AMRITA VISHWA VIDYAPEETHAM

Second Year B.Tech

(Computer Science and Engineering)

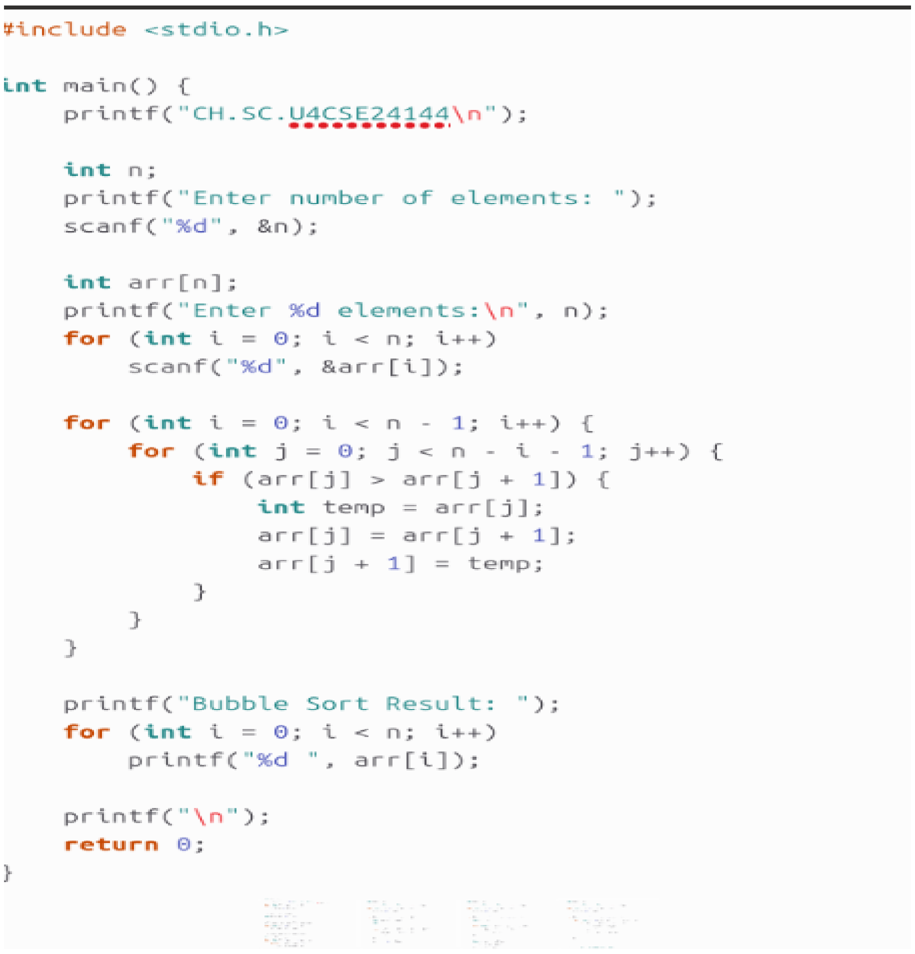
Design and Analysis of Algorithms

TASK – 2

Name: Shaik Kolimi Dada Hussain
College Name: Amrita vishwa Vidyapeetham
Roll No : ch.sc.u4cse24144
Department: CSE-B
Academic Year : 2024-2028

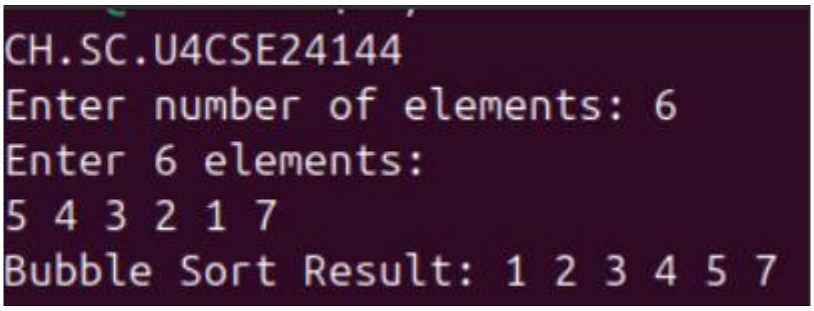
1)Bubble Sort ,its time complexity and space complexity with justification.

CODE:



```c
#include <stdio.h>

int main() {
    printf("CH.SC.U4CSE24144\n");

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

    int arr[n];
    printf("Enter %d elements:\n", 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("Bubble Sort Result: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

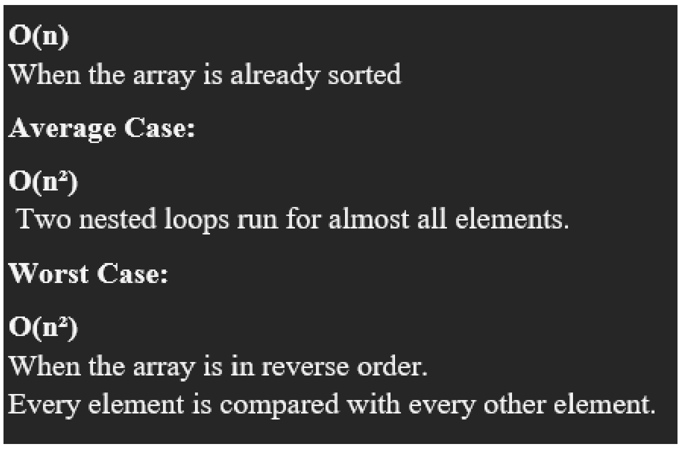
OUTPUT:



```
CH.SC.U4CSE24144
Enter number of elements: 6
Enter 6 elements:
5 4 3 2 1 7
Bubble Sort Result: 1 2 3 4 5 7
```

Time and Space Complexity:

Best Case:



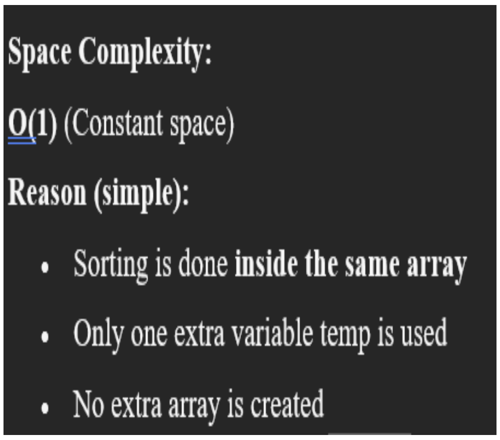
O(n)
When the array is already sorted

Average Case:

O(n^2)
Two nested loops run for almost all elements.

Worst Case:

O(n^2)
When the array is in reverse order.
Every element is compared with every other element.



Space Complexity:

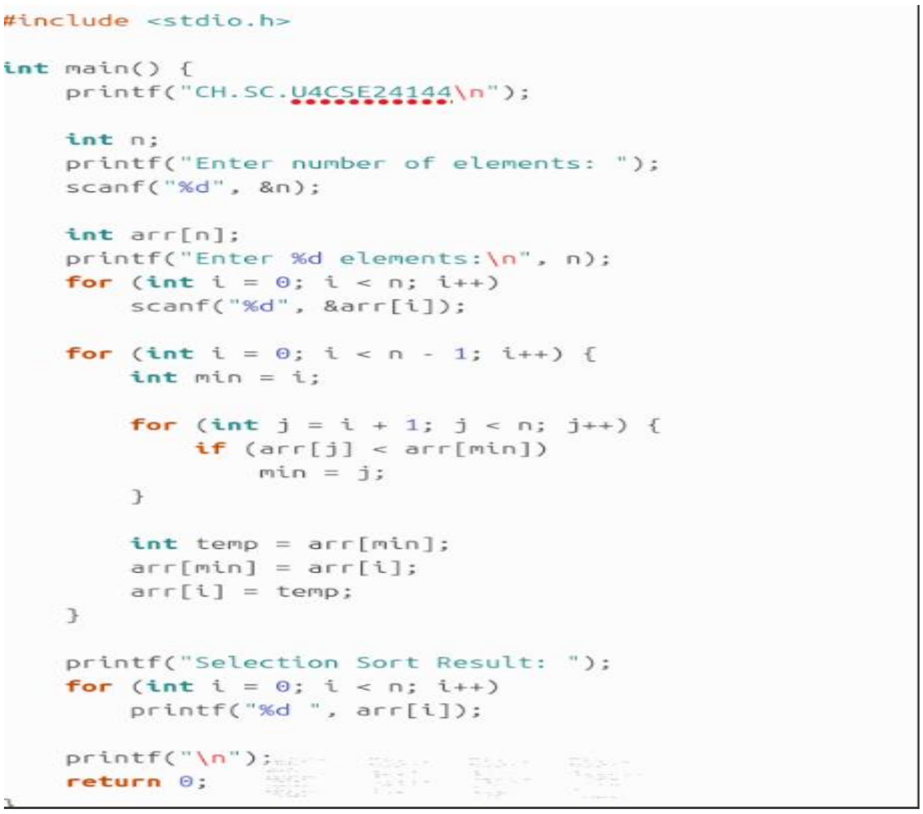
O(1) (Constant space)

Reason (simple):

- Sorting is done **inside the same array**
- Only one extra variable temp is used
- No extra array is created

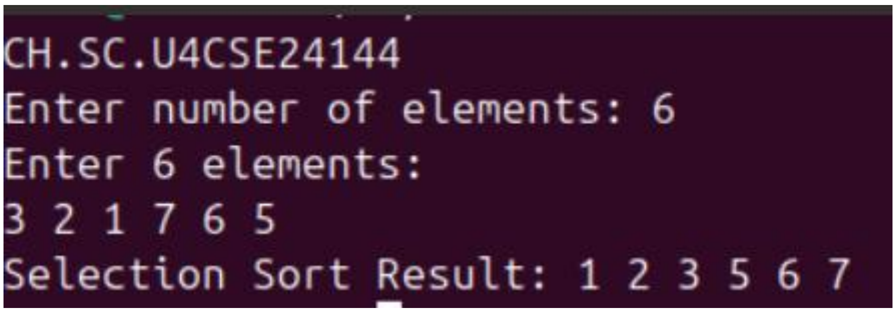
2) Selection Sort ,its time complexity and space complexity with justification.

CODE:



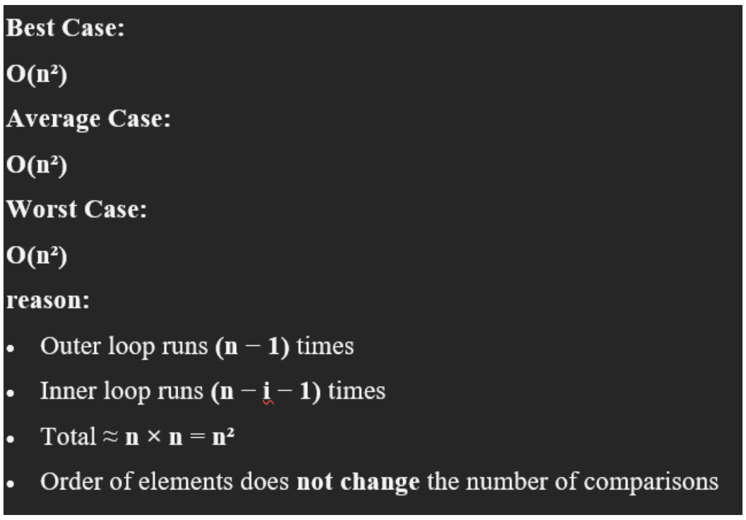
```
#include <stdio.h>

int main() {
    printf("CH.SC.U4CSE24144\n");

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

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

    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;
    }

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

    printf("\n");
    return 0;
```

OUTPUT:



```
CH.SC.U4CSE24144
Enter number of elements: 6
Enter 6 elements:
3 2 1 7 6 5
Selection Sort Result: 1 2 3 5 6 7
```

Time and Space Complexity:



Best Case:

O(n²)

Average Case:

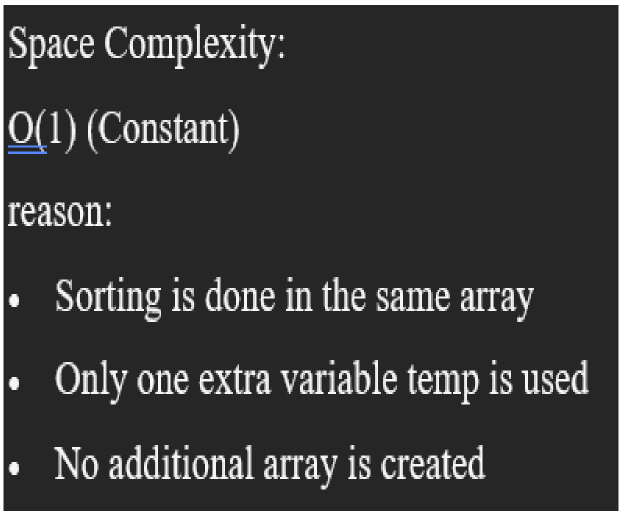
O(n²)

Worst Case:

O(n²)

reason:

- Outer loop runs **(n − 1)** times
- Inner loop runs **(n − i − 1)** times
- Total ≈ **n × n = n²**
- Order of elements does **not change** the number of comparisons



Space Complexity:

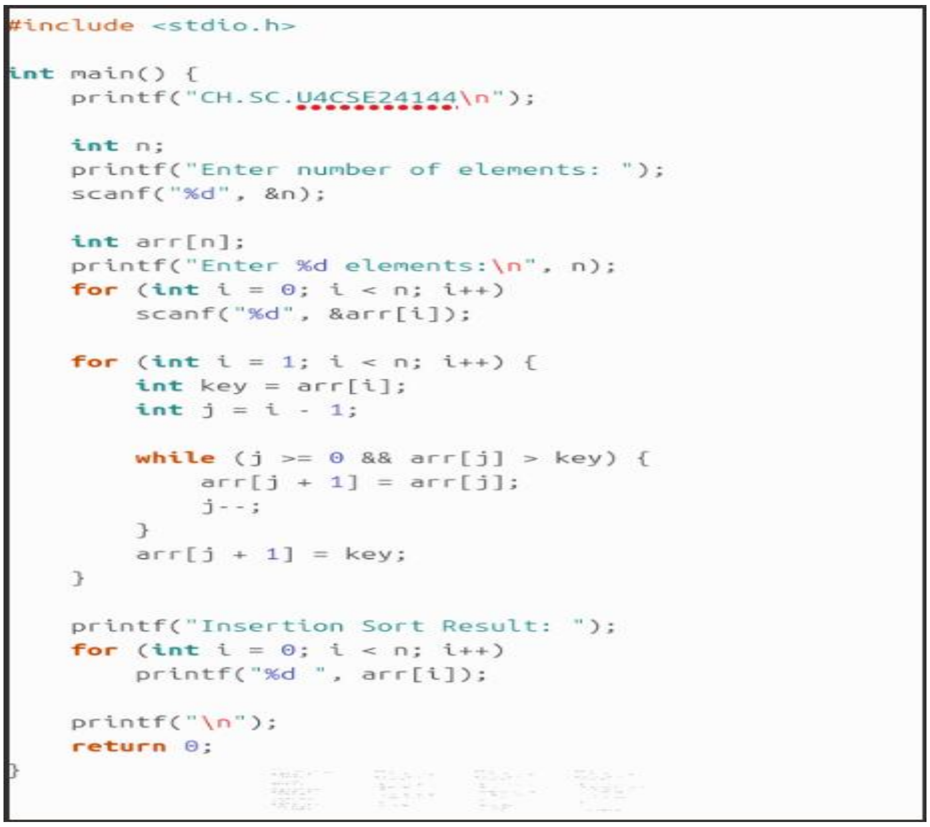
O(1) (Constant)

reason:

- Sorting is done in the same array
- Only one extra variable temp is used
- No additional array is created

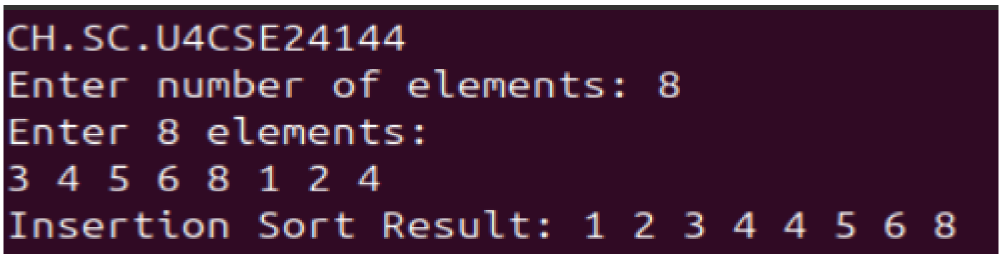
3) Insertion Sort ,its time complexity and space complexity with justification.

CODE:



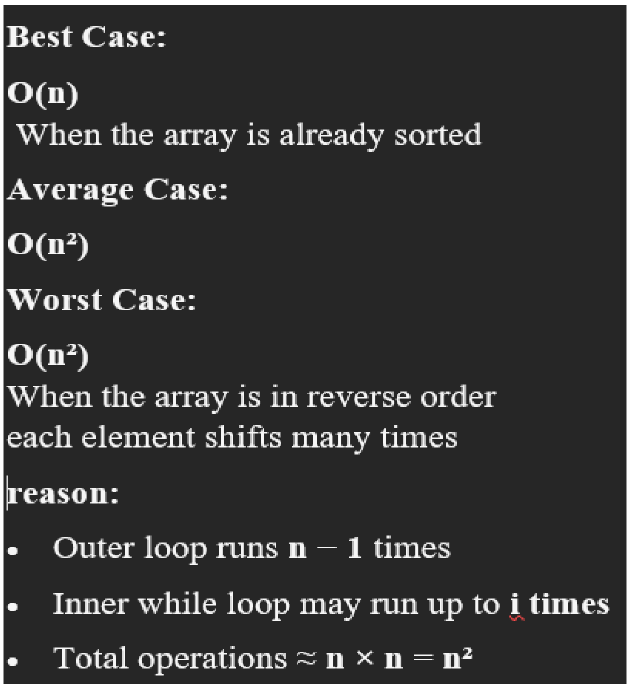
```
#include <stdio.h>

int main() {
    printf("CH.SC.U4CSE24144\n");

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

    int arr[n];
    printf("Enter %d elements:\n", 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("Insertion Sort Result: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

OUTPUT:



```
CH.SC.U4CSE24144
Enter number of elements: 8
Enter 8 elements:
3 4 5 6 8 1 2 4
Insertion Sort Result: 1 2 3 4 4 5 6 8
```

Time and Space Complexity:



Best Case:

O(n)
When the array is already sorted

Average Case:

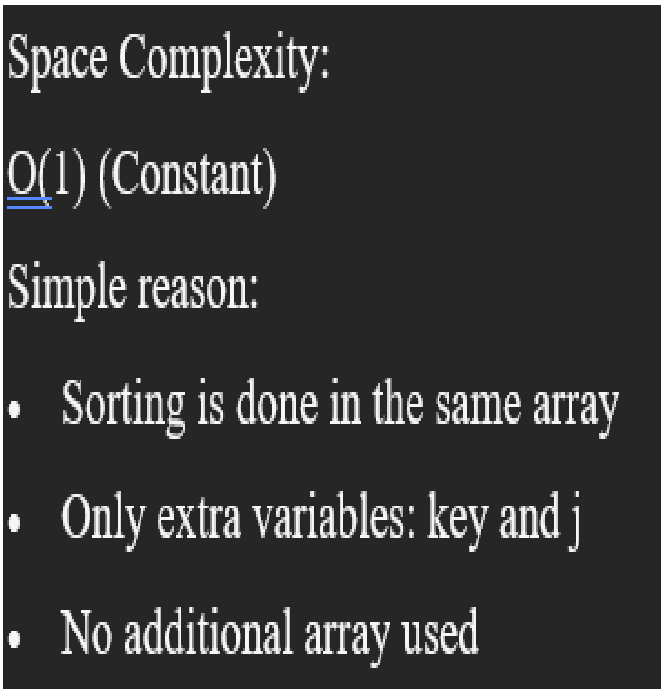
O(n²)

Worst Case:

O(n²)
When the array is in reverse order
each element shifts many times

reason:

- Outer loop runs **n − 1** times
- Inner while loop may run up to **i times**
- Total operations ≈ **n × n = n²**



Space Complexity:

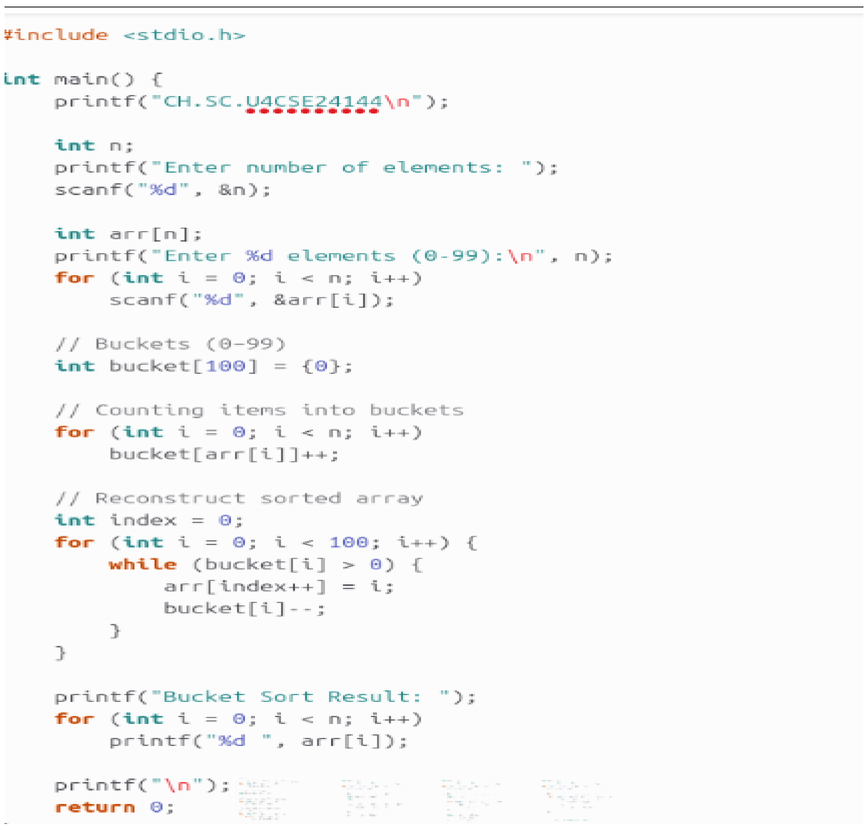
O(1) (Constant)

Simple reason:

- Sorting is done in the same array
- Only extra variables: key and j
- No additional array used

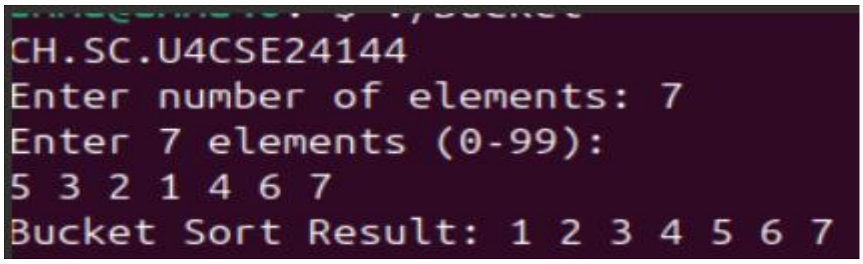
4)Bucket Sort ,its time complexity and space complexity with justification.

CODE:



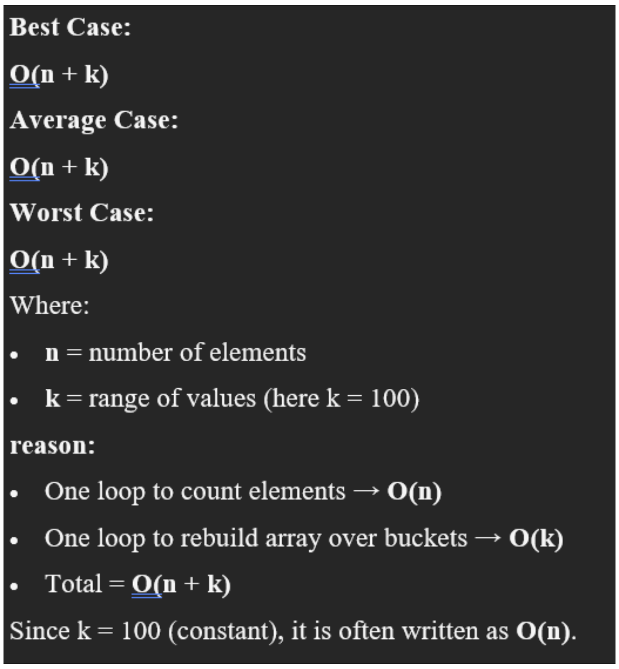
```
#include <stdio.h>

int main() {
    printf("CH.SC.U4CSE24144\n");

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

    int arr[n];
    printf("Enter %d elements (0-99):\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    // Buckets (0-99)
    int bucket[100] = {0};

    // Counting items into buckets
    for (int i = 0; i < n; i++)
        bucket[arr[i]]++;

    // Reconstruct sorted array
    int index = 0;
    for (int i = 0; i < 100; i++) {
        while (bucket[i] > 0) {
            arr[index++] = i;
            bucket[i]--;
        }
    }

    printf("Bucket Sort Result: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
```

