NAME: CHANTATI SAI PURNISHA MAHI

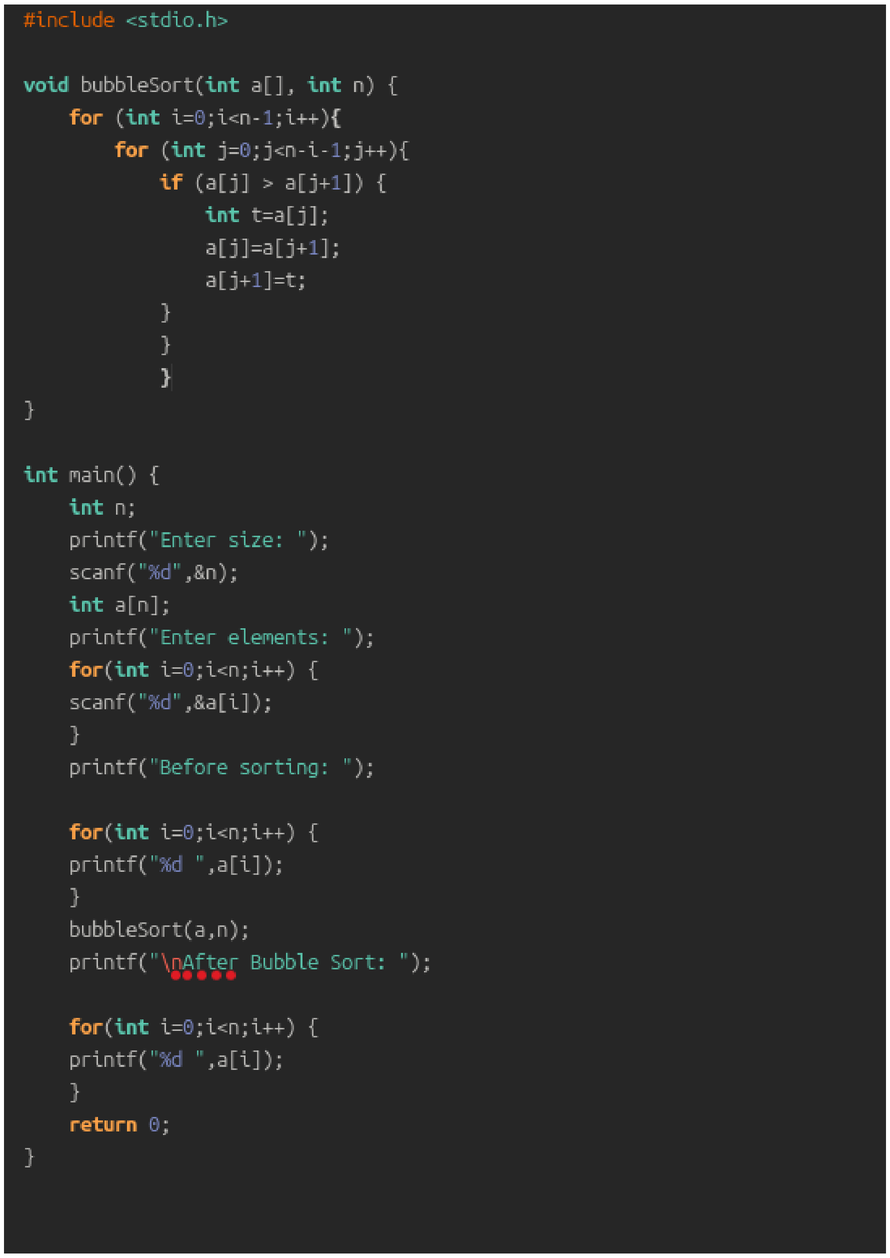
ROLL NO:CH.SC.U4CSE24156

(Design and Analysis Algorithms)

WEEK2:-

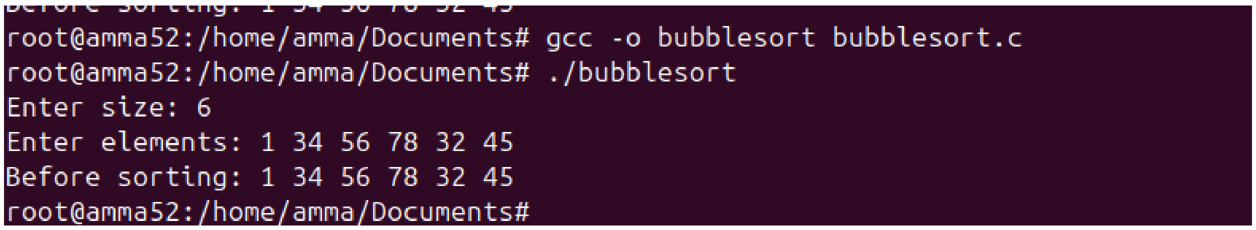
1Q) BUBBLE SORT

CODE:



```c
#include <stdio.h>

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

int main() {
    int n;
    printf("Enter size: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements: ");
    for(int i=0;i<n;i++) {
    scanf("%d",&a[i]);
    }
    printf("Before sorting: ");

    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    bubbleSort(a,n);
    printf("\nAfter Bubble Sort: ");

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

OUTPUT:



```
root@amma52:/home/amma/Documents# gcc -o bubblesort bubblesort.c
root@amma52:/home/amma/Documents# ./bubblesort
Enter size: 6
Enter elements: 1 34 56 78 32 45
Before sorting: 1 34 56 78 32 45
root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(n^2)

JUSTIFICATION:-

Outer loop runs n times.

