AMRITA VISHWA VIDYAPEETHAM

DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK

WEEK - 2

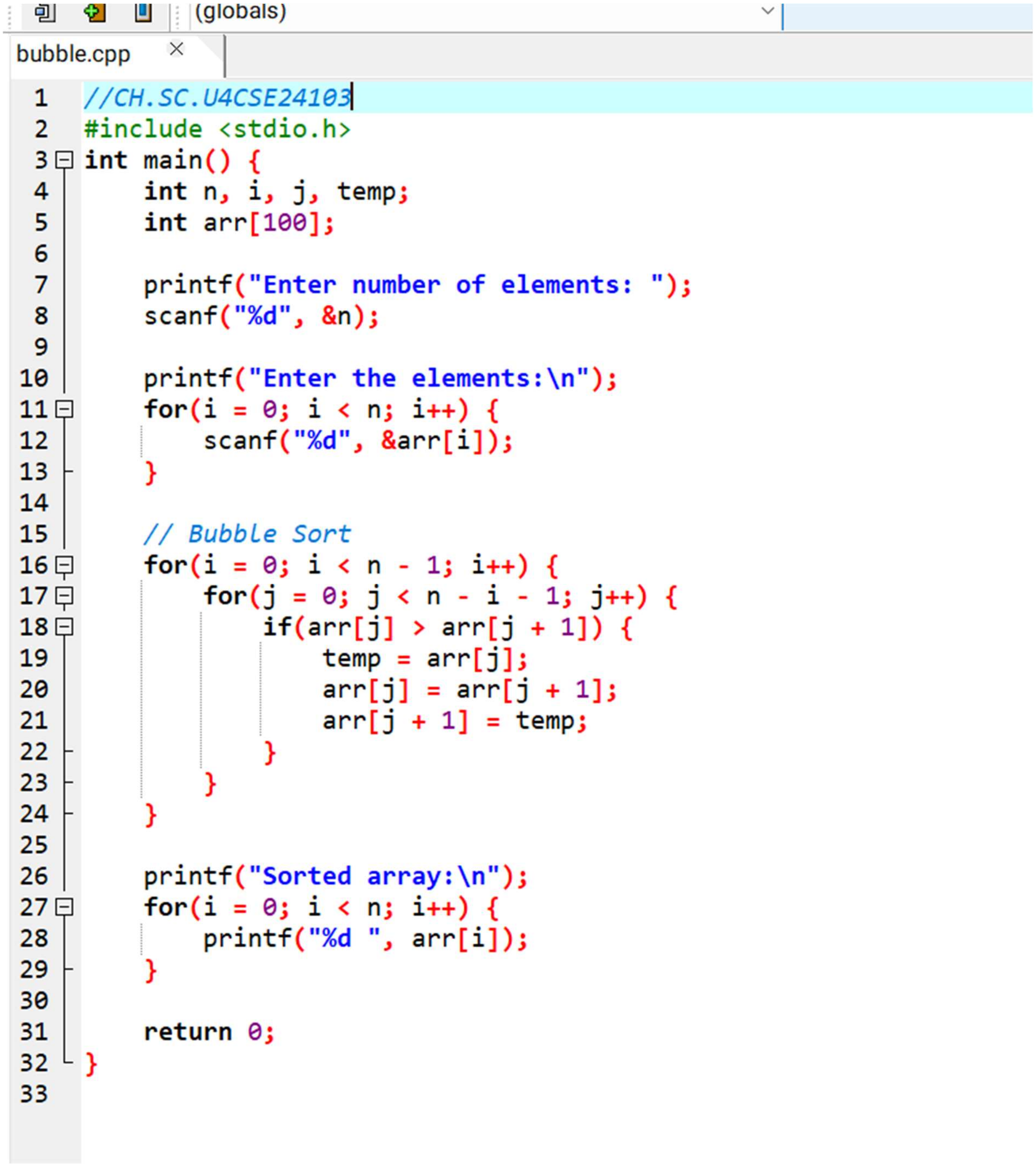
NAME : B S CHENTHIL HARI

ROLL NUMBER : CH.SC.U4CSE24103

CLASS : CSE-B

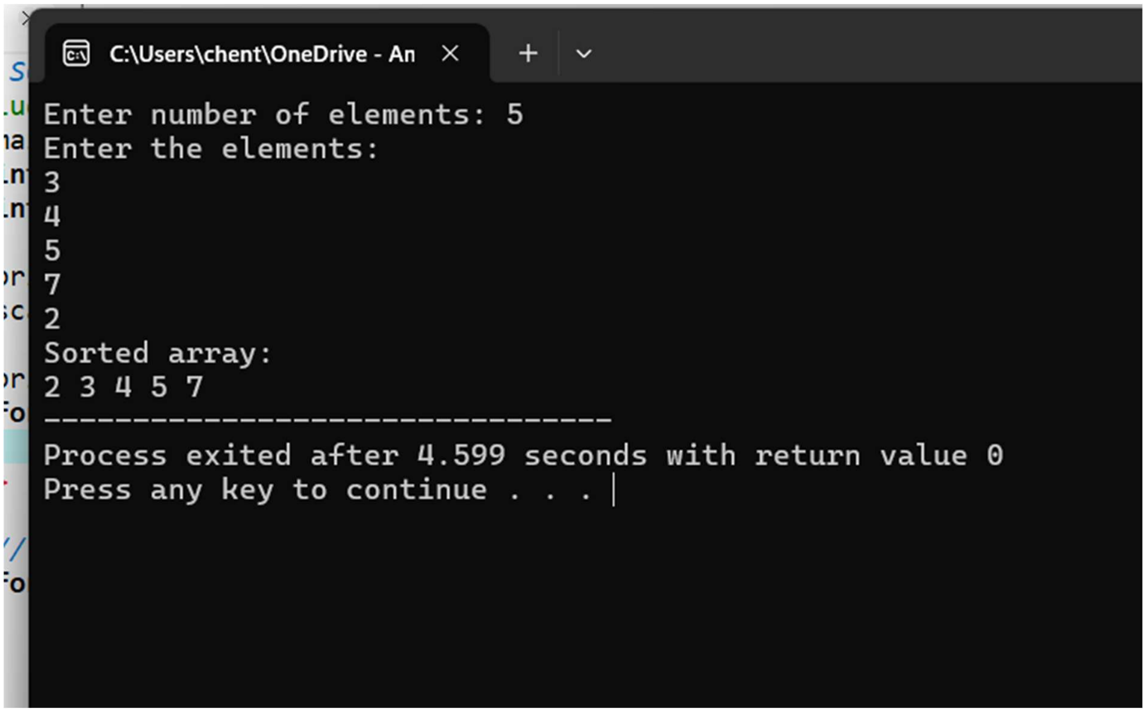
Question 1: BUBBLE SORT

CODE:



```c
//CH.SC.U4CSE24103
#include <stdio.h>
int main() {
    int n, i, j, temp;
    int arr[100];

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

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

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

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

    return 0;
}
```

OUTPUT:



```
Enter number of elements: 5
Enter the elements:
3
4
5
7
2
Sorted array:
2 3 4 5 7
--------------------------------
Process exited after 4.599 seconds with return value 0
Press any key to continue . . .
```

SPACE AND TIME COMPLEXITY JUSTFICATION:

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int temp → 4 bytes

Array used:

- int arr[size] → size × 4 bytes
- Total memory used by variables (excluding input array):
- 4 + 4 + 4 + 4 = 16 bytes

Time Complexity: O(n²)