OUTPUT:



```
CH.SC.U4CSE24144
Enter number of elements: 7
Enter 7 elements (0-99):
5 3 2 1 4 6 7
Bucket Sort Result: 1 2 3 4 5 6 7
```

Time and Space Complexity:



Best Case:

O(n + k)

Average Case:

O(n + k)

Worst Case:

O(n + k)

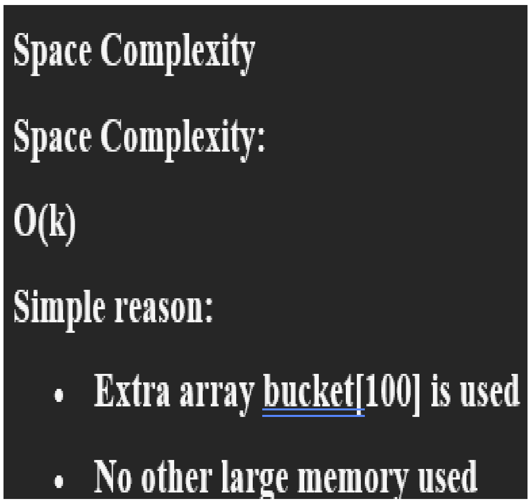
Where:

- **n** = number of elements
- **k** = range of values (here k = 100)

reason:

- One loop to count elements → **O(n)**
- One loop to rebuild array over buckets → **O(k)**
- Total = **O(n + k)**

Since k = 100 (constant), it is often written as **O(n)**.



Space Complexity

Space Complexity:

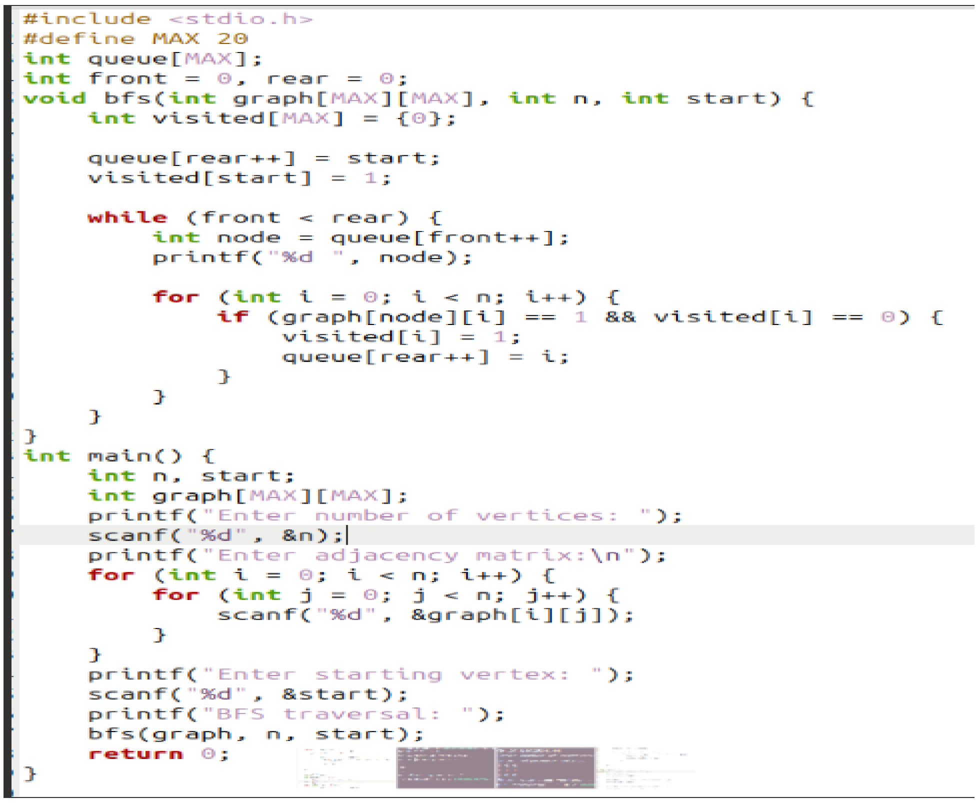
O(k)

Simple reason:

- Extra array bucket[100] is used
- No other large memory used

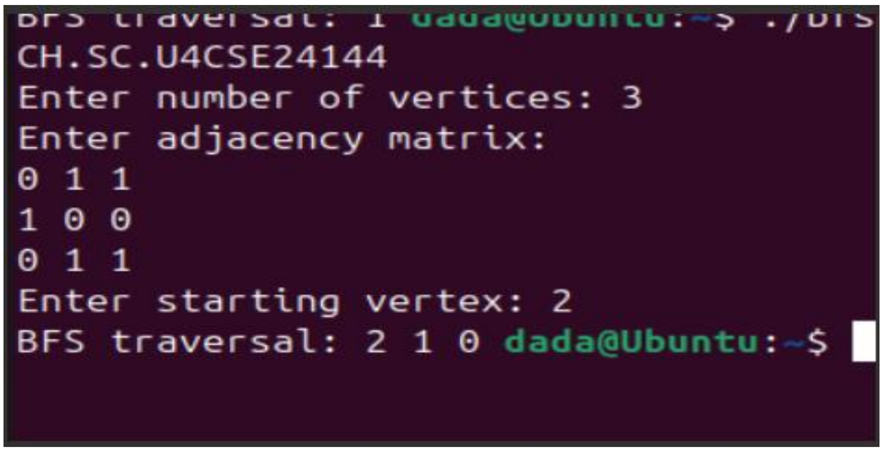
5) BFS ,its time complexity and space complexity with justification.

