DESIGN AND ANALYSIS OF ALGORITHMS

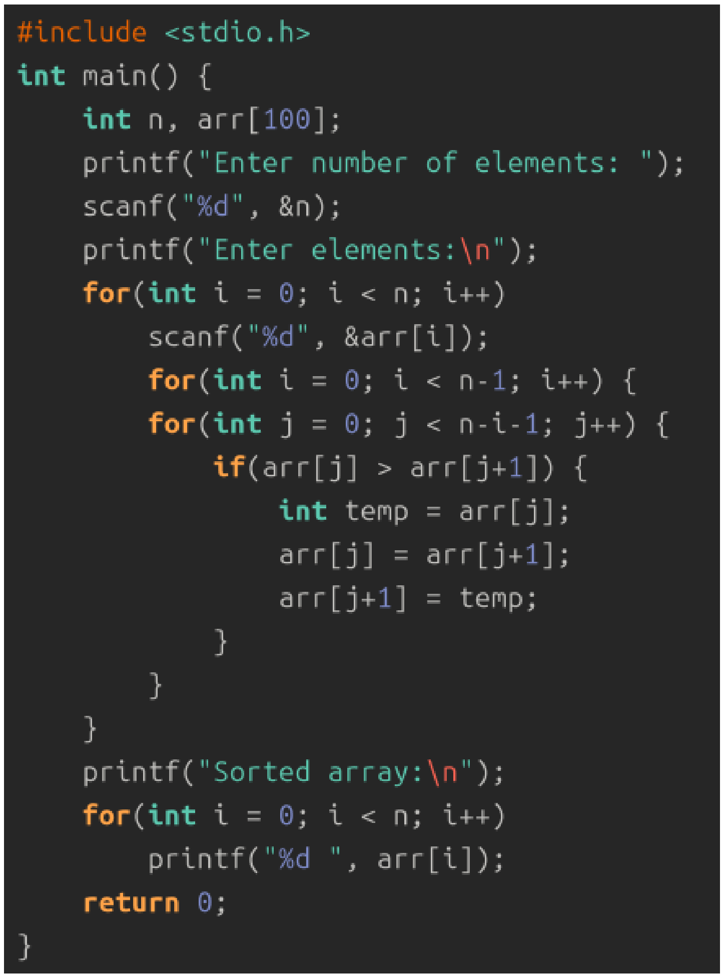
LAB WORKBOOK

WEEK – 2 & 3

NAME	: M.RAAHITHYA
ROLL NUMBER	: CH.SC.U4CSE24124
CLASS	: CSE-B

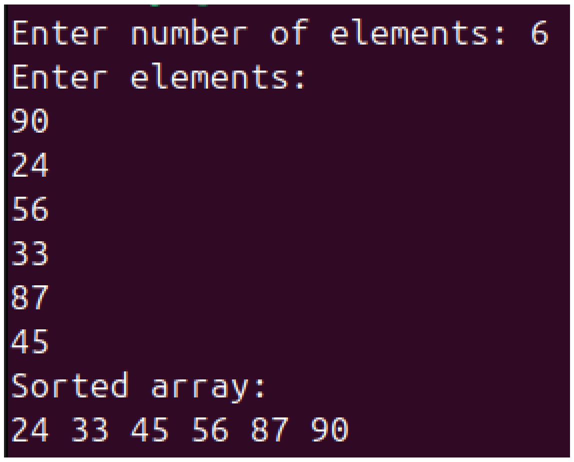
1.BUBBLE SORT:

CODE:



```c
#include <stdio.h>
int main() {
    int n, arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
        for(int i = 0; i < n-1; i++) {
        for(int j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:



```
Enter number of elements: 6
Enter elements:
90
24
56
33
87
45
Sorted array:
24 33 45 56 87 90
```

Time Complexity:

Outer loop runs (n−1) times.

Inner loop runs (n−i−1) times.

Total comparisons ≈ n * n.

Time Complexity = O(n^2)

Space Complexity:

int n → 4 bytes

int i → 4 bytes

int j → 4 bytes

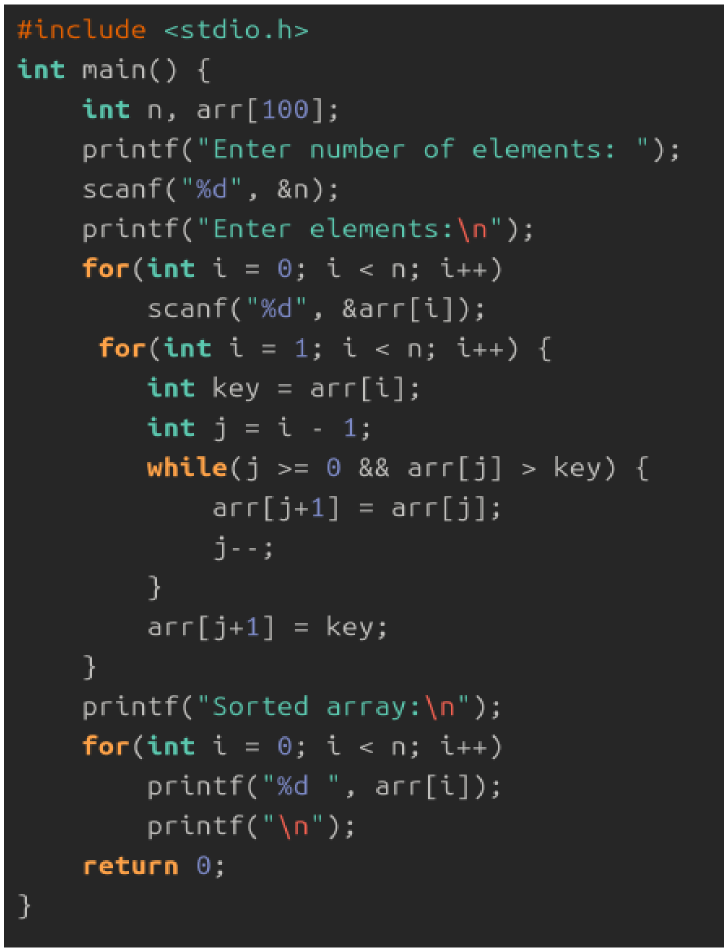
int temp → 4 bytes

Total space used is constant.

Space Complexity = O(1)

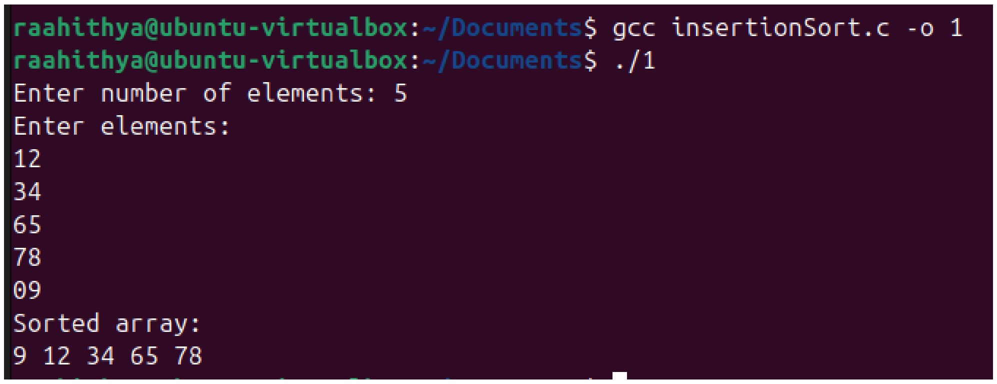
2.INSERTION SORT

CODE:



```c
#include <stdio.h>
int main() {
    int n, arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
     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;
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
        printf("\n");
    return 0;
}
```

OUTPUT:



```
raahithya@ubuntu-virtualbox:~/Documents$ gcc insertionSort.c -o 1
raahithya@ubuntu-virtualbox:~/Documents$ ./1
Enter number of elements: 5
Enter elements:
12
34
65
78
09
Sorted array:
9 12 34 65 78
```

Time Complexity:

Outer loop runs n times.

Inner while loop runs up to n times.

Total operations-- n * n.

Time Complexity = O(n^2)

Space Complexity:

int n → 4 bytes

int i → 4 bytes

int j → 4 bytes

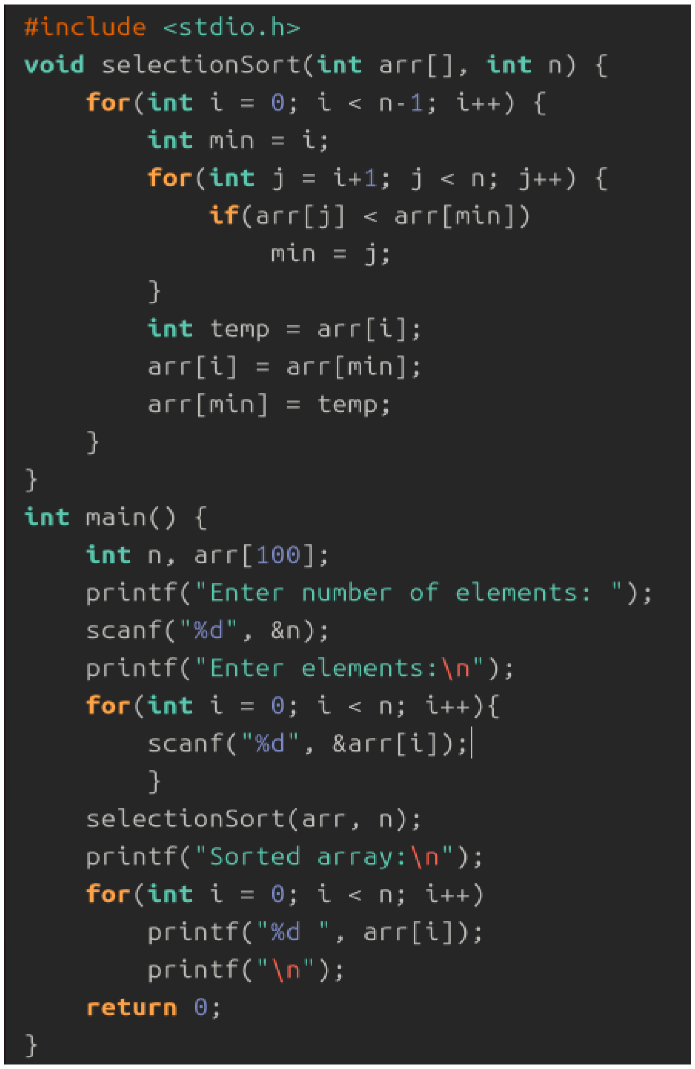
int key → 4 bytes

Only constant extra space is used.

Space Complexity = O(1)

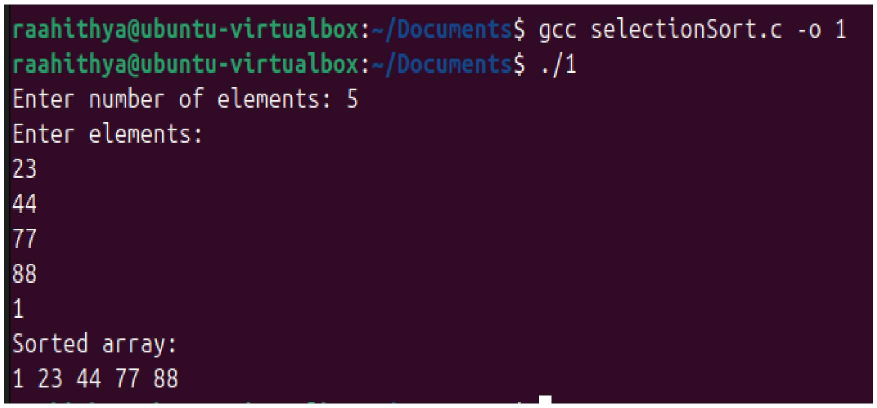
3.SELECTION SORT

CODE:



```c
#include <stdio.h>
void selectionSort(int arr[], int n) {
    for(int i = 0; i < n-1; i++) {
        int min = i;
        for(int j = i+1; j < n; j++) {
            if(arr[j] < arr[min])
                min = j;
        }
        int temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }
}
int main() {
    int n, arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
        }
    selectionSort(arr, n);
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
        printf("\n");
    return 0;
}
```

OUTPUT:



```
raahithya@ubuntu-virtualbox:~/Documents$ gcc selectionSort.c -o 1
raahithya@ubuntu-virtualbox:~/Documents$ ./1
Enter number of elements: 5
Enter elements:
23
44
77
88
1
Sorted array:
1 23 44 77 88
```

Time Complexity:

Outer loop runs (n−1) times.

Inner loop runs n times.

Total comparisons ≈ n*n.

Time Complexity = O(n^2)

Space Complexity:

int n → 4 bytes

int i → 4 bytes

int j → 4 bytes

int min → 4 bytes

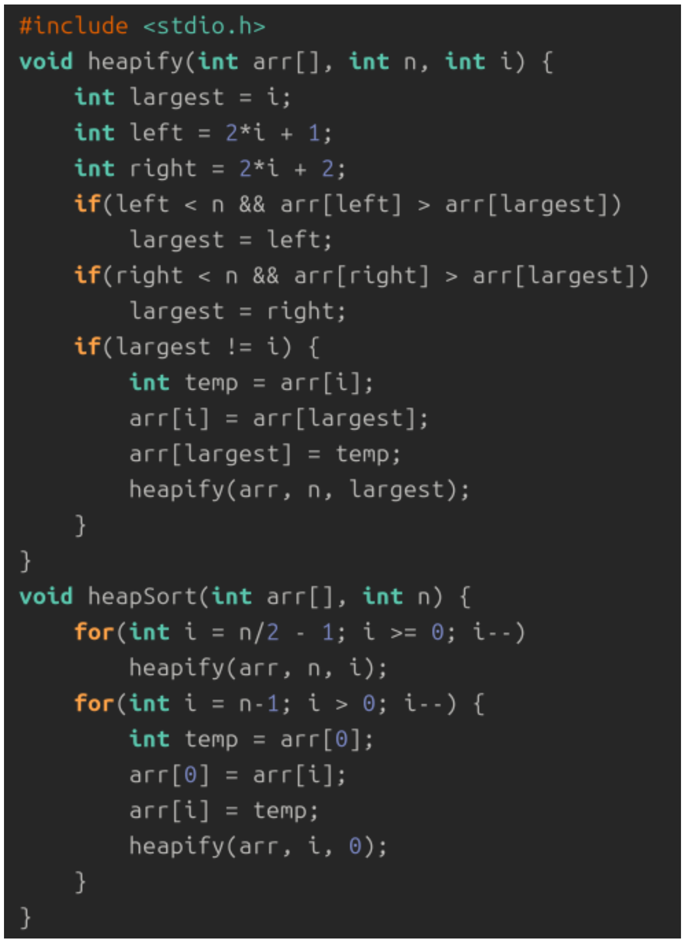
int temp → 4 bytes

Total space is constant.

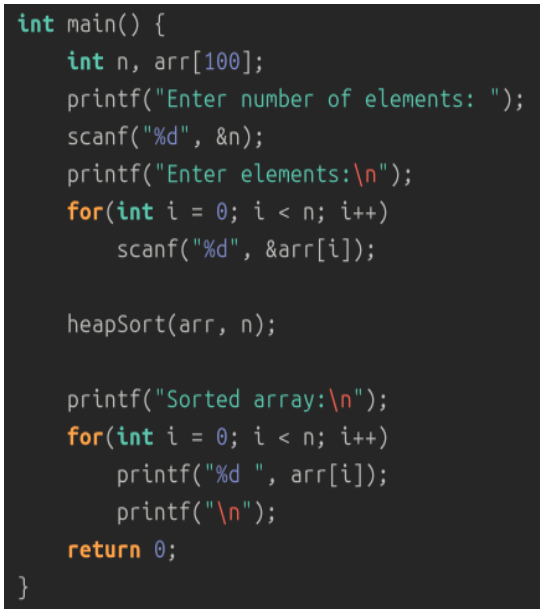
Space Complexity = O(1)

4.HEAP SORT:

CODE:



```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);
    }
}
```

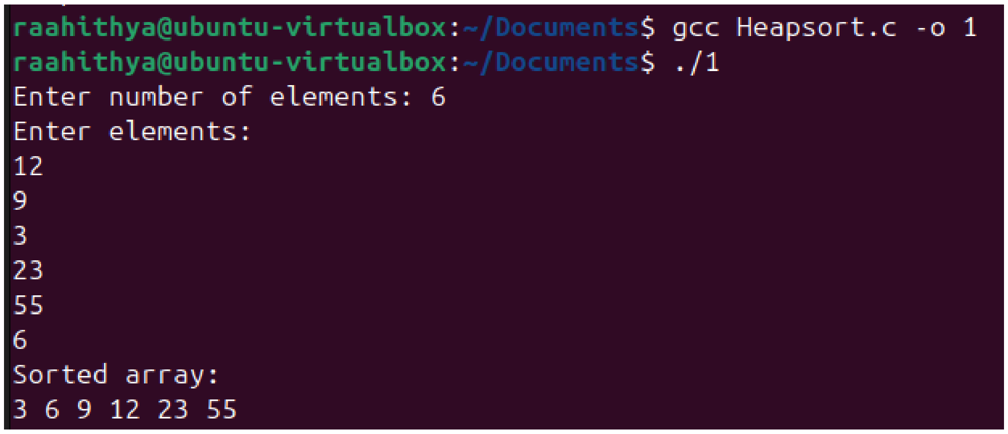


