Design Analysis and Algorithm – Lab Work

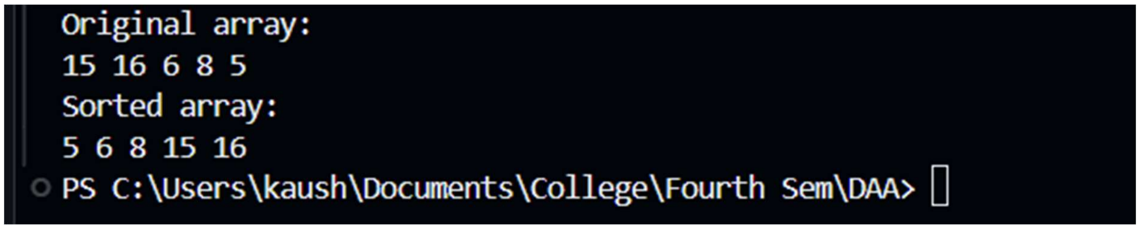
Week 2

Question 1: Write a program to implement bubble sort

Code:

```c
#include <stdio.h>

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

int main() {
    int i, j;
    int a[5] = {15, 16, 6, 8, 5};
    int n = 5;
    int temp;

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

    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

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

    return 0;
}
```

Output:



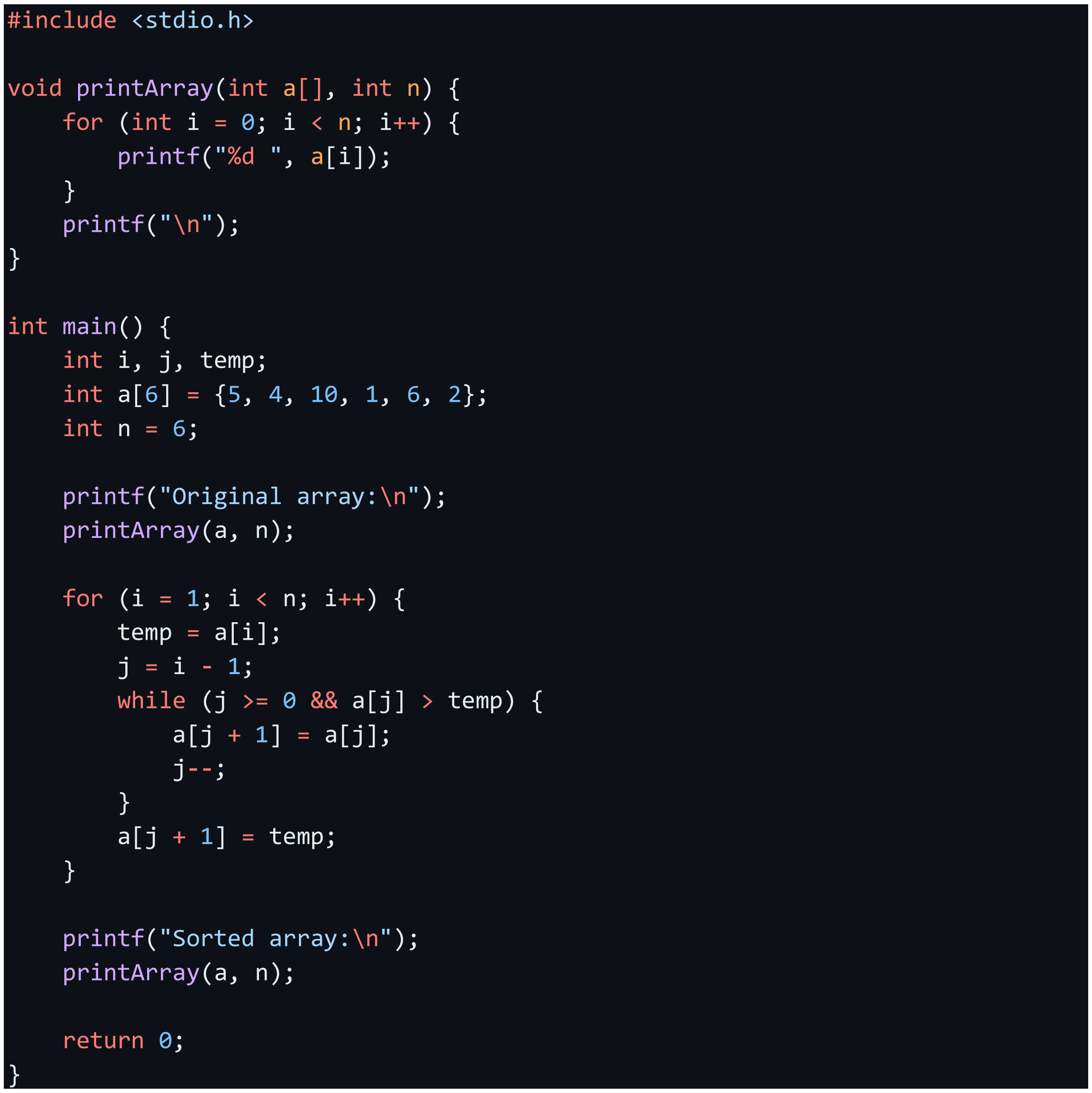
```
Original array:
15 16 6 8 5
Sorted array:
5 6 8 15 16
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(1)

Time Complexity: O(n^2)

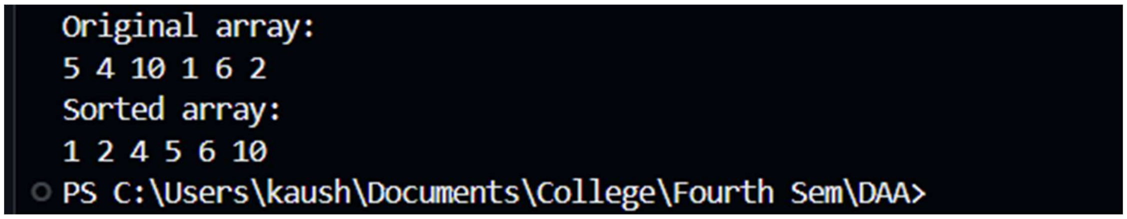
Question 2: Write a program to implement insertion sort.

Code:



```c
#include <stdio.h>

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

int main() {
    int i, j, temp;
    int a[6] = {5, 4, 10, 1, 6, 2};
    int n = 6;

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

    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;
        while (j >= 0 && a[j] > temp) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = temp;
    }

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

    return 0;
}
```

Output:



```
Original array:
5 4 10 1 6 2
Sorted array:
1 2 4 5 6 10
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(1)

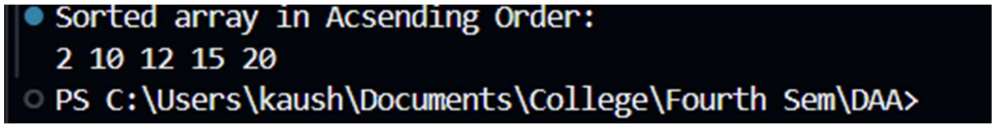
Time Complexity: O(n^2)

Question 3: Write a program to implement selection sort.

Code:

```c
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void selectionSort(int array[], int size) {
    for (int step = 0; step < size - 1; step++) {
        int min_idx = step;
        for (int i = step + 1; i < size; i++) {
            if (array[i] < array[min_idx])
                min_idx = i;
            }
        swap(&array[min_idx], &array[step]);
    }
}
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        printf("%d ", array[i]);
    }
    printf("\n");
}
int main() {
    int data[] = {20, 12, 10, 15, 2};
    int size = sizeof(data) / sizeof(data[0]);
    selectionSort(data, size);
    printf("Sorted array in Acsending Order:\n");
    printArray(data, size);
}
```

Output:



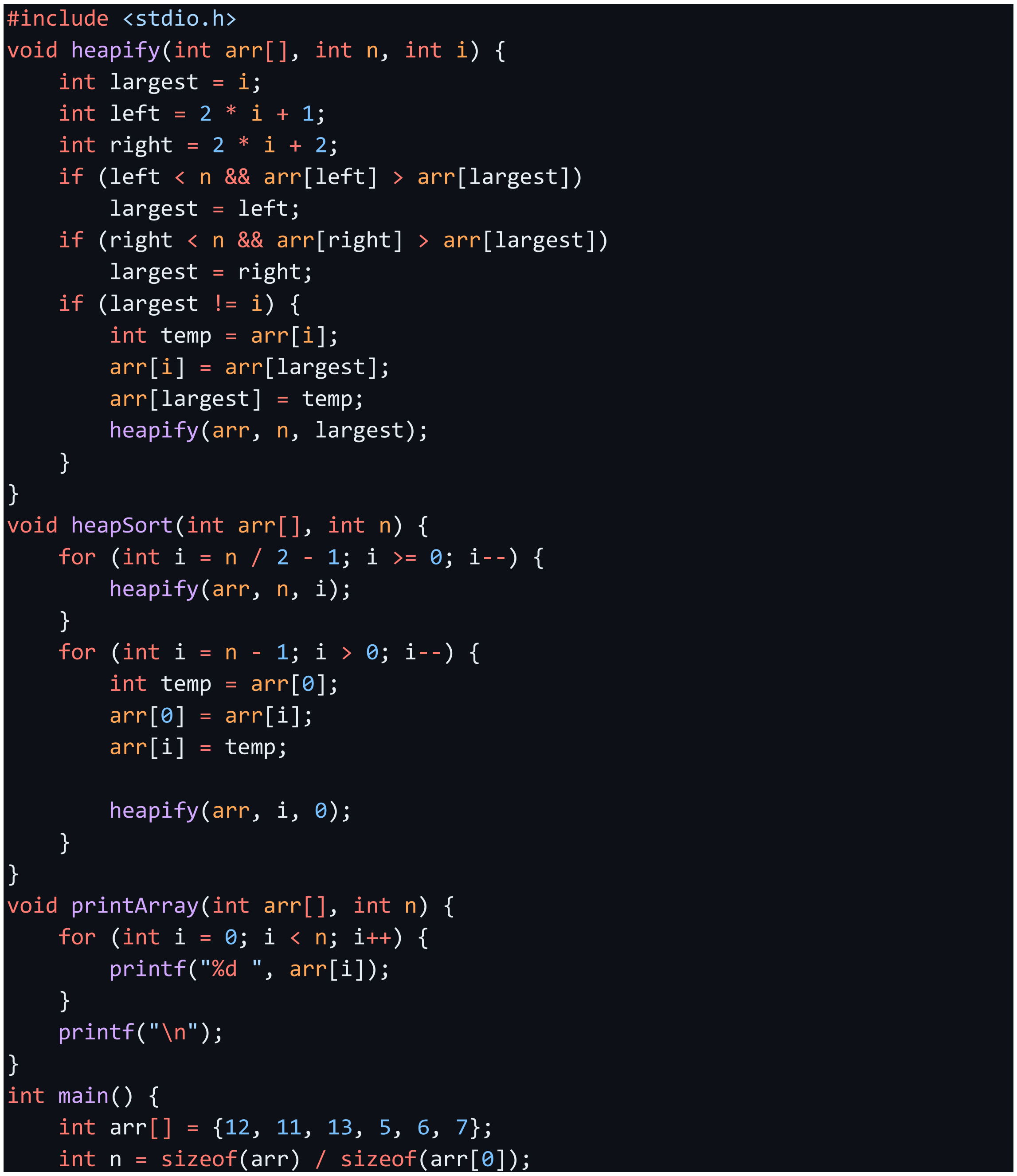
```
Sorted array in Acsending Order:
2 10 12 15 20
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(1)

Time Complexity: O(n^2)

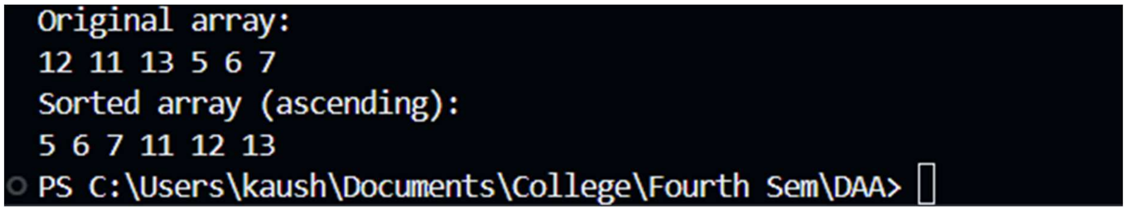
Question 4: Write a program to implement heap sort.

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);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array:\n");
    printArray(arr, n);
    heapSort(arr, n);
    printf("Sorted array (ascending):\n");
    printArray(arr, n);
    return 0;
}
```