CODE:



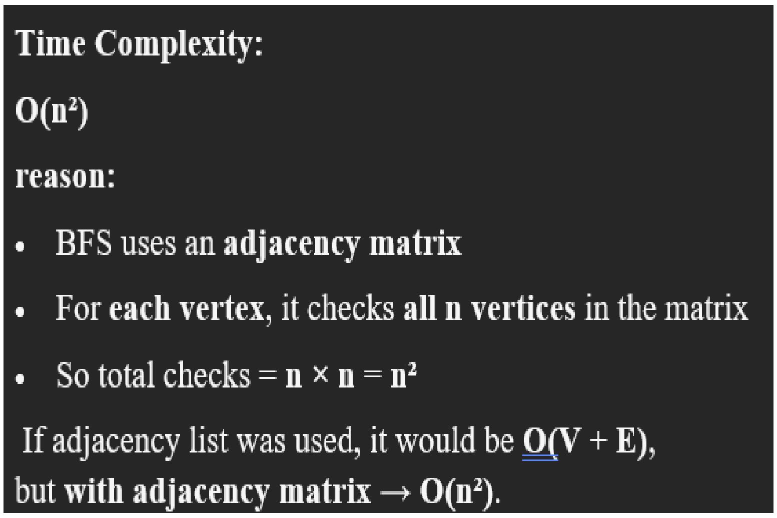
```c
#include <stdio.h>
#define MAX 20
int queue[MAX];
int front = 0, rear = 0;
void bfs(int graph[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};

    queue[rear++] = start;
    visited[start] = 1;

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

        for (int i = 0; i < n; i++) {
            if (graph[node][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}
int main() {
    int n, start;
    int graph[MAX][MAX];
    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]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("BFS traversal: ");
    bfs(graph, n, start);
    return 0;
}
```

OUTPUT:



```
BFS traversal: 1 dada@Ubuntu:~$ ./bfs
CH.SC.U4CSE24144
Enter number of vertices: 3
Enter adjacency matrix:
0 1 1
1 0 0
0 1 1
Enter starting vertex: 2
BFS traversal: 2 1 0 dada@Ubuntu:~$
```

Time and Space Complexity:



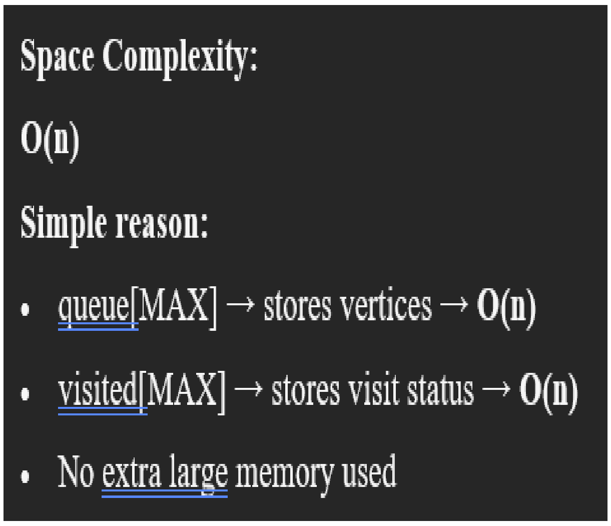
Time Complexity:

O(n²)

reason:

- BFS uses an **adjacency matrix**
- For **each vertex**, it checks **all n vertices** in the matrix
- So total checks = **n × n = n²**

If adjacency list was used, it would be **O(V + E)**, but **with adjacency matrix → O(n²)**.



Space Complexity:

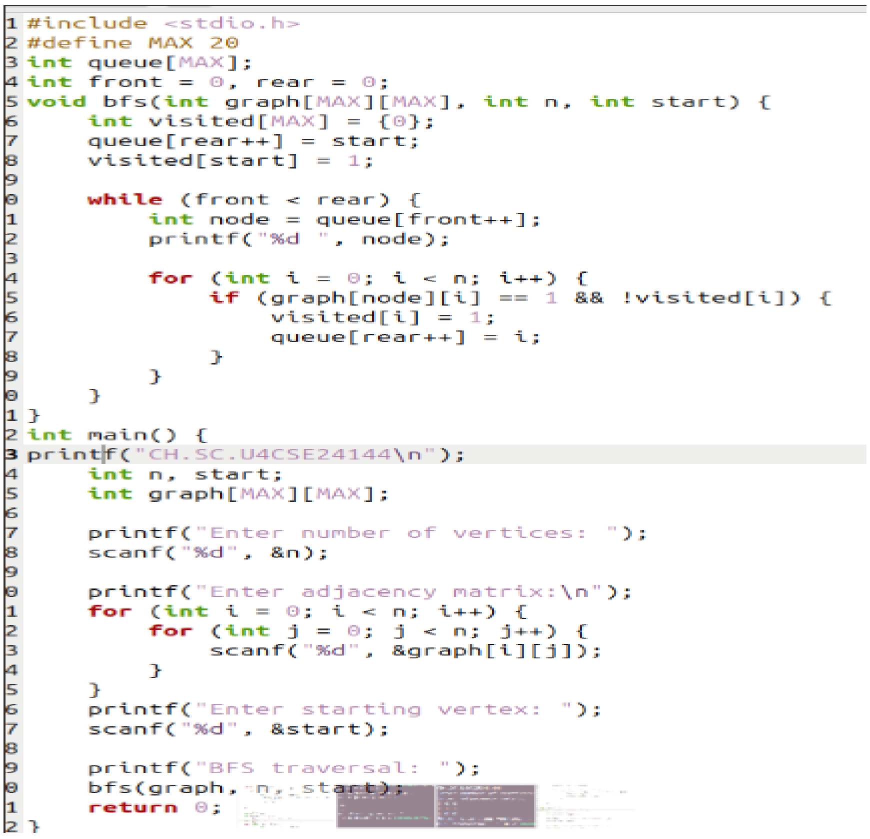
O(n)

Simple reason:

- queue[MAX] → stores vertices → **O(n)**
- visited[MAX] → stores visit status → **O(n)**
- No extra large memory used

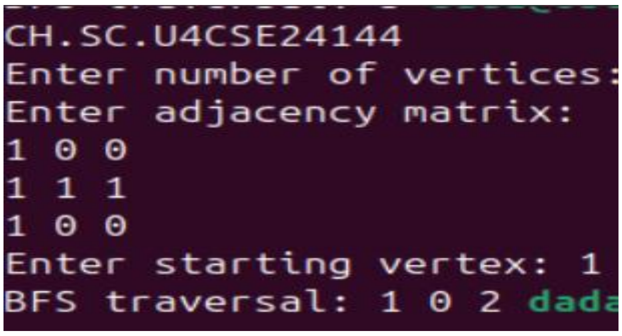
6) DFS ,its time complexity and space complexity with justification.