```c
int main() {
    int n, arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    heapSort(arr, n);

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

OUTPUT:



```
raahithya@ubuntu-virtualbox:~/Documents$ gcc Heapsort.c -o 1
raahithya@ubuntu-virtualbox:~/Documents$ ./1
Enter number of elements: 6
Enter elements:
12
9
3
23
55
6
Sorted array:
3 6 9 12 23 55
```

Time Complexity:

Heap is built in O(n).

Heapify is called n times.

Each heapify takes log n time.

Time Complexity = O(n log n)

Space Complexity:

int n → 4 bytes

int i → 4 bytes

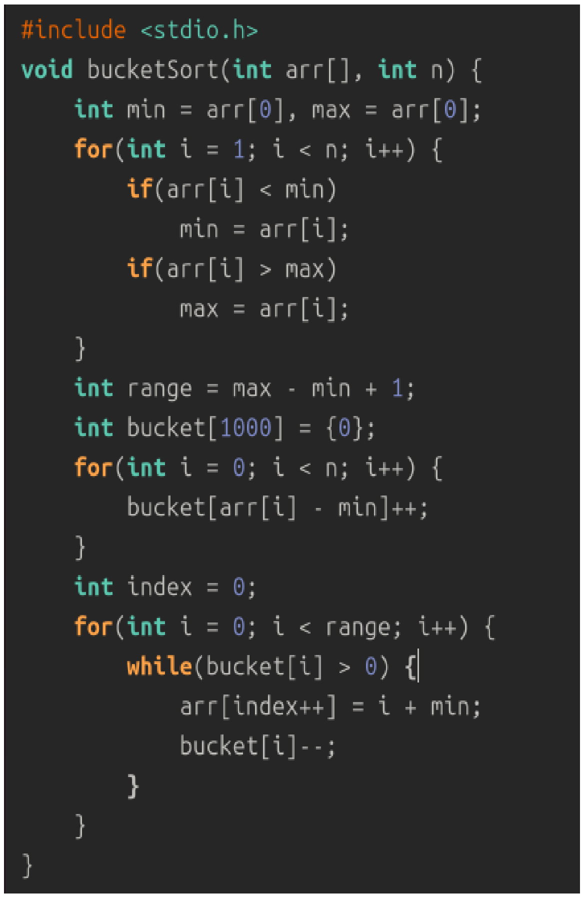
int temp → 4 bytes

Sorting is done in the same array.

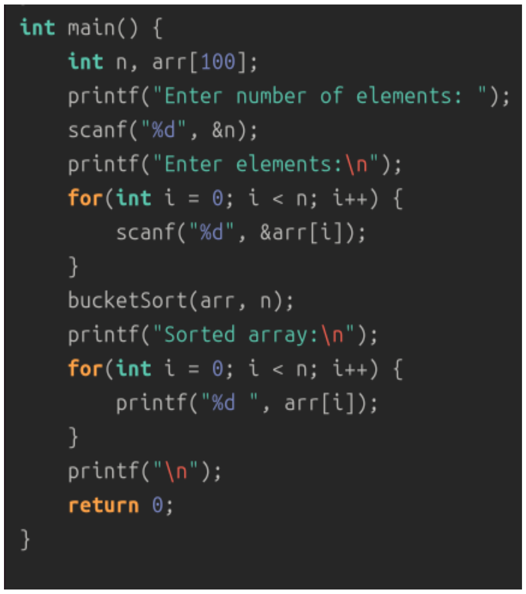
Space Complexity = O(1)

5.BUCKET SORT

CODE:

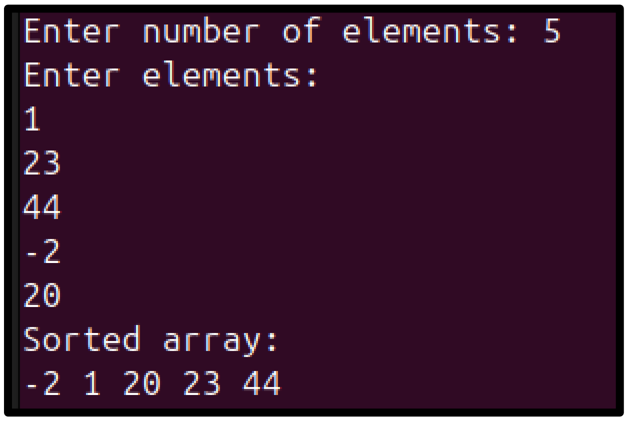


```c
#include <stdio.h>
void bucketSort(int arr[], int n) {
    int min = arr[0], max = arr[0];
    for(int i = 1; i < n; i++) {
        if(arr[i] < min)
            min = arr[i];
        if(arr[i] > max)
            max = arr[i];
    }
    int range = max - min + 1;
    int bucket[1000] = {0};
    for(int i = 0; i < n; i++) {
        bucket[arr[i] - min]++;
    }
    int index = 0;
    for(int i = 0; i < range; i++) {
        while(bucket[i] > 0) {
            arr[index++] = i + min;
            bucket[i]--;
        }
    }
}
```



```c
int main() {
    int n, arr[100];
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    bucketSort(arr, n);
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:



```
Enter number of elements: 5
Enter elements:
1
23
44
-2
20
Sorted array:
-2 1 20 23 44
```

Time Complexity

Each element is placed into a bucket once.