Inner loop runs up to n times.

Total operations ≈ n × n.

SPACE COMPLEXITY:- O(1)

JUSTIFICATION:-

int n → 4 bytes

int i → 4 bytes

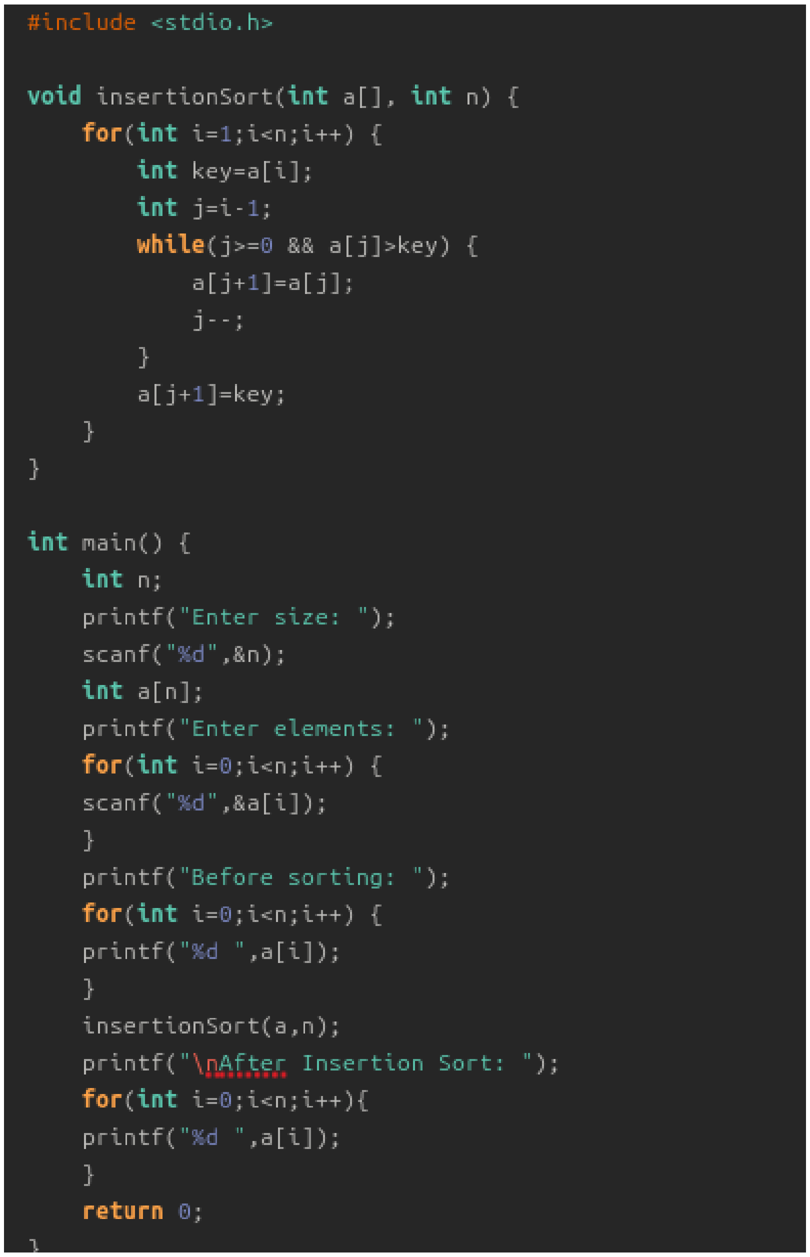
int j → 4 bytes

int temp → 4 bytes

Only constant extra space is used.

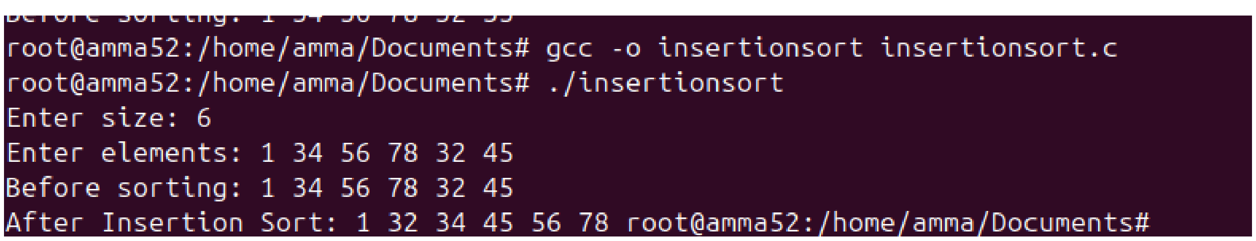
2Q) INSERTION SORT

CODE:



```c
#include <stdio.h>

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

int main() {
    int n;
    printf("Enter size: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements: ");
    for(int i=0;i<n;i++) {
    scanf("%d",&a[i]);
    }
    printf("Before sorting: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    insertionSort(a,n);
    printf("\nAfter Insertion Sort: ");
    for(int i=0;i<n;i++){
    printf("%d ",a[i]);
    }
    return 0;
}
```