CODE:



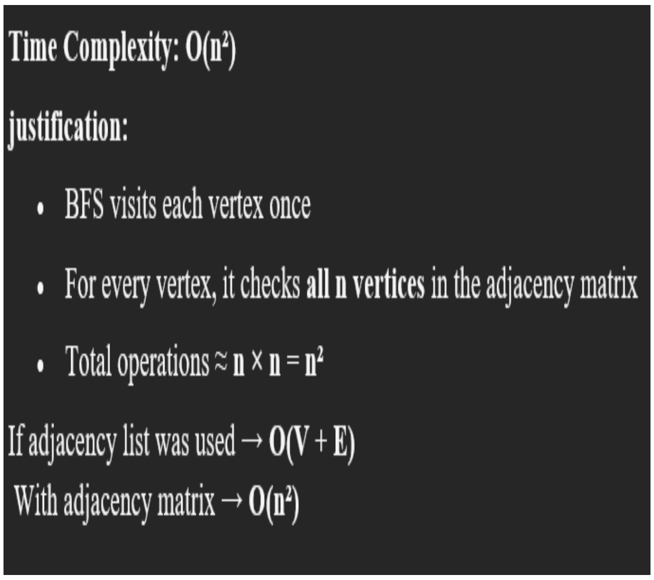
```
#include <stdio.h>
#define MAX 20
int queue[MAX];
int front = 0, rear = 0;
void bfs(int graph[MAX][MAX], int n, int start) {
    int visited[MAX] = {0};
    queue[rear++] = start;
    visited[start] = 1;

    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() {
printf("CH.SC.U4CSE24144\n");
    int n, start;
    int graph[MAX][MAX];

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

    printf("BFS traversal: ");
    bfs(graph, n, start);
    return 0;
}
```

OUTPUT:



```
CH.SC.U4CSE24144
Enter number of vertices:
Enter adjacency matrix:
1 0 0
1 1 1
1 0 0
Enter starting vertex: 1
BFS traversal: 1 0 2 dada
```

Time and Space Complexity:



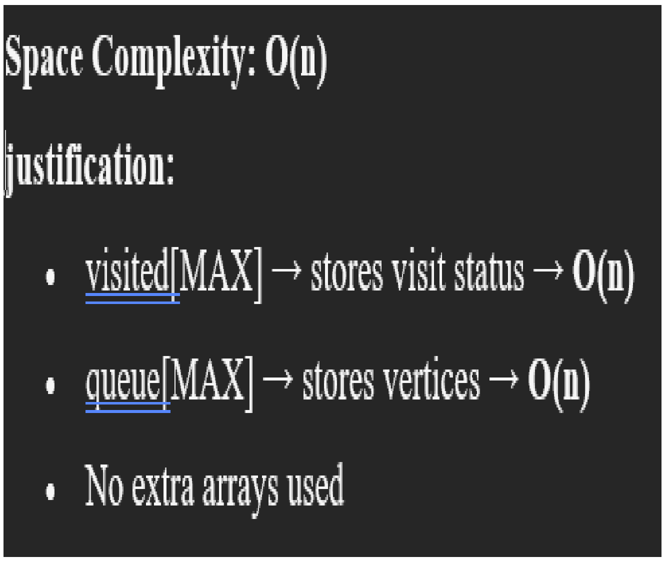
Time Complexity: $O(n^2)$

justification:

- BFS visits each vertex once
- For every vertex, it checks **all n vertices** in the adjacency matrix
- Total operations $\approx n \times n = n^2$

If adjacency list was used → $O(V + E)$

With adjacency matrix → $O(n^2)$



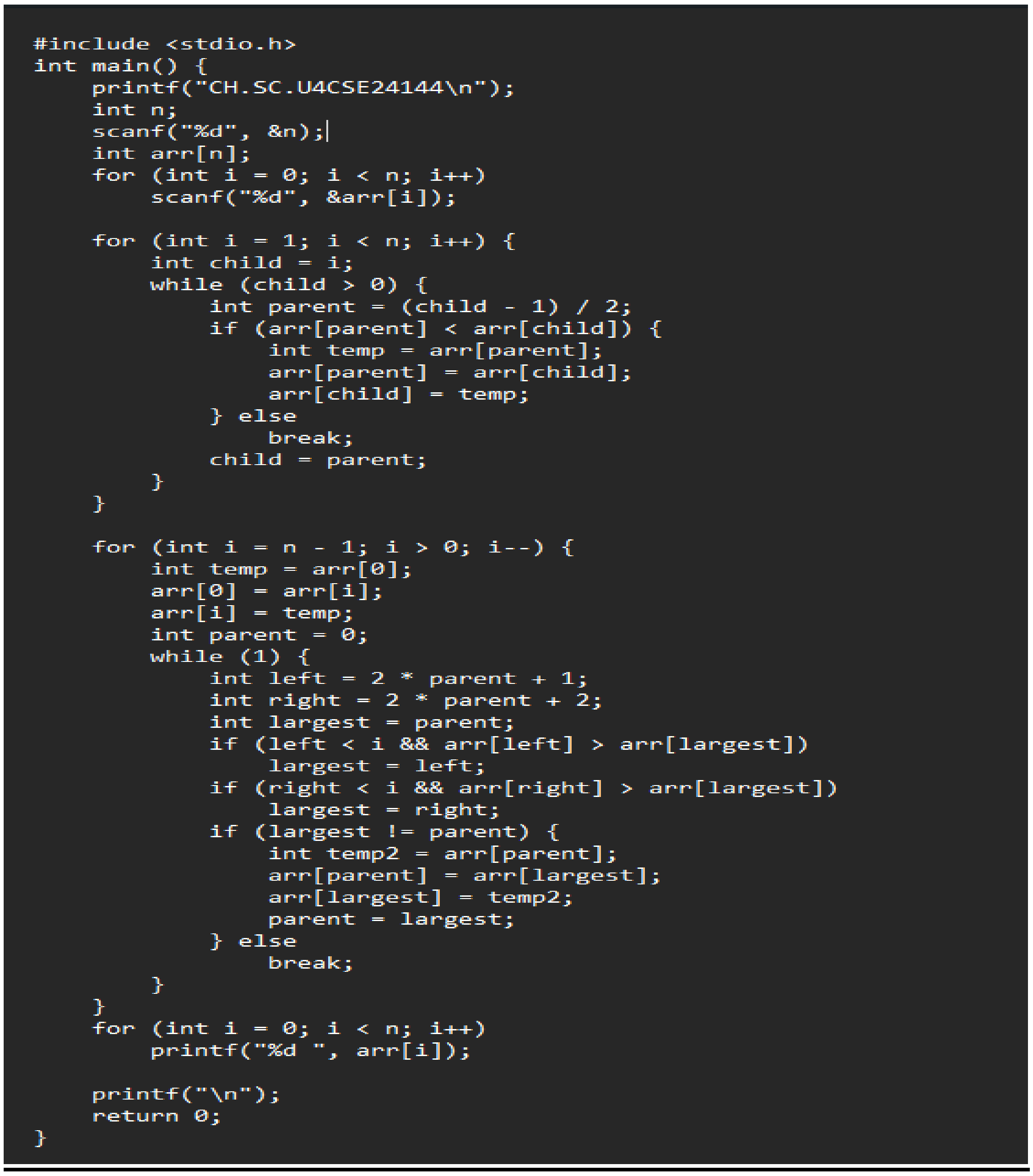
Space Complexity: O(n)