- **Outer loop:**

Runs (n − 1) times

- **Inner loop:**

Runs (n − 1 − i) times for each outer loop iteration

- **Total number of comparisons:**

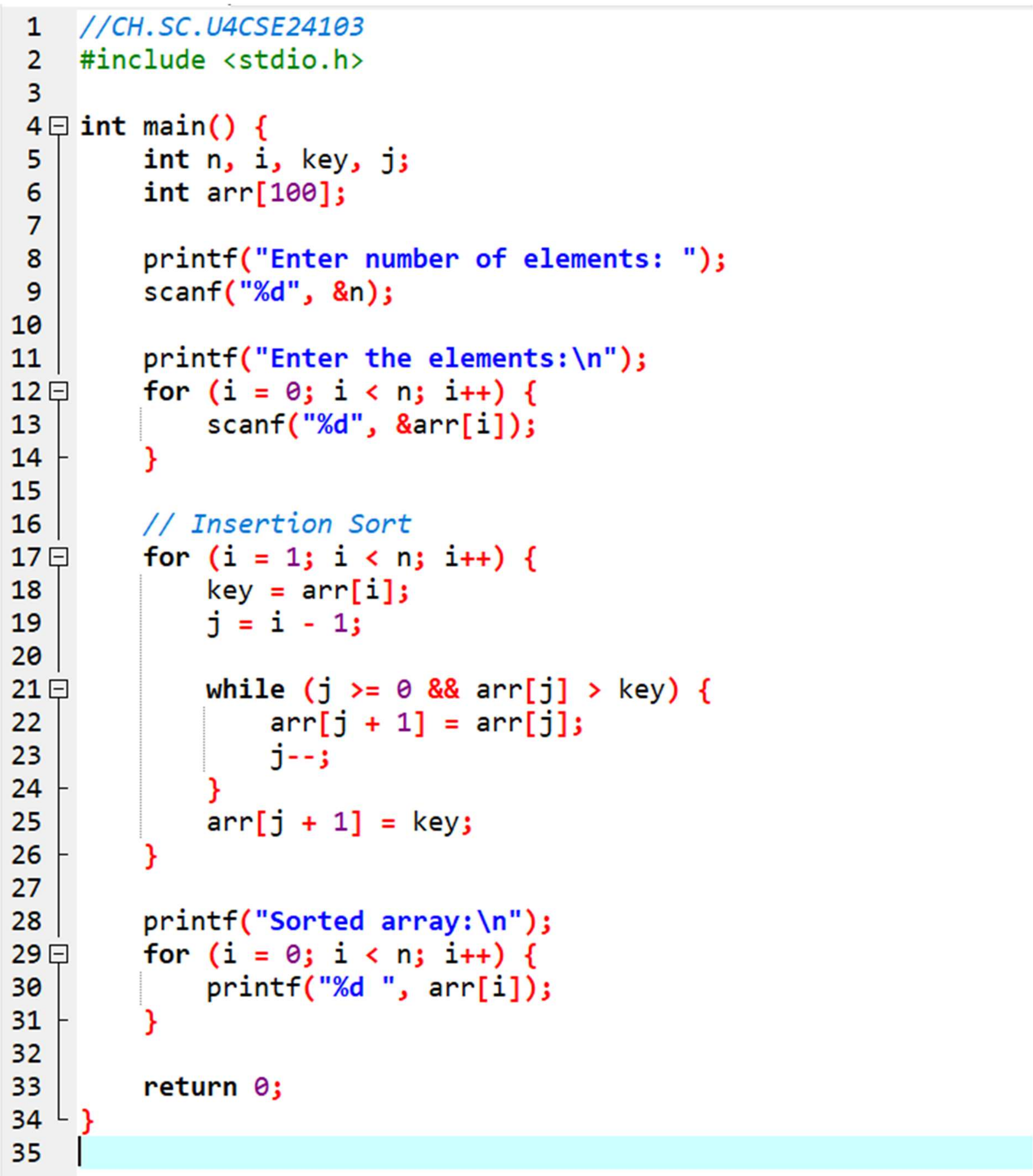
(n − 1) + (n − 2) + ... + 1
= n(n − 1) / 2

Overall time complexity:

- **O(n²)**

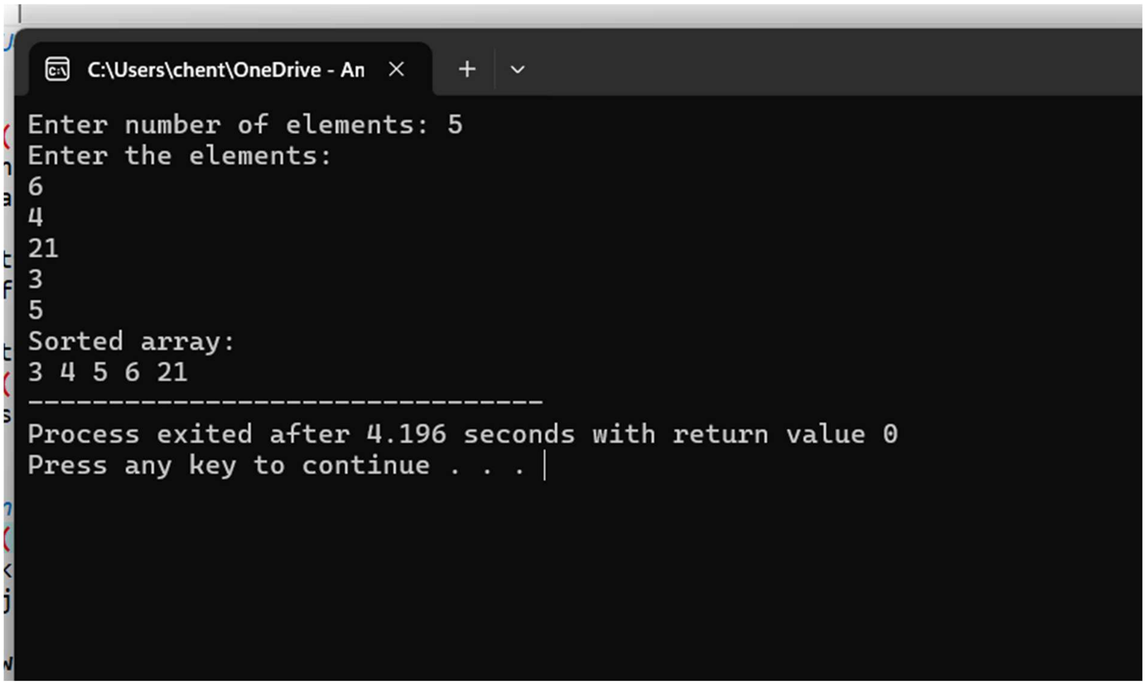
Question 2: INSERTION SORT

CODE:



```c
//CH.SC.U4CSE24103
#include <stdio.h>

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

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

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

    // Insertion Sort
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

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

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

    return 0;
}

