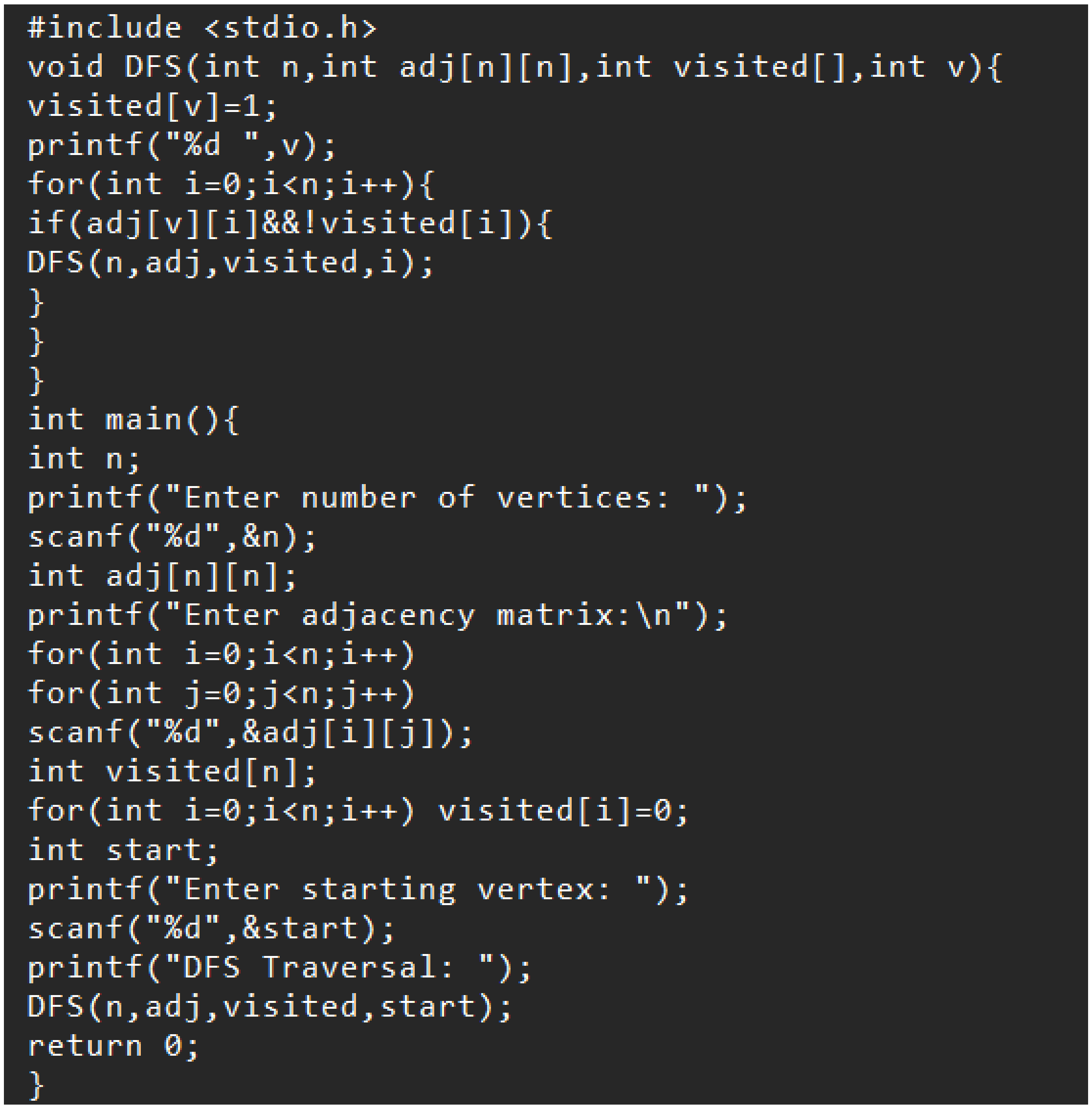
O(n^2) for adjacency matrix representation (V = number of vertices).

Space complexity:

O(n) for the visited array and the queue.

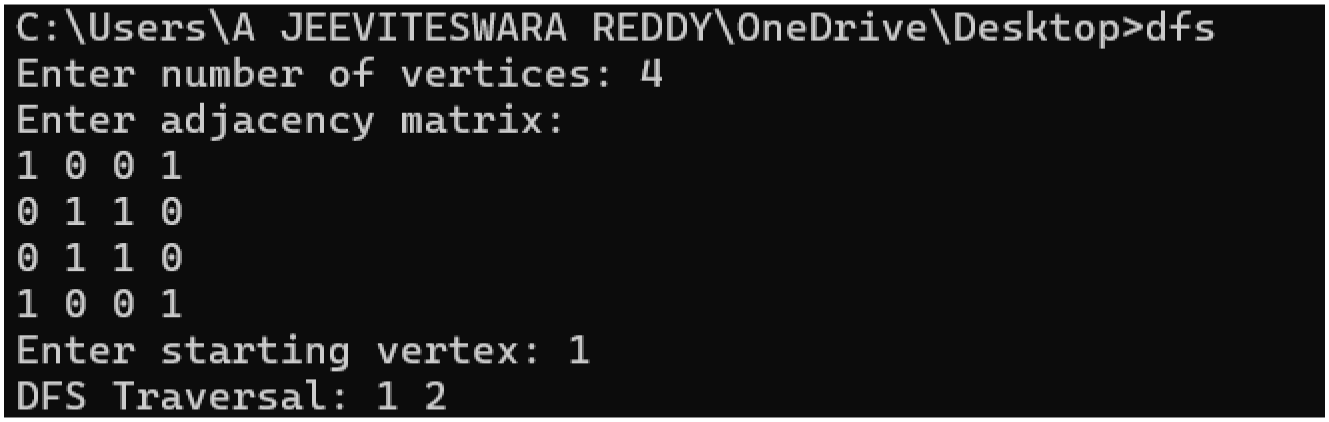
7. Dfs traversal

C Code:



```c
#include <stdio.h>
void DFS(int n,int adj[n][n],int visited[],int v){
visited[v]=1;
printf("%d ",v);
for(int i=0;i<n;i++){
if(adj[v][i]&&!visited[i]){
DFS(n,adj,visited,i);
}
}
}
int main(){
int n;
printf("Enter number of vertices: ");
scanf("%d",&n);
int adj[n][n];
printf("Enter adjacency matrix:\n");
for(int i=0;i<n;i++)
for(int j=0;j<n;j++)
scanf("%d",&adj[i][j]);
int visited[n];
for(int i=0;i<n;i++) visited[i]=0;
int start;
printf("Enter starting vertex: ");
scanf("%d",&start);
printf("DFS Traversal: ");
DFS(n,adj,visited,start);
return 0;
}
```

OUTPUT:



```
C:\Users\A JEEVITESWARA REDDY\OneDrive\Desktop>dfs
Enter number of vertices: 4
Enter adjacency matrix:
1 0 0 1
0 1 1 0
0 1 1 0
1 0 0 1
Enter starting vertex: 1
DFS Traversal: 1 2
```

Time complexity:

O(n^2) for adjacency matrix representation (V = number of vertices).

Space complexity:

O(n) for the visited array and the queue.