Design and Analysis of Algorithms
Sorting

Name: J Darunkumar

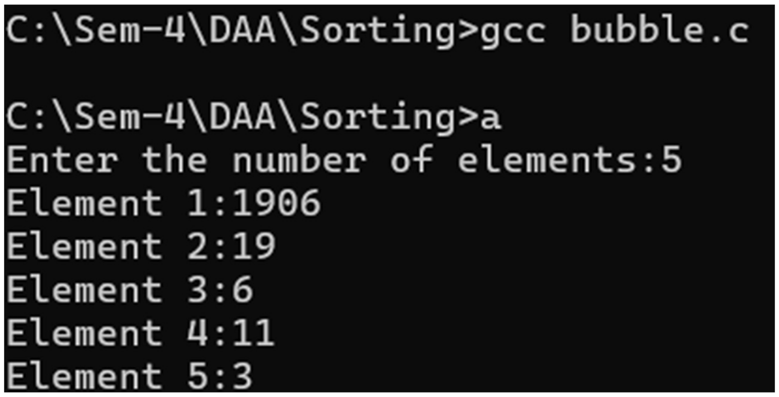
Roll-No: CH.SC.U4CSE24111

1. Write a program to implement bubble sort.

   Code:

```c
#include <stdio.h>

void bubbleSort(int arr[],int n){
    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;
            }
        }
    }
}

int main() {
    int n,i;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++){
        printf("Element %d:",i+1);
        scanf("%d",&arr[i]);
    }

    bubbleSort(arr, n);

    printf("Bubble Sorted array: ");
    for(i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Output:



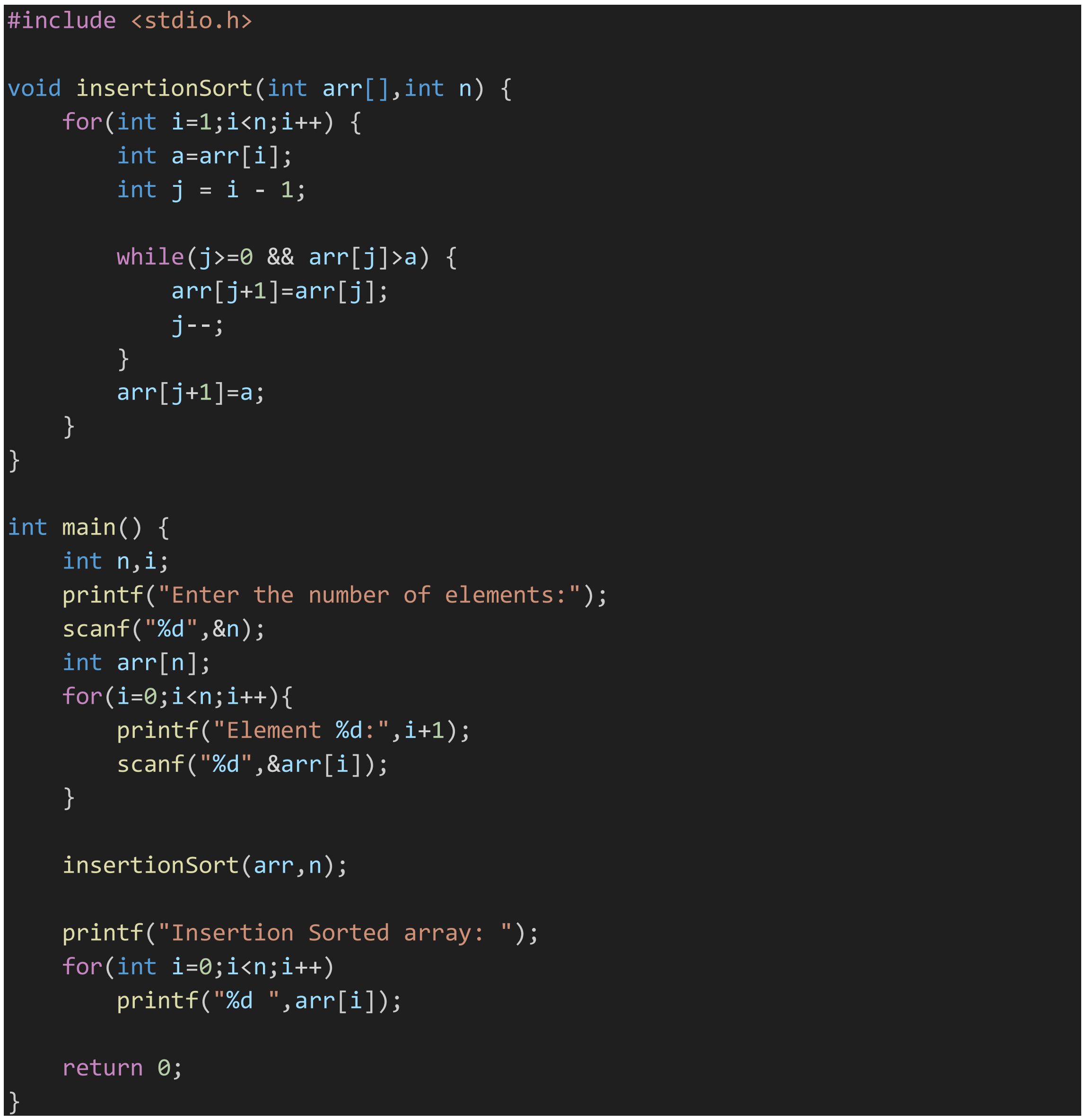
```
C:\Sem-4\DAA\Sorting>gcc bubble.c