justification:

- visited[MAX] → stores visit status → **O(n)**
- queue[MAX] → stores vertices → **O(n)**
- No extra arrays used

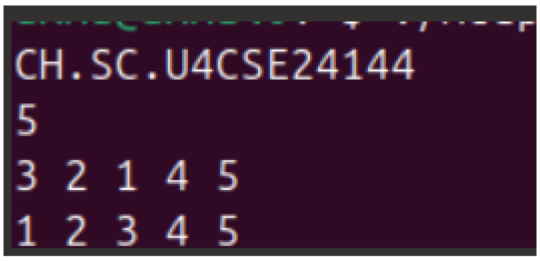
7) Heap sort,its time complexity and space complexity with justification.

CODE:



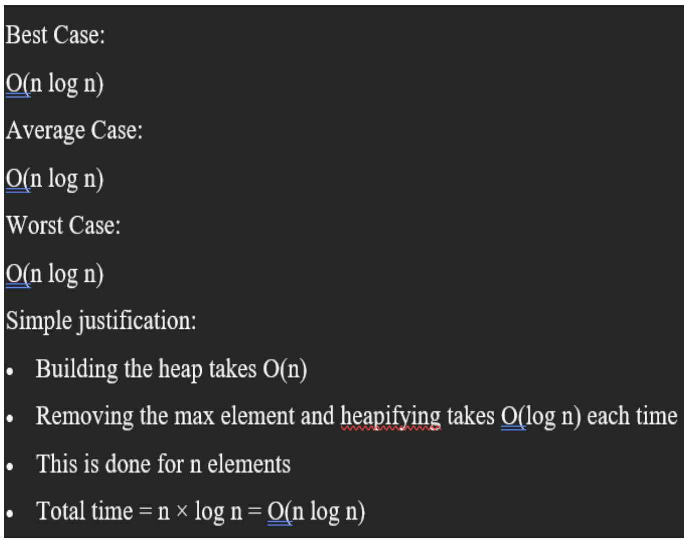
```
#include <stdio.h>
int main() {
    printf("CH.SC.U4CSE24144\n");
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for (int i = 1; i < n; i++) {
        int child = i;
        while (child > 0) {
            int parent = (child - 1) / 2;
            if (arr[parent] < arr[child]) {
                int temp = arr[parent];
                arr[parent] = arr[child];
                arr[child] = temp;
            } else
                break;
            child = parent;
        }
    }

    for (int i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        int parent = 0;
        while (1) {
            int left = 2 * parent + 1;
            int right = 2 * parent + 2;
            int largest = parent;
            if (left < i && arr[left] > arr[largest])
                largest = left;
            if (right < i && arr[right] > arr[largest])
                largest = right;
            if (largest != parent) {
                int temp2 = arr[parent];
                arr[parent] = arr[largest];
                arr[largest] = temp2;
                parent = largest;
            } else
                break;
        }
    }
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n");
    return 0;
}
```

OUTPUT:



```
CH.SC.U4CSE24144
5
3 2 1 4 5
1 2 3 4 5
```

Time and Space Complexity:



Best Case:

O(n log n)

Average Case:

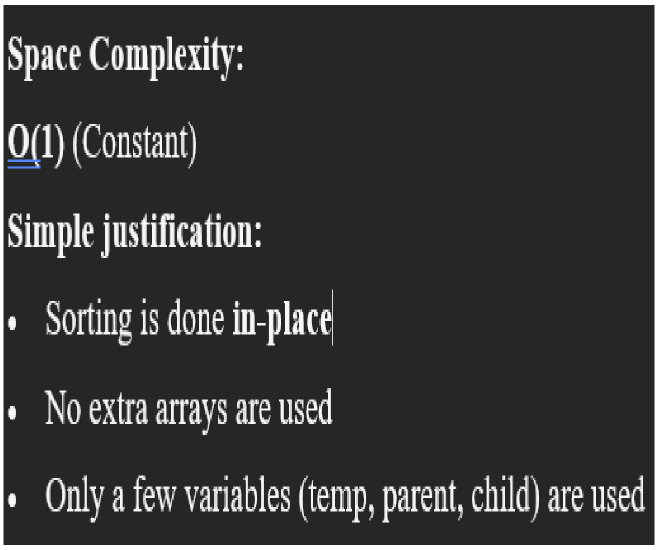
O(n log n)

Worst Case:

O(n log n)

Simple justification:

- Building the heap takes O(n)
- Removing the max element and heapifying takes O(log n) each time
- This is done for n elements
- Total time = n × log n = O(n log n)



Space Complexity:

O(1) (Constant)

Simple justification:

- Sorting is done **in-place**
- No extra arrays are used
- Only a few variables (temp, parent, child) are used