OUTPUT:



```
Before sorting: 1 34 56 78 32 45
root@amma52:/home/amma/Documents# gcc -o insertionsort insertionsort.c
root@amma52:/home/amma/Documents# ./insertionsort
Enter size: 6
Enter elements: 1 34 56 78 32 45
Before sorting: 1 34 56 78 32 45
After Insertion Sort: 1 32 34 45 56 78 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(n^2)

JUSTIFICATION:-

Outer loop runs n times.

Inner while loop runs up to n times.

Total operations ≈ n × n.

SPACE COMPLEXITY:- O(1)

JUSTIFICATION:-

int n → 4 bytes

int i → 4 bytes

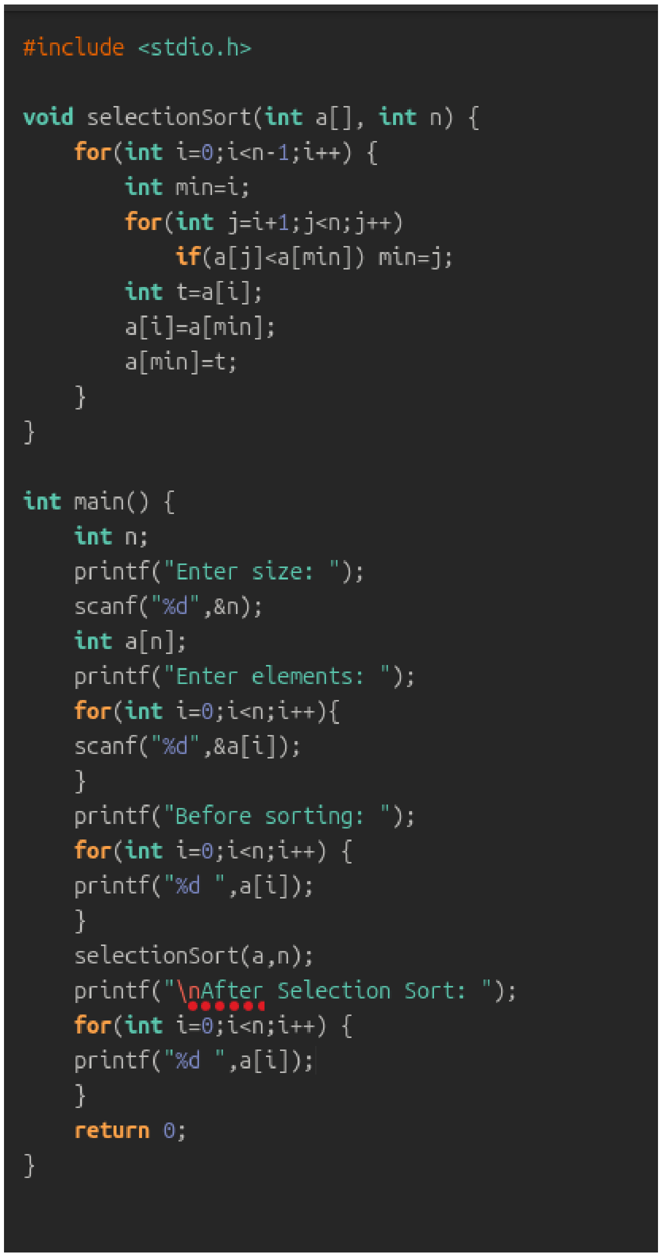
int j → 4 bytes

int key → 4 bytes

Only constant extra space is used.

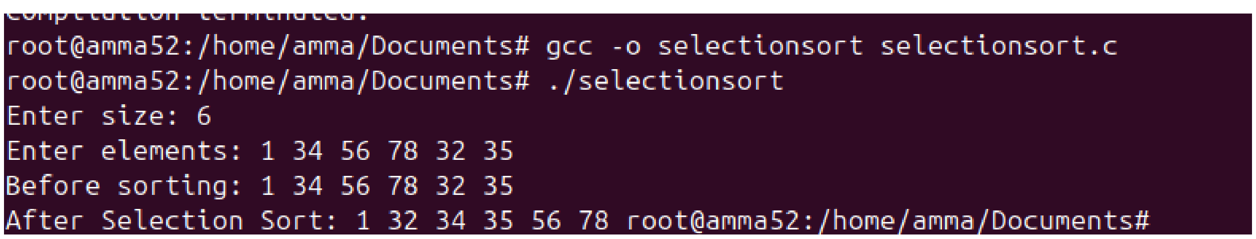
3Q) SELECTION SORT

CODE:



```c
#include <stdio.h>

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

int main() {
    int n;
    printf("Enter size: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements: ");
    for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
    }
    printf("Before sorting: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    selectionSort(a,n);
    printf("\nAfter Selection Sort: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    return 0;
}
```

OUTPUT:



```
root@amma52:/home/amma/Documents# gcc -o selectionsort selectionsort.c
root@amma52:/home/amma/Documents# ./selectionsort
Enter size: 6
Enter elements: 1 34 56 78 32 35
Before sorting: 1 34 56 78 32 35
After Selection Sort: 1 32 34 35 56 78 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(n^2)

JUSTIFICATION:-

Outer loop runs n times.

Inner while loop runs up to n times.

Total operations ≈ n × n.

SPACE COMPLEXITY:- O(1)

JUSTIFICATION:-

int n → 4 bytes

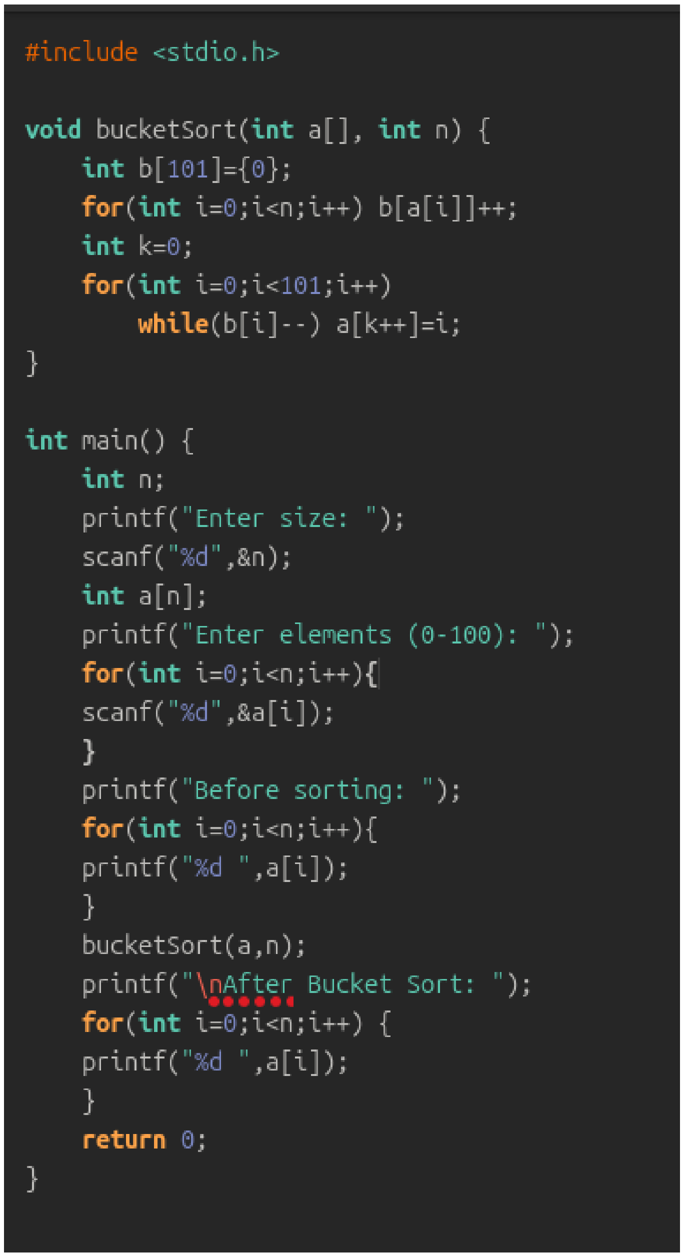
int i → 4 bytes

int j → 4 bytes

int key → 4 bytes

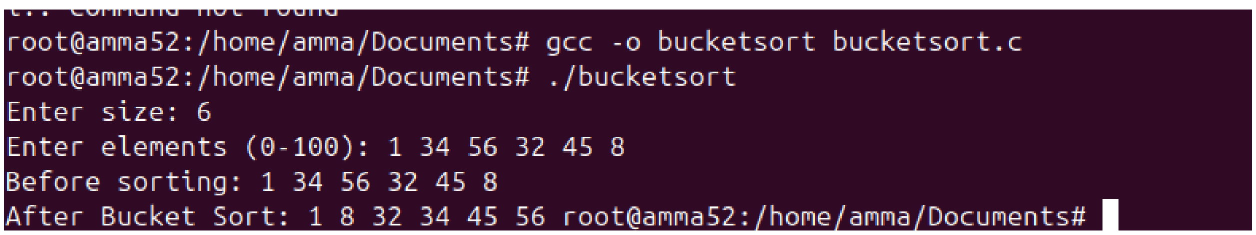
Only constant extra space is used.