C:\Sem-4\DAA\Sorting>a
Enter the number of elements:5
Element 1:1906
Element 2:19
Element 3:6
Element 4:11
Element 5:3
```

Space complexity of Bubble sort: Uses constant extra variables. O(1)

Time complexity of Bubble sort: A double nested loop used to compare and bubble up elements. O(n^2)

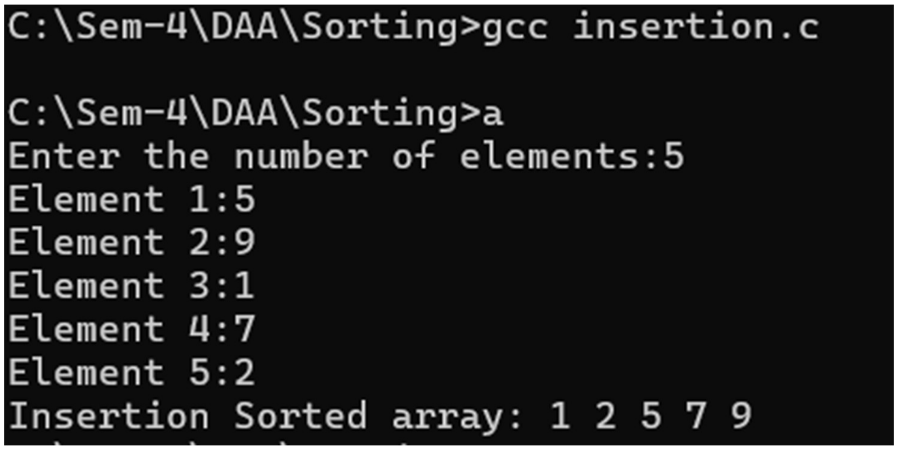
2. Write a program to implement insertion sort.

   Code:



```
#include <stdio.h>

void insertionSort(int arr[],int n) {
    for(int i=1;i<n;i++) {
        int a=arr[i];
        int j = i - 1;

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

int main() {
    int n,i;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++){
        printf("Element %d:",i+1);
        scanf("%d",&arr[i]);
    }

    insertionSort(arr,n);

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

    return 0;
}
```

Output:



```
C:\Sem-4\DAA\Sorting>gcc insertion.c

C:\Sem-4\DAA\Sorting>a
Enter the number of elements:5
Element 1:5
Element 2:9
Element 3:1
Element 4:7
Element 5:2
Insertion Sorted array: 1 2 5 7 9
```

Space complexity of Insertion sort: Uses constant extra variables. O(1)

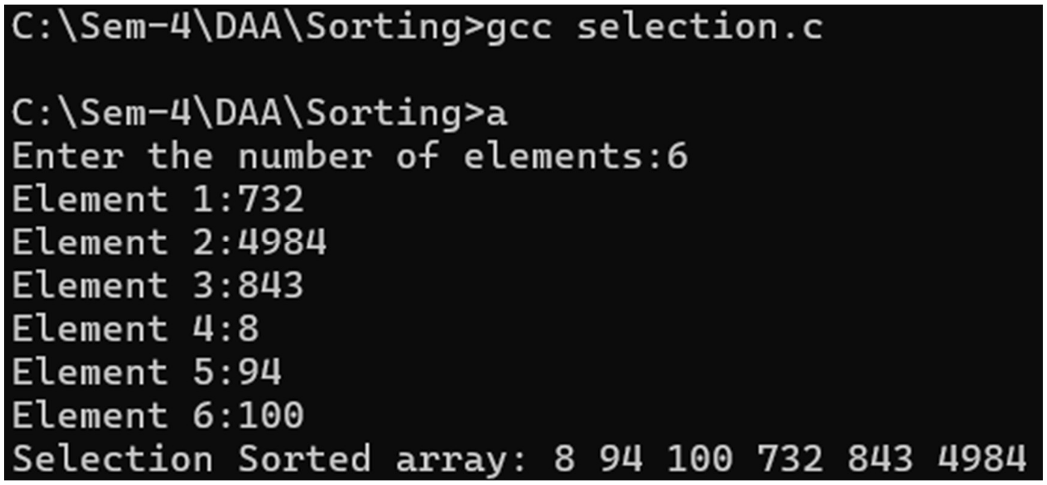
Time complexity of Insertion sort: A double nested loop is used to compare and shift elements. $O(n^2)$

3. Write a program to implement 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[min];
        arr[min]=arr[i];
        arr[i]=temp;
    }
}

int main() {
    int n,i;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++){
        printf("Element %d:",i+1);
        scanf("%d",&arr[i]);
    }

    selectionSort(arr,n);

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

    return 0;
}
```

Output:



```
C:\Sem-4\DAA\Sorting>gcc selection.c

C:\Sem-4\DAA\Sorting>a
Enter the number of elements:6
Element 1:732
Element 2:4984
Element 3:843
Element 4:8
Element 5:94
Element 6:100
Selection Sorted array: 8 94 100 732 843 4984
```

Space complexity of Selection sort: Uses constant extra variables. O(1)

Time complexity of Selection sort: Uses a double nested loop to scan the minimum element in the remaining array. O(n^2)

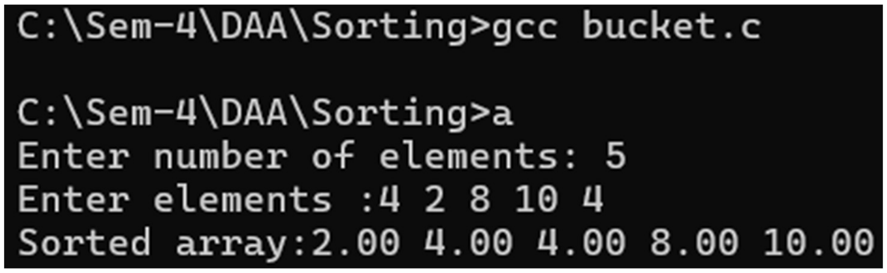
4. Write a program to implement Bucket Sort.

Code:

```c
#include <stdio.h>

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

void bucketSort(float arr[], int n) {
    int i, j;

    float min = arr[0], max = arr[0];
    for (i = 1; i < n; i++) {
        if (arr[i] < min) min = arr[i];
        if (arr[i] > max) max = arr[i];
    }

    float bucket[n][n];
    int count[n];

    for (i = 0; i < n; i++)
        count[i] = 0;

    for (i = 0; i < n; i++) {
        int bi = (int)(((arr[i] - min) / (max - min)) * (n - 1));
        bucket[bi][count[bi]++] = arr[i];
    }

    for (i = 0; i < n; i++)
```

```c
        insertionSort(bucket[i], count[i]);

    int index = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < count[i]; j++)
            arr[index++] = bucket[i][j];
}

int main() {
    int n, i;
    float arr[100];

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for (i = 0; i < n; i++)
        scanf("%f", &arr[i]);

    bucketSort(arr, n);

    printf("Sorted array: ");
    for (i = 0; i < n; i++)
        printf("%.2f ", arr[i]);

    return 0;
}
```