```

OUTPUT:



```
Enter number of elements: 5
Enter the elements:
6
4
21
3
5
Sorted array:
3 4 5 6 21
--------------------------------
Process exited after 4.196 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:
Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int key → 4 bytes

- **Array used:**
- int arr[size] → size × 4 bytes

- **Total memory used by variables (excluding input array):**
- 4 + 4 + 4 + 4 = 16 bytes

Time Complexity: O(n^2)

Outer loop:

- Runs (n – 1) times

Inner while loop:

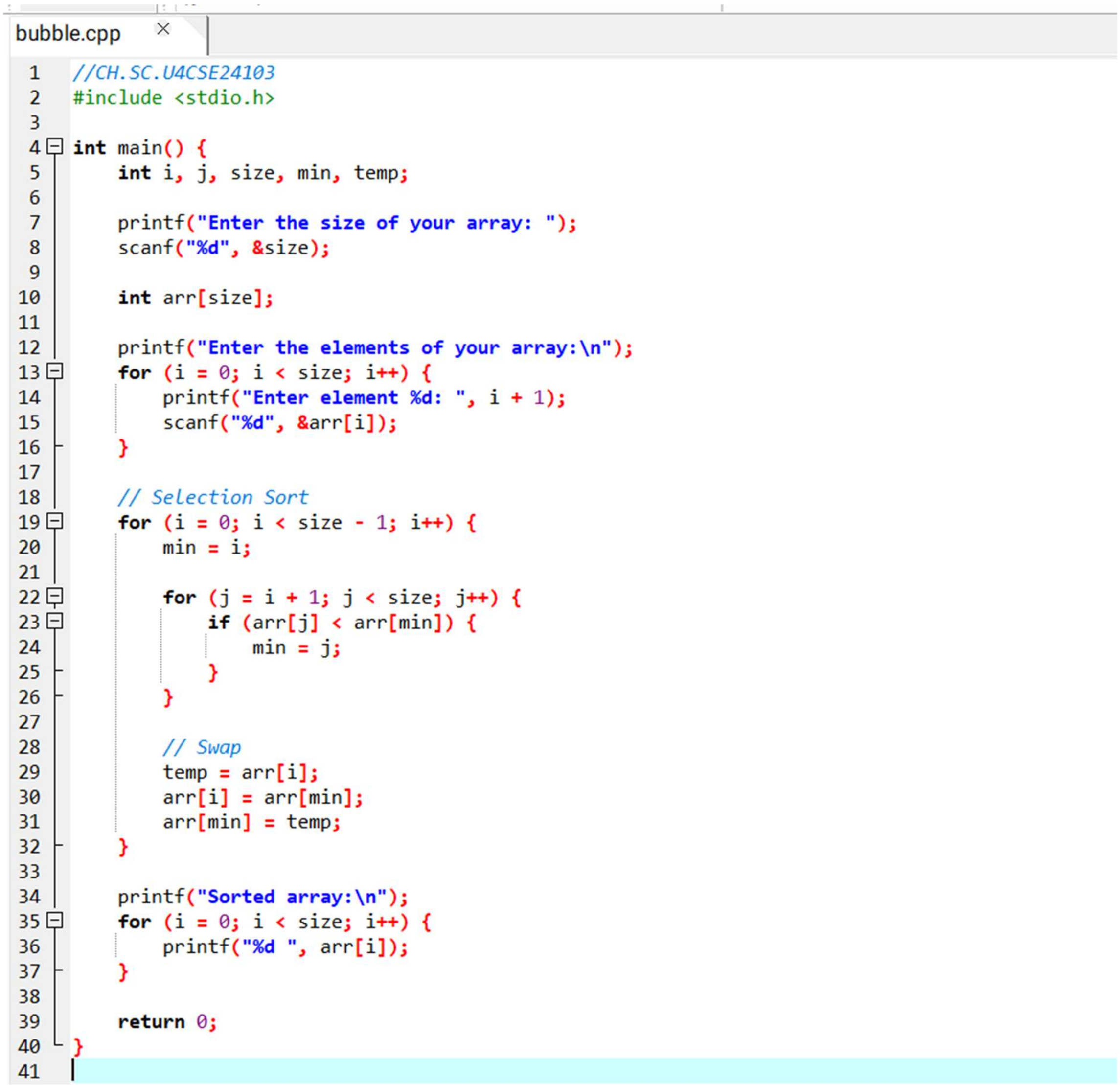
- In the worst case, runs up to i times for each iteration

- **Total number of comparisons (worst case):**
- 1 + 2 + 3 + ... + (n − 1)