4Q) BUCKET SORT
CODE:



```c
#include <stdio.h>

void bucketSort(int a[], int n) {
    int b[101]={0};
    for(int i=0;i<n;i++) b[a[i]]++;
    int k=0;
    for(int i=0;i<101;i++)
        while(b[i]--) a[k++]=i;
}

int main() {
    int n;
    printf("Enter size: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements (0-100): ");
    for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
    }
    printf("Before sorting: ");
    for(int i=0;i<n;i++){
    printf("%d ",a[i]);
    }
    bucketSort(a,n);
    printf("\nAfter Bucket Sort: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    return 0;
}
```

OUTPUT:



```
t:: command not found
root@amma52:/home/amma/Documents# gcc -o bucketsort bucketsort.c
root@amma52:/home/amma/Documents# ./bucketsort
Enter size: 6
Enter elements (0-100): 1 34 56 32 45 8
Before sorting: 1 34 56 32 45 8
After Bucket Sort: 1 8 32 34 45 56 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(n + k)

JUSTIFICATION:-

Elements are distributed into k buckets.

Each bucket is sorted individually.

Average case operations ≈ n + k.

SPACE COMPLEXITY:- O(n + k)

JUSTIFICATION:-

Buckets array → extra space

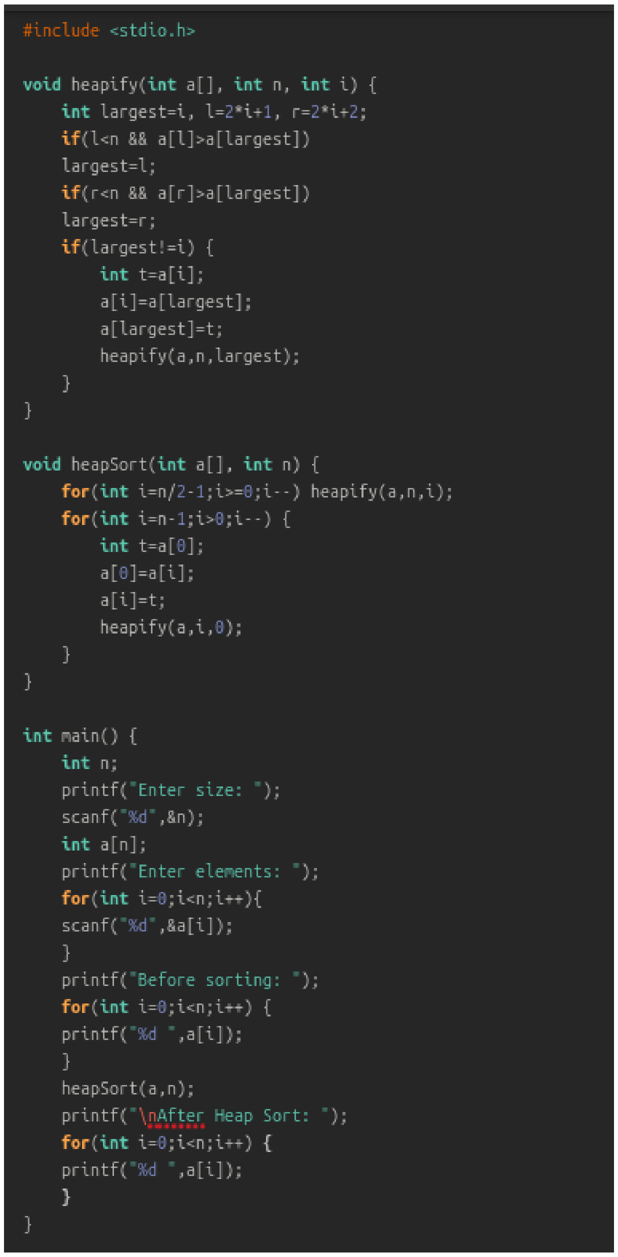
int n → 4 bytes

int i → 4 bytes

Additional space for buckets is required.

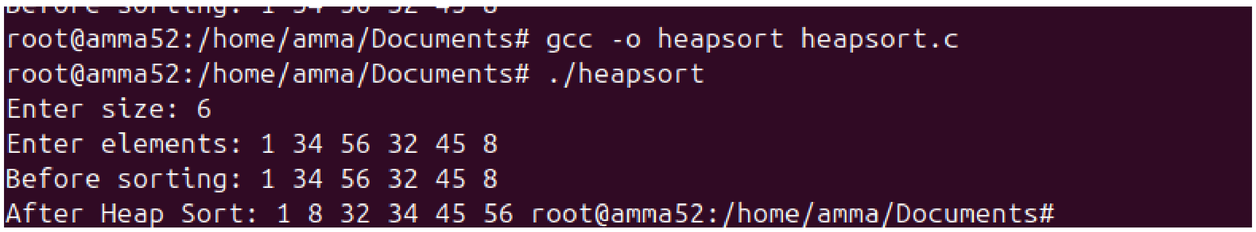
5Q) HEAP SORT

CODE:



```c
#include <stdio.h>

void heapify(int a[], int n, int i) {
    int largest=i, l=2*i+1, r=2*i+2;
    if(l<n && a[l]>a[largest])
    largest=l;
    if(r<n && a[r]>a[largest])
    largest=r;
    if(largest!=i) {
        int t=a[i];
        a[i]=a[largest];
        a[largest]=t;
        heapify(a,n,largest);
    }
}

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

int main() {
    int n;
    printf("Enter size: ");
    scanf("%d",&n);
    int a[n];
    printf("Enter elements: ");
    for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
    }
    printf("Before sorting: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
    heapSort(a,n);
    printf("\nAfter Heap Sort: ");
    for(int i=0;i<n;i++) {
    printf("%d ",a[i]);
    }
}
```

OUTPUT:



```
root@amma52:/home/amma/Documents# gcc -o heapsort heapsort.c
root@amma52:/home/amma/Documents# ./heapsort
Enter size: 6
Enter elements: 1 34 56 32 45 8
Before sorting: 1 34 56 32 45 8
After Heap Sort: 1 8 32 34 45 56 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(n log n)

JUSTIFICATION:-

Building heap takes O(n).

Heapify operation runs log n for each element.

Total operations ≈ n log n.

SPACE COMPLEXITY:-O(1)

JUSTIFICATION:-

int n → 4 bytes

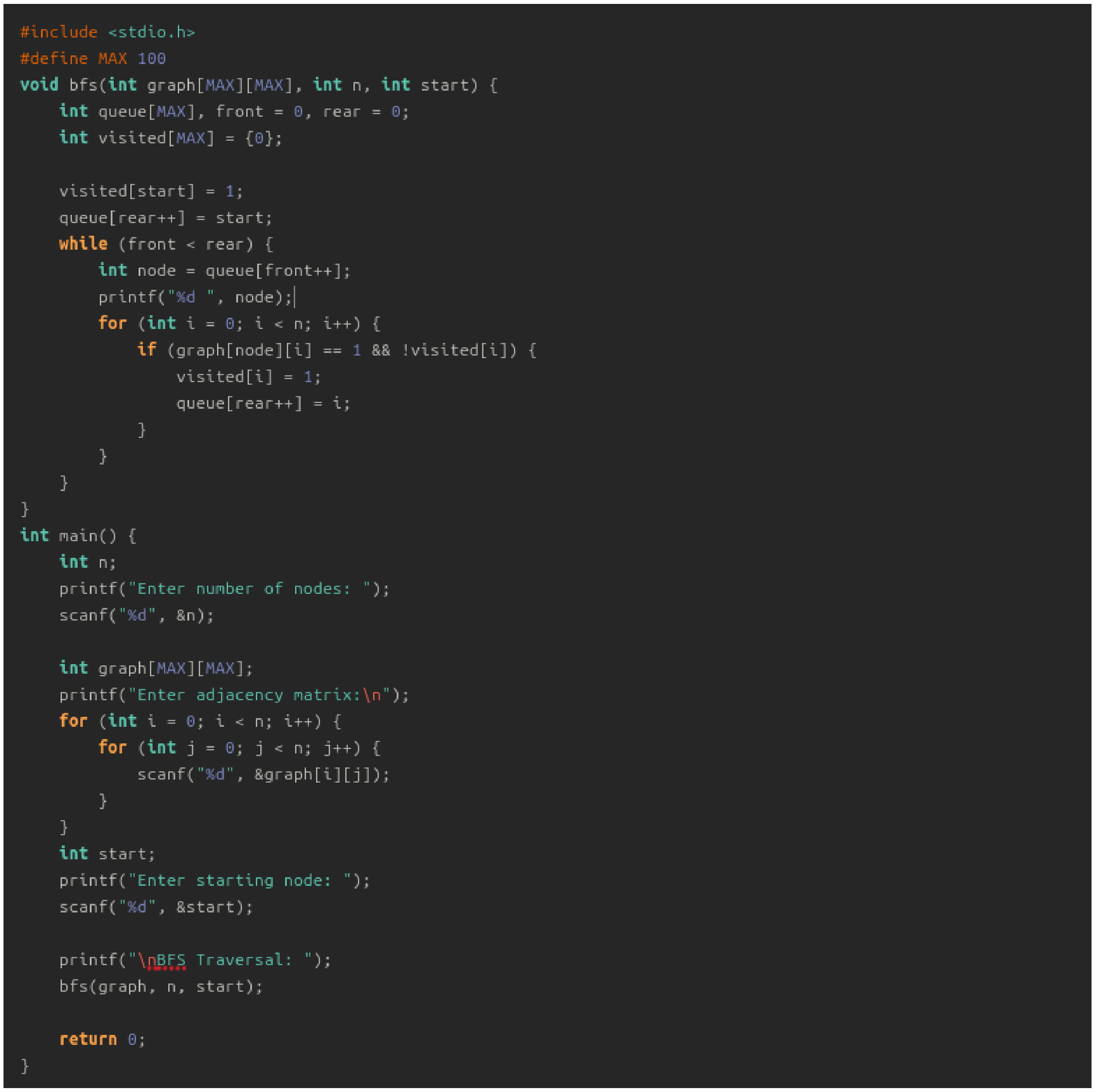
int i → 4 bytes

int temp → 4 bytes

Sorting is done in-place.