For each element, its bucket index is calculated.

Elements from all buckets are copied back to the array.

Total operations ≈ n + k.

Time Complexity = O(n + k)

Where:

n = number of elements

k = number of buckets (or value range

Space Complexity

int bucket[k] → depends on value range

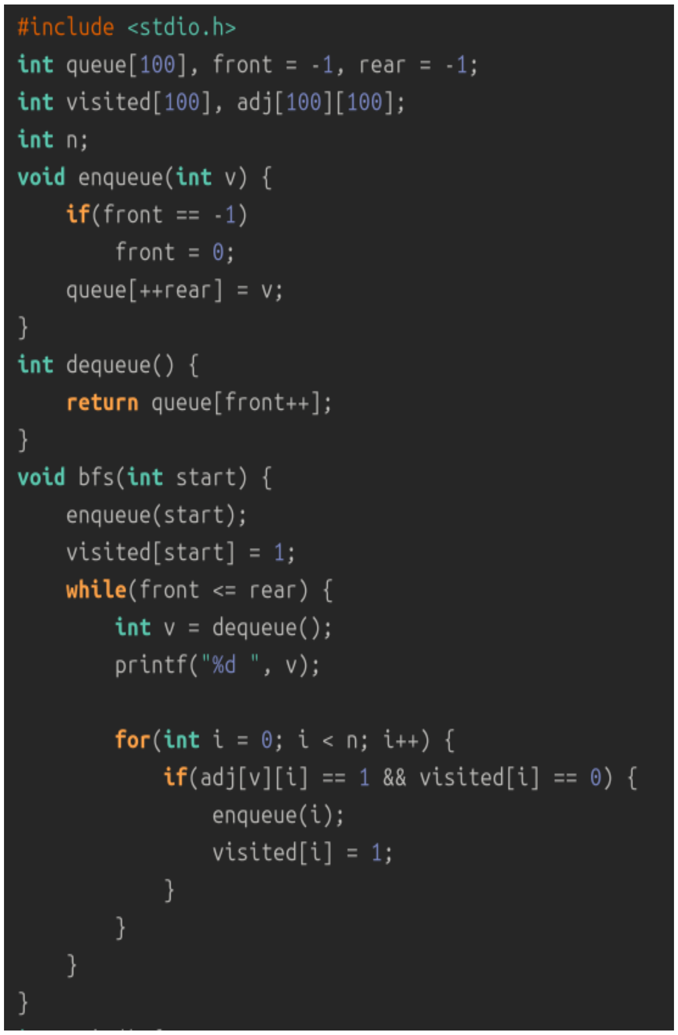
Auxiliary variables → constant space

Total space depends on k.

Space Complexity = O(k)

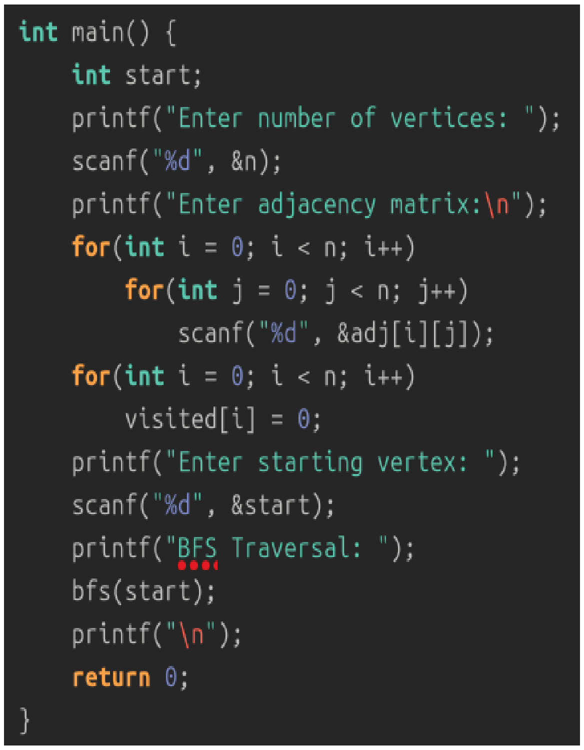
6.BFS:

CODE:



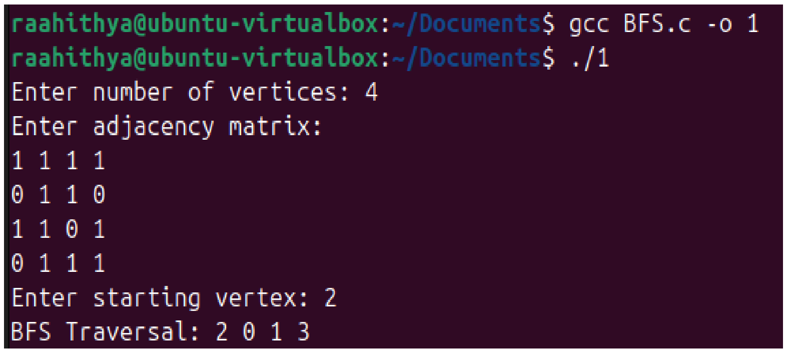
```c
#include <stdio.h>
int queue[100], front = -1, rear = -1;
int visited[100], adj[100][100];
int n;
void enqueue(int v) {
    if(front == -1)
        front = 0;
    queue[++rear] = v;
}
int dequeue() {
    return queue[front++];
}
void bfs(int start) {
    enqueue(start);
    visited[start] = 1;
    while(front <= rear) {
        int v = dequeue();
        printf("%d ", v);