Output:



```
Original array:
12 11 13 5 6 7
Sorted array (ascending):
5 6 7 11 12 13
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

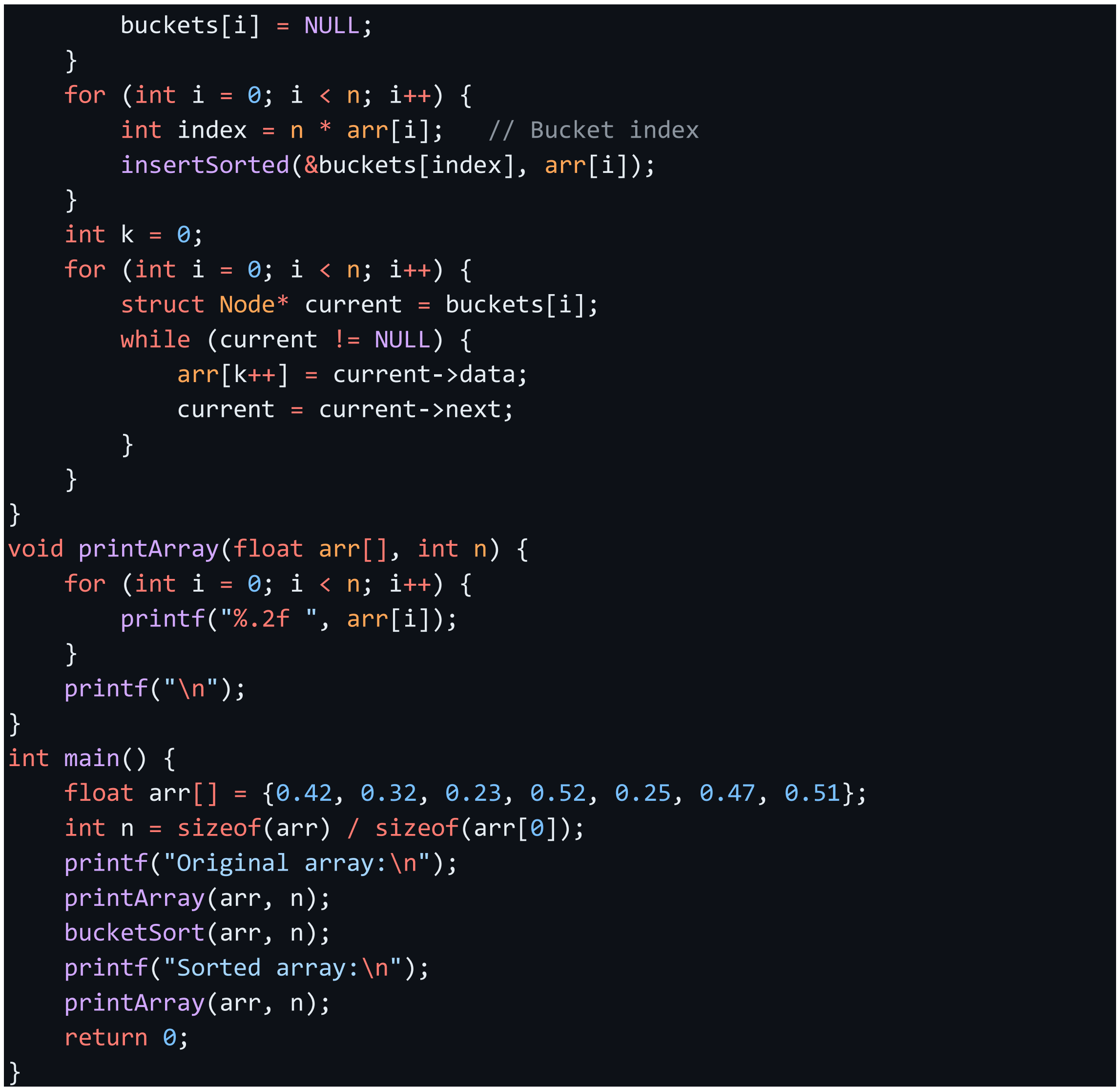
Space Complexity: O(1)

Time Complexity: O(n.log(n))

Question 5: Write a program to implement bucket sort.

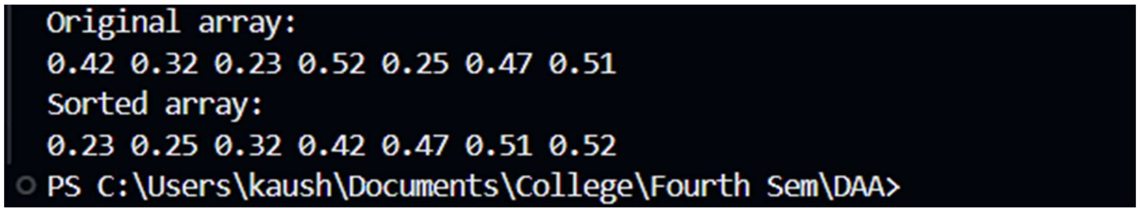
Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    float data;
    struct Node* next;
};
void insertSorted(struct Node** head, float value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL || (*head)->data >= value) {
        newNode->next = *head;
        *head = newNode;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL && current->next->data < value) {
        current = current->next;
    }
    newNode->next = current->next;
    current->next = newNode;
}
void bucketSort(float arr[], int n) {
    struct Node* buckets[n];
    for (int i = 0; i < n; i++) {
```



```c
        buckets[i] = NULL;
    }
    for (int i = 0; i < n; i++) {
        int index = n * arr[i];    // Bucket index
        insertSorted(&buckets[index], arr[i]);
    }
    int k = 0;
    for (int i = 0; i < n; i++) {
        struct Node* current = buckets[i];
        while (current != NULL) {
            arr[k++] = current->data;
            current = current->next;
        }
    }
}
void printArray(float arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%.2f ", arr[i]);
    }
    printf("\n");
}
int main() {
    float arr[] = {0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51};
    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:



```
Original array:
0.42 0.32 0.23 0.52 0.25 0.47 0.51
Sorted array:
0.23 0.25 0.32 0.42 0.47 0.51 0.52
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(n+k)

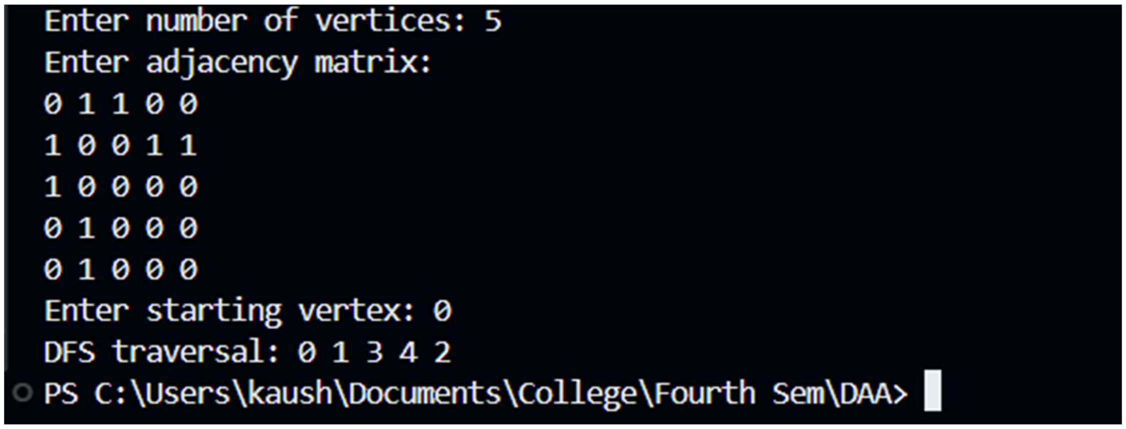
Time Complexity: $O(n^2)$

Question 6: Write a program to implement Breadth First Search (BFS) in a Graph

Code:

```c
#include <stdio.h>
#define MAX 10
int graph[MAX][MAX];
int visited[MAX];
int n;

void DFS(int v) {
    printf("%d ", v);
    visited[v] = 1;
    for (int i = 0; i < n; i++) {
        if (graph[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", &graph[i][j]);
        }
    }
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("DFS traversal: ");
    DFS(start);
    return 0;
}
```

Output:



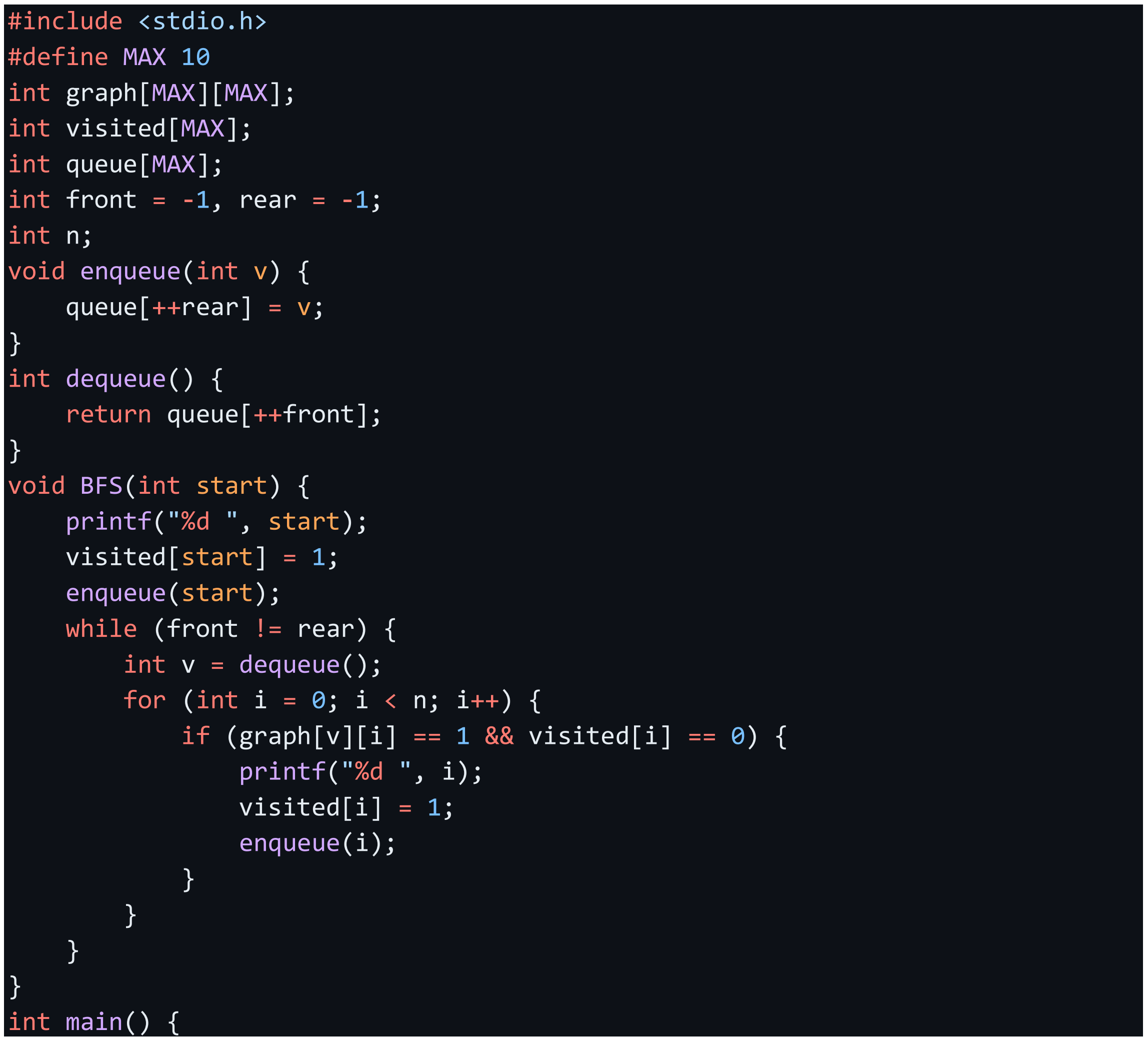
```
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 0 0
1 0 0 1 1
1 0 0 0 0
0 1 0 0 0
0 1 0 0 0
Enter starting vertex: 0
DFS traversal: 0 1 3 4 2
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(V)

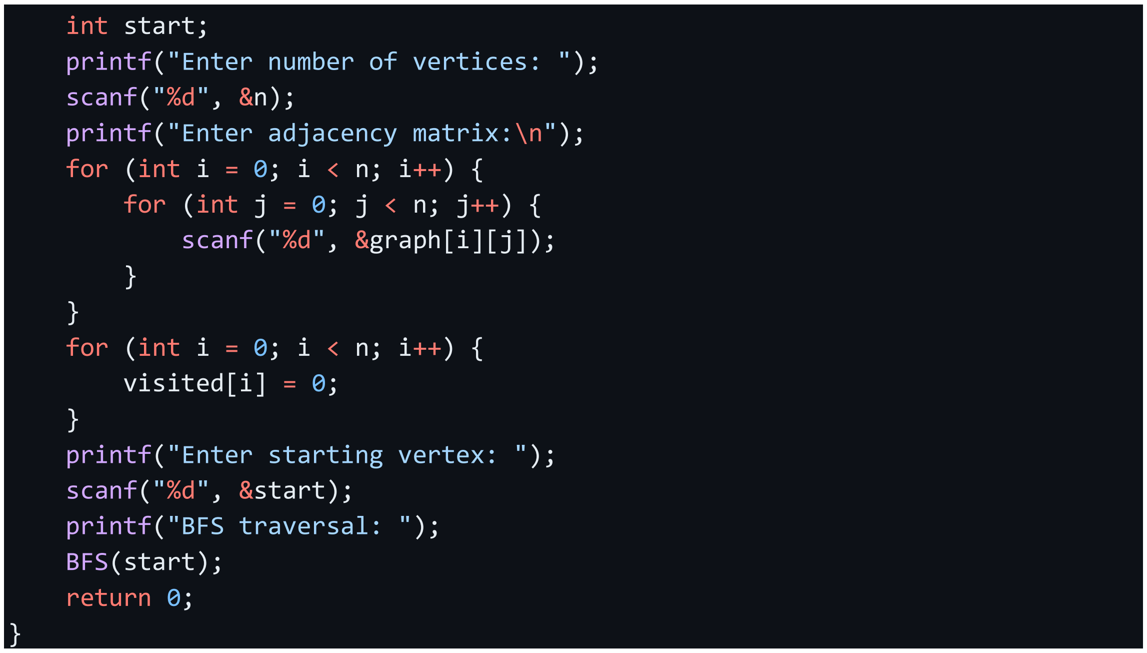
Time Complexity: O(V^2)

Question 7: Write a program to implement Deapth First Search (DFS) in a Graph

Code:

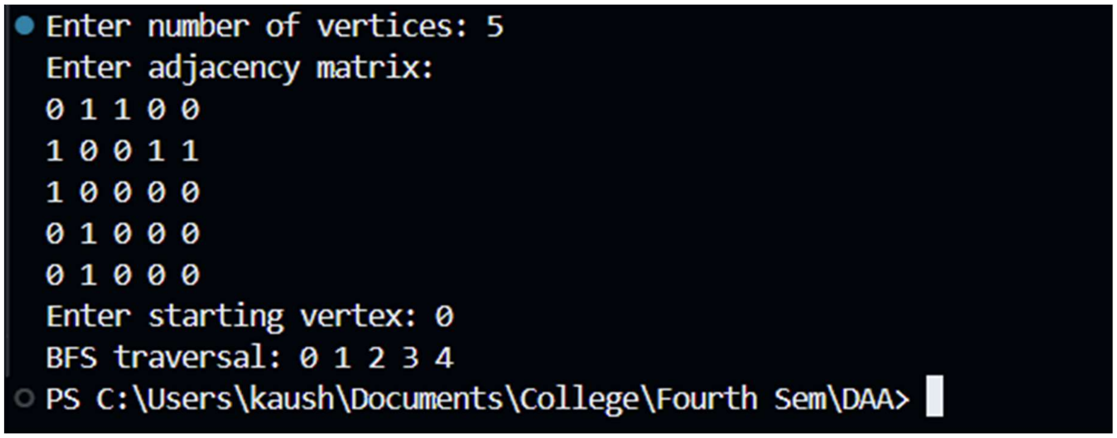


```c
#include <stdio.h>
#define MAX 10
int graph[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = -1, rear = -1;
int n;
void enqueue(int v) {
    queue[++rear] = v;
}
int dequeue() {
    return queue[++front];
}
void BFS(int start) {
    printf("%d ", start);
    visited[start] = 1;
    enqueue(start);
    while (front != rear) {
        int v = dequeue();
        for (int i = 0; i < n; i++) {
            if (graph[v][i] == 1 && visited[i] == 0) {
                printf("%d ", i);
                visited[i] = 1;
                enqueue(i);
            }
        }
    }
}
int main() {
```



```c
    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", &graph[i][j]);
        }
    }
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("BFS traversal: ");
    BFS(start);
    return 0;
}
```

Output:



```
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 0 0
1 0 0 1 1
1 0 0 0 0
0 1 0 0 0
0 1 0 0 0
Enter starting vertex: 0
BFS traversal: 0 1 2 3 4
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(V)

Time Complexity: O(V^2)