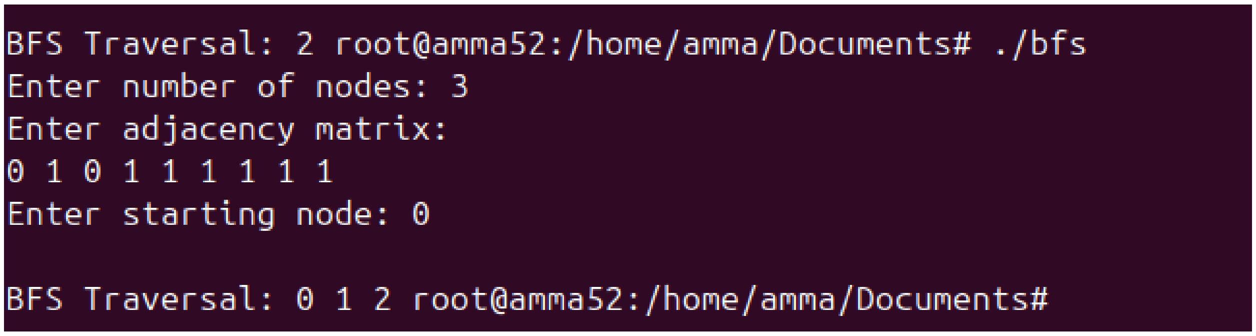
6Q) BFS

CODE:



```c
#include <stdio.h>
#define MAX 100
void bfs(int graph[MAX][MAX], int n, int start) {
    int queue[MAX], front = 0, rear = 0;
    int visited[MAX] = {0};

    visited[start] = 1;
    queue[rear++] = start;
    while (front < rear) {
        int node = queue[front++];
        printf("%d ", node);
        for (int i = 0; i < n; i++) {
            if (graph[node][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}
int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    int graph[MAX][MAX];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    int start;
    printf("Enter starting node: ");
    scanf("%d", &start);

    printf("\nBFS Traversal: ");
    bfs(graph, n, start);

    return 0;
}
```