Output:



```
C:\Sem-4\DAA\Sorting>gcc bucket.c

C:\Sem-4\DAA\Sorting>a
Enter number of elements: 5
Enter elements :4 2 8 10 4
Sorted array:2.00 4.00 4.00 8.00 10.00
```

Space complexity of Bucket sort: Uses a nxn bucket array. O(n^2)

Time complexity of Bucket sort: Uses insertion sort to sort elements. O(n^2)

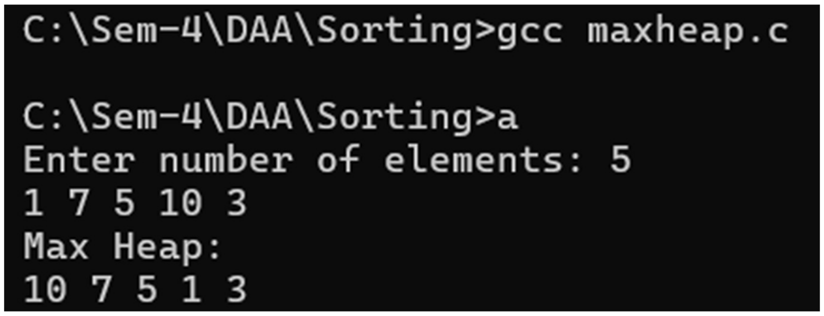
5. Write a program to implement Heap sort .

   i) Max heap sort

   Code:

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void maxHeapify(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) {
        swap(&arr[i], &arr[largest]);
        maxHeapify(arr, n, largest);
    }
}

void buildMaxHeap(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        maxHeapify(arr, n, i);
}

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    buildMaxHeap(arr, n);

    printf("Max Heap:\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Output:

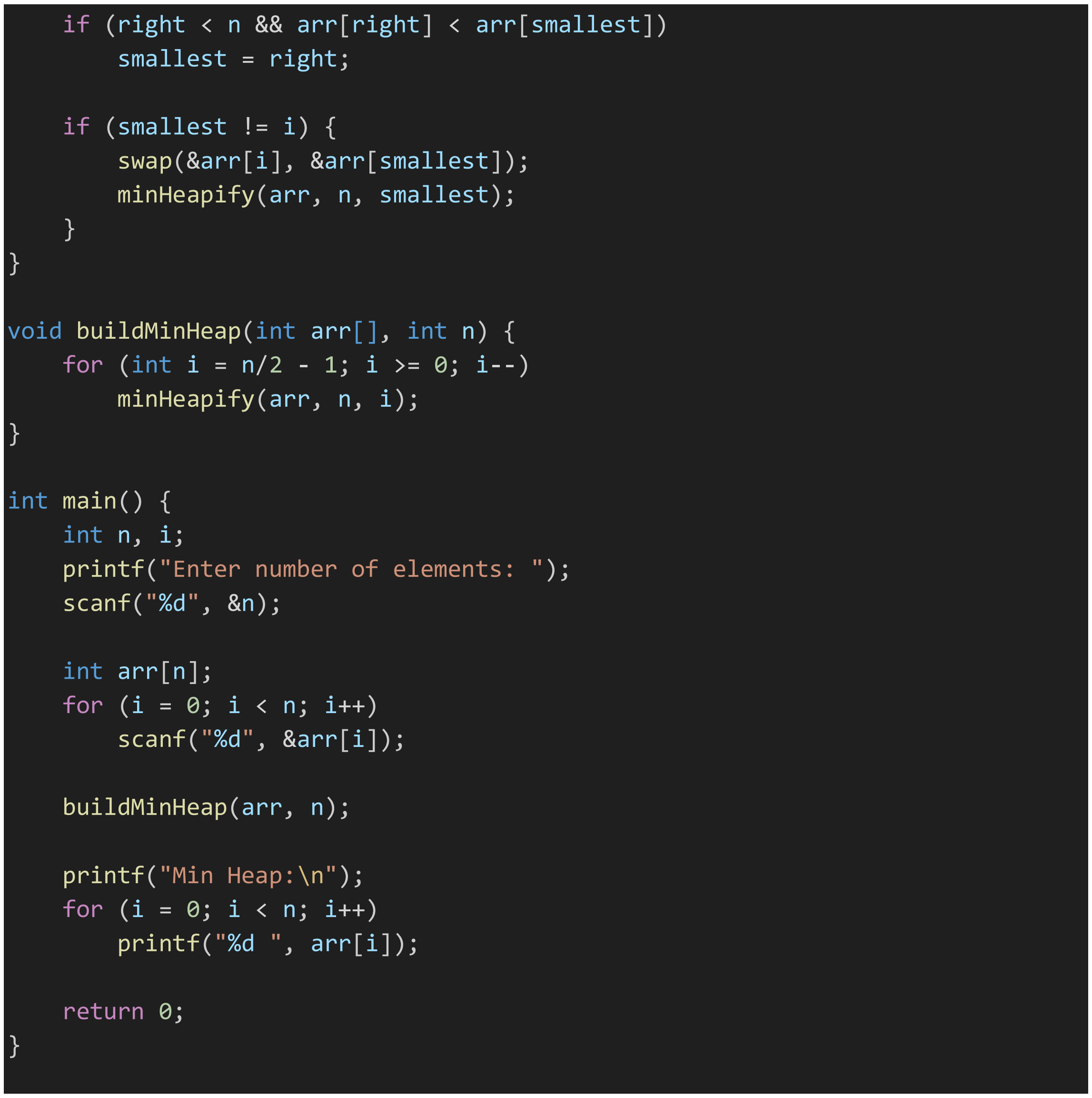


```
C:\Sem-4\DAA\Sorting>gcc maxheap.c

C:\Sem-4\DAA\Sorting>a
Enter number of elements: 5
1 7 5 10 3
Max Heap:
10 7 5 1 3
```