- = n(n − 1) / 2

- **Overall time complexity: O(n^2)**

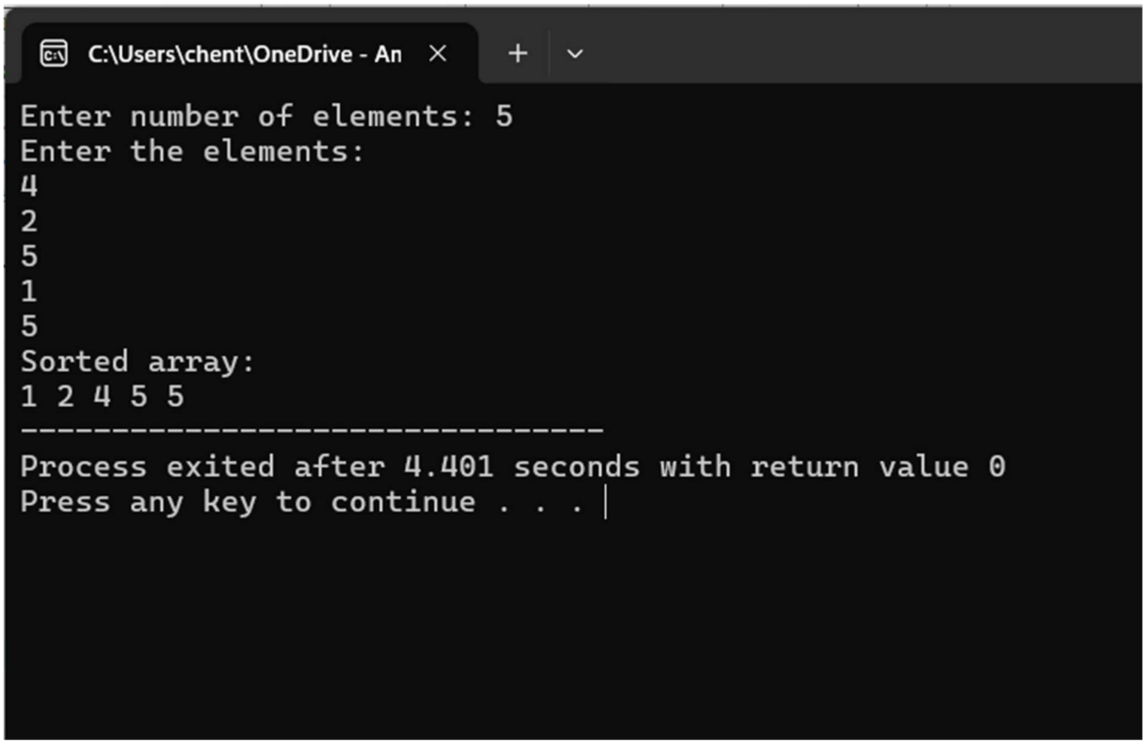
Question 3:

CODE:



```c
//CH.SC.U4CSE24103
#include <stdio.h>

int main() {
    int i, j, size, min, temp;

    printf("Enter the size of your array: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter the elements of your array:\n");
    for (i = 0; i < size; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    // Selection Sort
    for (i = 0; i < size - 1; i++) {
        min = i;

        for (j = i + 1; j < size; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }

        // Swap
        temp = arr[i];
        arr[i] = arr[min];
        arr[min] = temp;
    }