OUTPUT:



```
BFS Traversal: 2 root@amma52:/home/amma/Documents# ./bfs
Enter number of nodes: 3
Enter adjacency matrix:
0 1 0 1 1 1 1 1 1
Enter starting node: 0

BFS Traversal: 0 1 2 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(V + E)

JUSTIFICATION:-

Each vertex is visited once → O(V)

Each edge is explored once → O(E)

Total operations ≈ V + E

SPACE COMPLEXITY:- O(V)

JUSTIFICATION:-

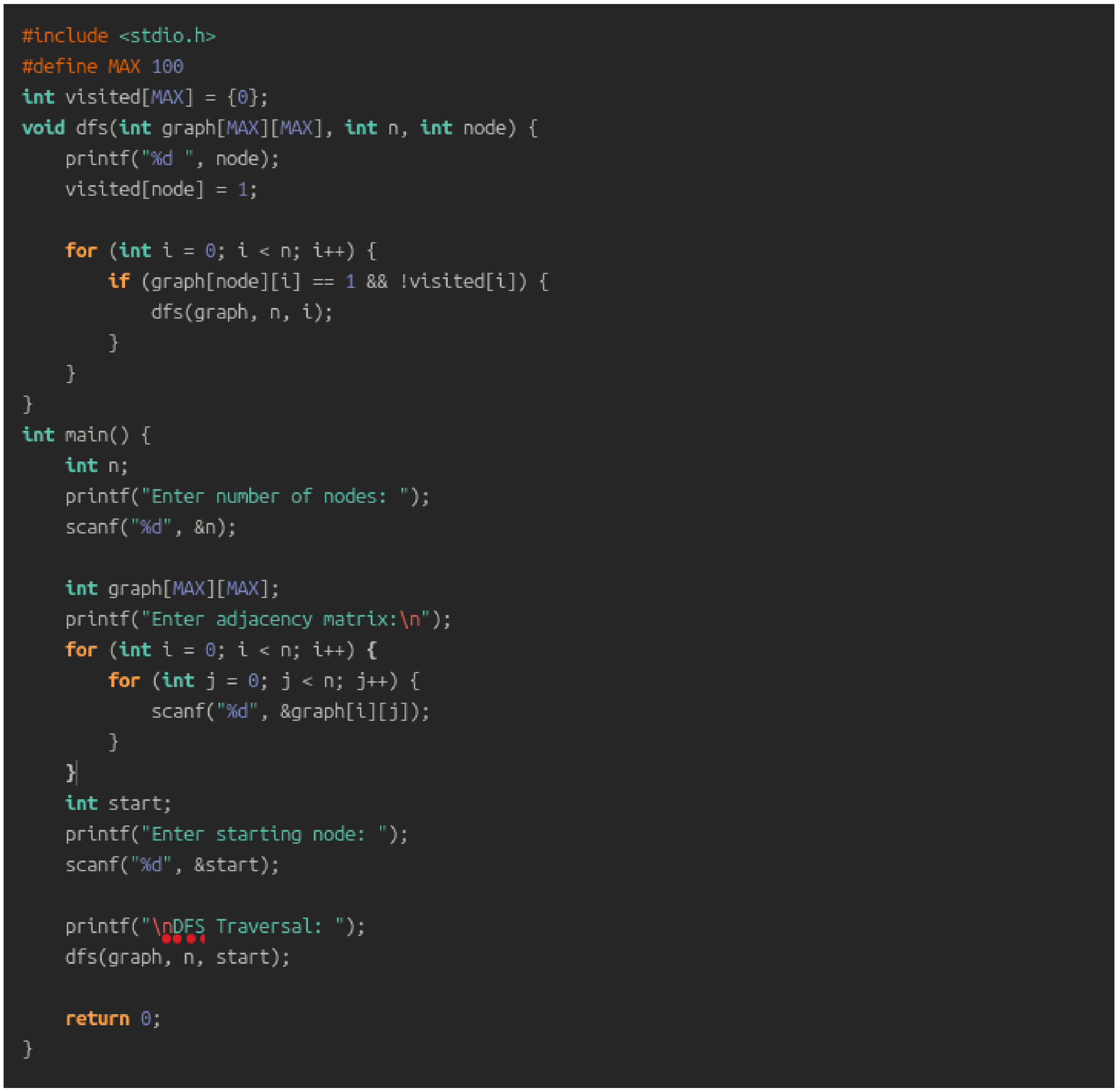
Queue can store up to V vertices

Visited array of size V

Only linear extra space is used

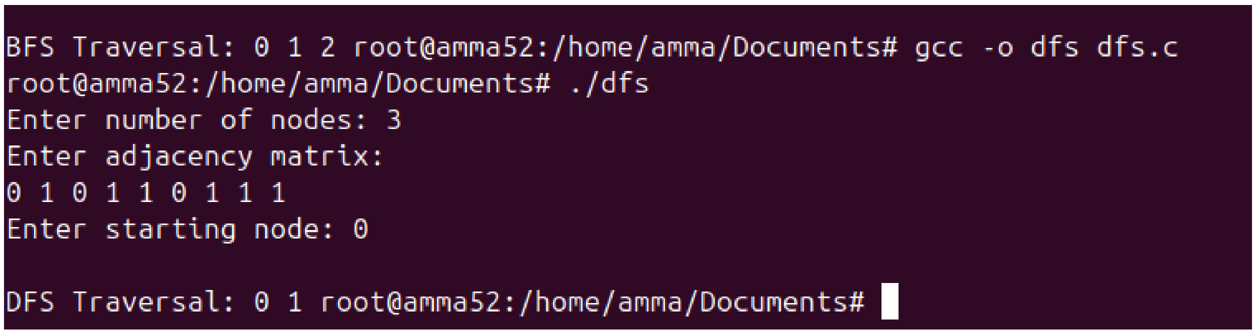
7Q) DFS

CODE:



```c
#include <stdio.h>
#define MAX 100
int visited[MAX] = {0};
void dfs(int graph[MAX][MAX], int n, int node) {
    printf("%d ", node);
    visited[node] = 1;

    for (int i = 0; i < n; i++) {
        if (graph[node][i] == 1 && !visited[i]) {
            dfs(graph, n, i);
        }
    }
}
int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    int graph[MAX][MAX];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    int start;
    printf("Enter starting node: ");
    scanf("%d", &start);

    printf("\nDFS Traversal: ");
    dfs(graph, n, start);

    return 0;
}
```

OUTPUT:



```
BFS Traversal: 0 1 2 root@amma52:/home/amma/Documents# gcc -o dfs dfs.c
root@amma52:/home/amma/Documents# ./dfs
Enter number of nodes: 3
Enter adjacency matrix:
0 1 0 1 1 0 1 1 1
Enter starting node: 0

DFS Traversal: 0 1 root@amma52:/home/amma/Documents#
```

TIME COMPLEXITY:- O(V + E)

JUSTIFICATION:-

Each vertex visited once → V

Each edge visited once → E

SPACE COMPLEXITY:- O(V)

JUSTIFICATION:-

Recursion stack → V

Visited array → V