ii) Min Heap sort

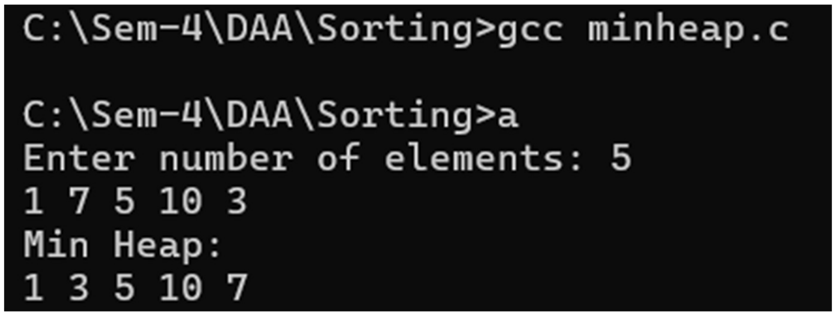
Code:

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void minHeapify(int arr[], int n, int i) {
    int smallest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < n && arr[left] < arr[smallest])
        smallest = left;
```



```c
    if (right < n && arr[right] < arr[smallest])
        smallest = right;

    if (smallest != i) {
        swap(&arr[i], &arr[smallest]);
        minHeapify(arr, n, smallest);
    }
}

void buildMinHeap(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        minHeapify(arr, n, i);
}

int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    buildMinHeap(arr, n);

    printf("Min Heap:\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);

    return 0;
}
```

Output:



```
C:\Sem-4\DAA\Sorting>gcc minheap.c

C:\Sem-4\DAA\Sorting>a
Enter number of elements: 5
1 7 5 10 3
Min Heap:
1 3 5 10 7
```

Space complexity of Heap sort: The heapify function is recursive and it work logn times, because the height of the heap is logn atmost . O(logn)

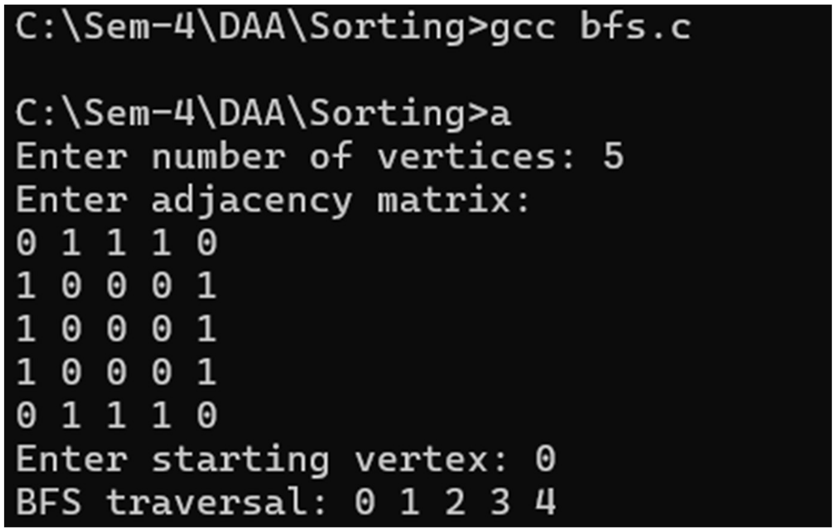
Time complexity of Heap sort: Heap operations take log n time for n elements, so O(nlogn)

6. Write a program to implement Breadth First Search.

   Code:

```c
#include <stdio.h>

int adj[10][10], visited[10], queue[10];
int n, front = 0, rear = -1;

void bfs(int start) {
    int i;
    visited[start] = 1;
    queue[++rear] = start;

    while (front <= rear) {
        int v = queue[front++];
        printf("%d ", v);

        for (i = 0; i < n; i++) {
            if (adj[v][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }
}

int main() {
    int i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    for (i = 0; i < n; i++)
        visited[i] = 0;

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("BFS traversal: ");
    bfs(start);

    return 0;
}
```

Output:



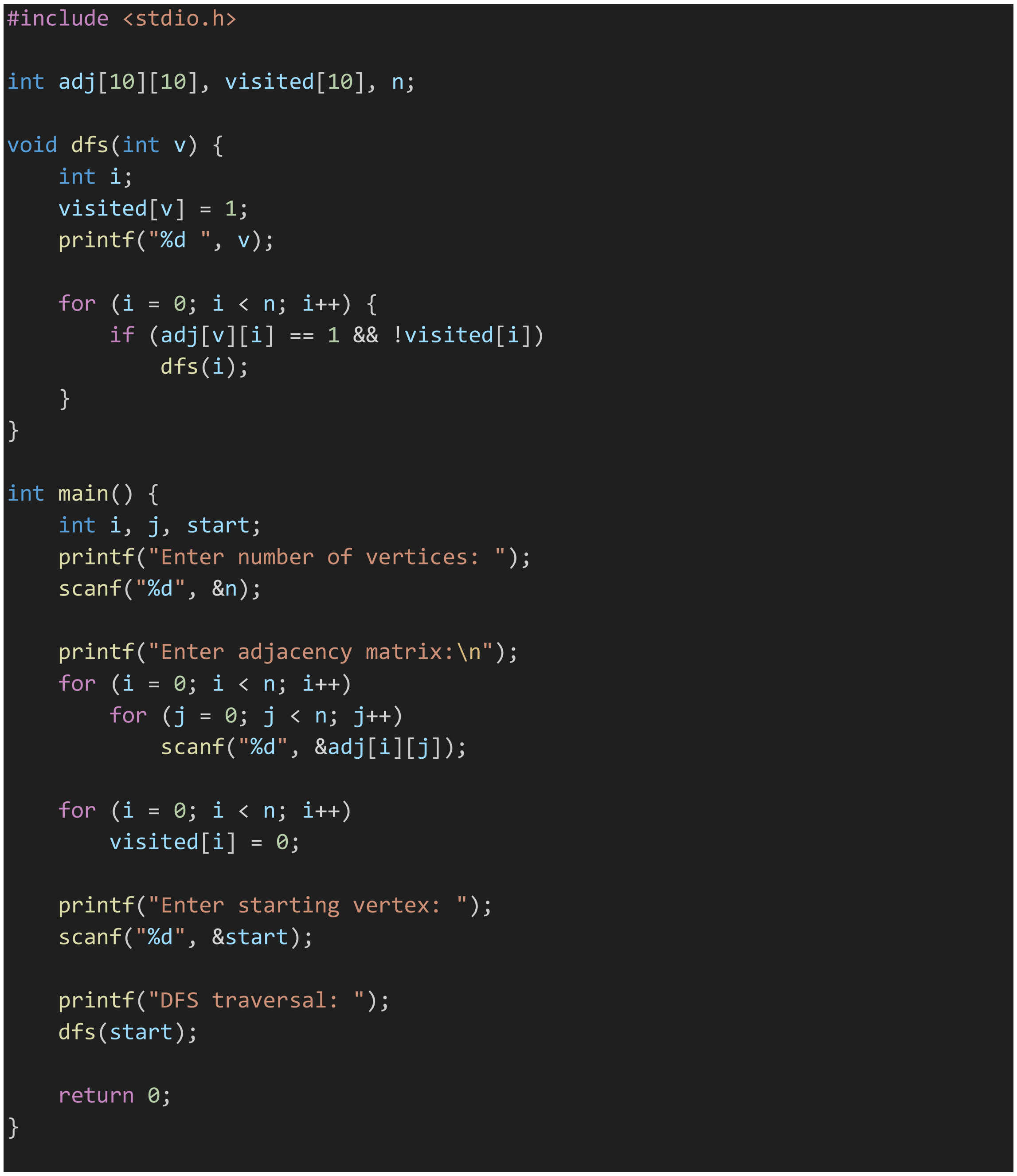
```
C:\Sem-4\DAA\Sorting>gcc bfs.c