        for(int i = 0; i < n; i++) {
            if(adj[v][i] == 1 && visited[i] == 0) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}
```



```c
int main() {
    int start;
    printf("Enter number of vertices: ");
    scanf("%d", &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]);
    for(int i = 0; i < n; i++)
        visited[i] = 0;
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("BFS Traversal: ");
    bfs(start);
    printf("\n");
    return 0;
}
```

OUTPUT:



```
raahithya@ubuntu-virtualbox:~/Documents$ gcc BFS.c -o 1
raahithya@ubuntu-virtualbox:~/Documents$ ./1
Enter number of vertices: 4
Enter adjacency matrix:
1 1 1 1
0 1 1 0
1 1 0 1
0 1 1 1
Enter starting vertex: 2
BFS Traversal: 2 0 1 3
```

Time Complexity:

Each vertex is visited once.

For each vertex, all n vertices are checked.

Total operations ≈ n * n.

Time Complexity = O(n^2)

Space complexity:

int graph[MAX][MAX] → fixed size

int queue[MAX] → fixed size

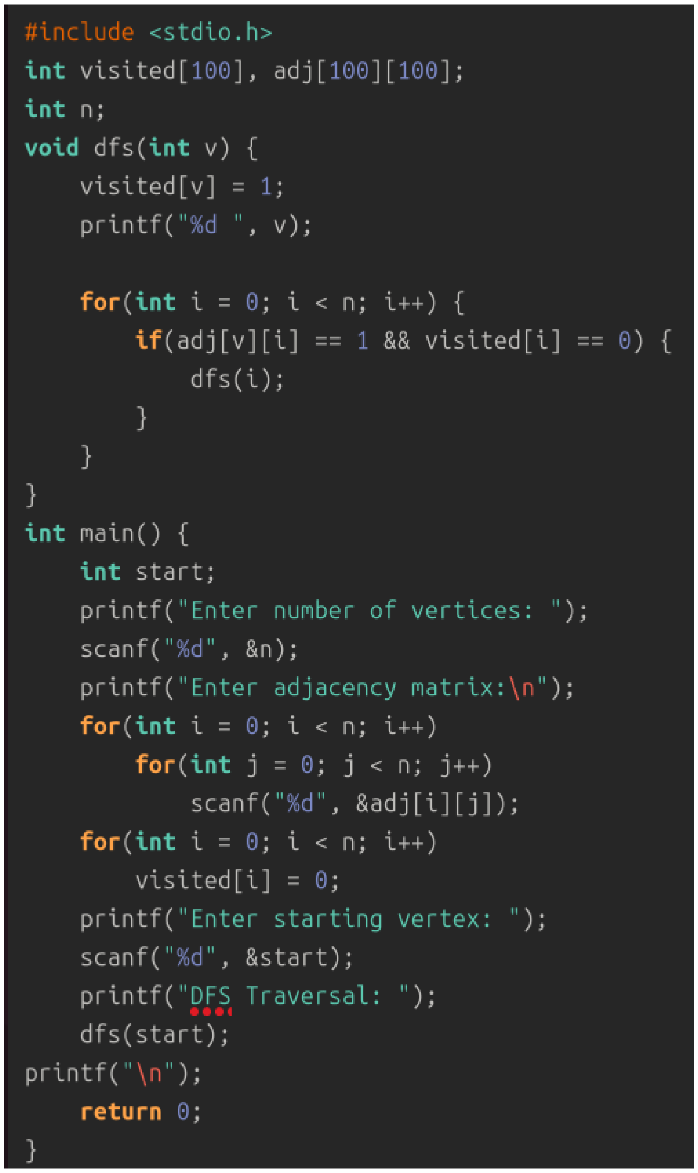
int visited[MAX] → fixed size

Total space is constant.

Space Complexity = O(1)

7.DFS:

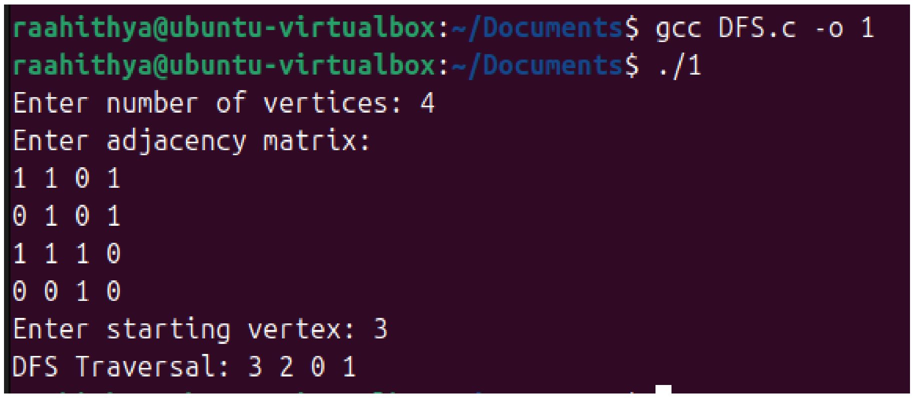
CODE:



```
#include <stdio.h>
int visited[100], adj[100][100];
int n;
void dfs(int v) {
    visited[v] = 1;
    printf("%d ", v);

    for(int i = 0; i < n; i++) {
        if(adj[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}
int main() {
    int start;
    printf("Enter number of vertices: ");
    scanf("%d", &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]);
    for(int i = 0; i < n; i++)
        visited[i] = 0;
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("DFS Traversal: ");
    dfs(start);
printf("\n");
    return 0;
}
```

OUTPUT:



```
raahithya@ubuntu-virtualbox:~/Documents$ gcc DFS.c -o 1
raahithya@ubuntu-virtualbox:~/Documents$ ./1
Enter number of vertices: 4
Enter adjacency matrix:
1 1 0 1
0 1 0 1
1 1 1 0
0 0 1 0
Enter starting vertex: 3
DFS Traversal: 3 2 0 1
```

Time Complexity:

Each vertex is visited once.

For each vertex, all n vertices are checked.

Time Complexity = O(n^2)

Space Complexity:

int visited[MAX] → fixed size

Recursive calls up to n.

Space depends on n.

Space Complexity = O(n)