C:\Sem-4\DAA\Sorting>a
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
0 1 1 1 0
Enter starting vertex: 0
BFS traversal: 0 1 2 3 4
```

Space complexity of Breadth First Search: The space of adjacency matrix dominates all other variables. O(n^2)

Time complexity of Breadth First Search: Each element of a 2D array (adjacency matrix) is scanned using a double nested loop. O(n^2)

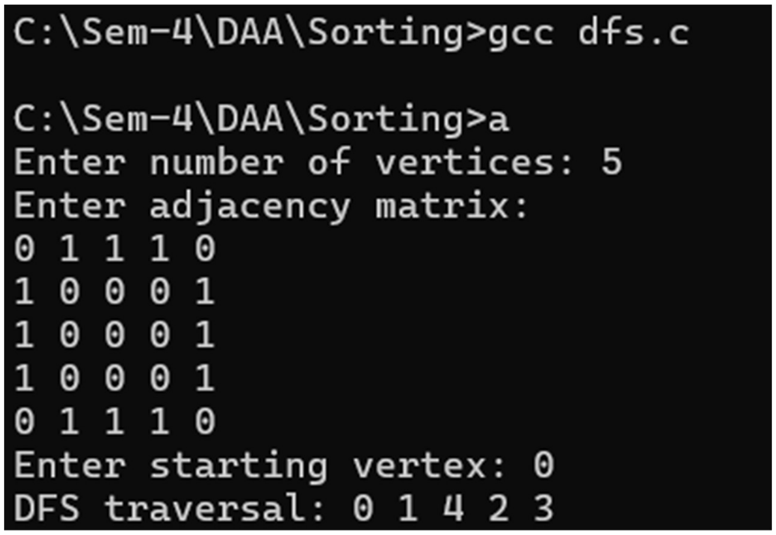
7. Write a program to implement Depth First Search.

Code:



```c
#include <stdio.h>

int adj[10][10], visited[10], n;

void dfs(int v) {
    int i;
    visited[v] = 1;
    printf("%d ", v);

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

int main() {
    int i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);

    for (i = 0; i < n; i++)
        visited[i] = 0;

    printf("Enter starting vertex: ");
    scanf("%d", &start);

    printf("DFS traversal: ");
    dfs(start);

    return 0;
}
```

Output:



```
C:\Sem-4\DAA\Sorting>gcc dfs.c

C:\Sem-4\DAA\Sorting>a
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
0 1 1 1 0
Enter starting vertex: 0
DFS traversal: 0 1 4 2 3
```

Space complexity of Depth First Search: Uses a recursive function inside a loop. O(n^2)

Time complexity of Depth First Search: Checking adjacency matrix with recursive function. O(n^2)