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

    return 0;
}
```

OUTPUT:



```
Enter number of elements: 5
Enter the elements:
4
2
5
1
5
Sorted array:
1 2 4 5 5
--------------------------------
Process exited after 4.401 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION:

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int min → 4 bytes
- int temp → 4 bytes

Array used:

- int arr[size] → size × 4 bytes

Total memory used by variables (excluding input array):

- 4 + 4 + 4 + 4 + 4 = 20 bytes

Auxiliary space:

- Constant → O(1)

Time Complexity: O(n^2)

Outer loop:

- Runs (n − 1) times

Inner loop:

- Runs (n − 1 − i) times for each outer loop iteration

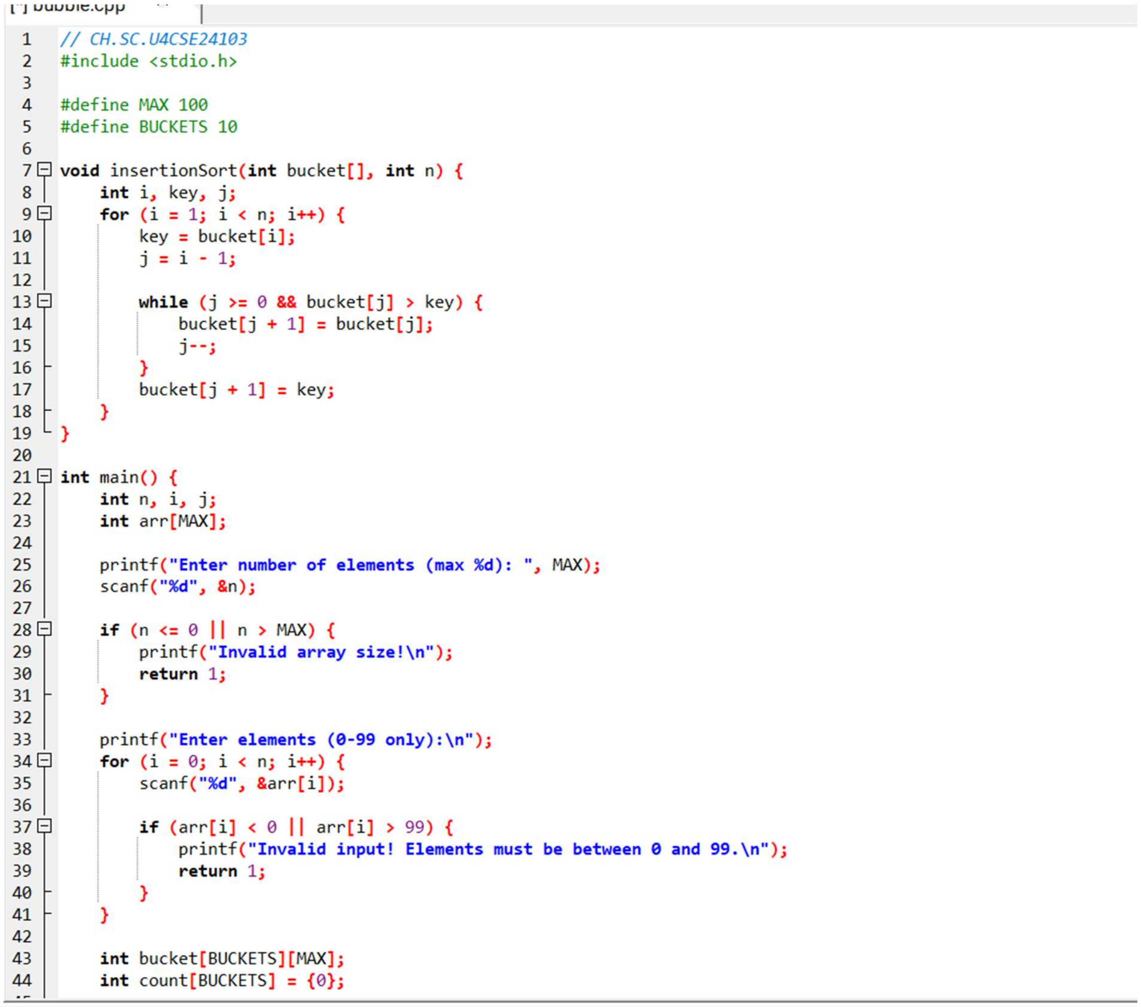
Total number of comparisons:

- (n − 1) + (n − 2) + ... + 1
- = n(n − 1) / 2

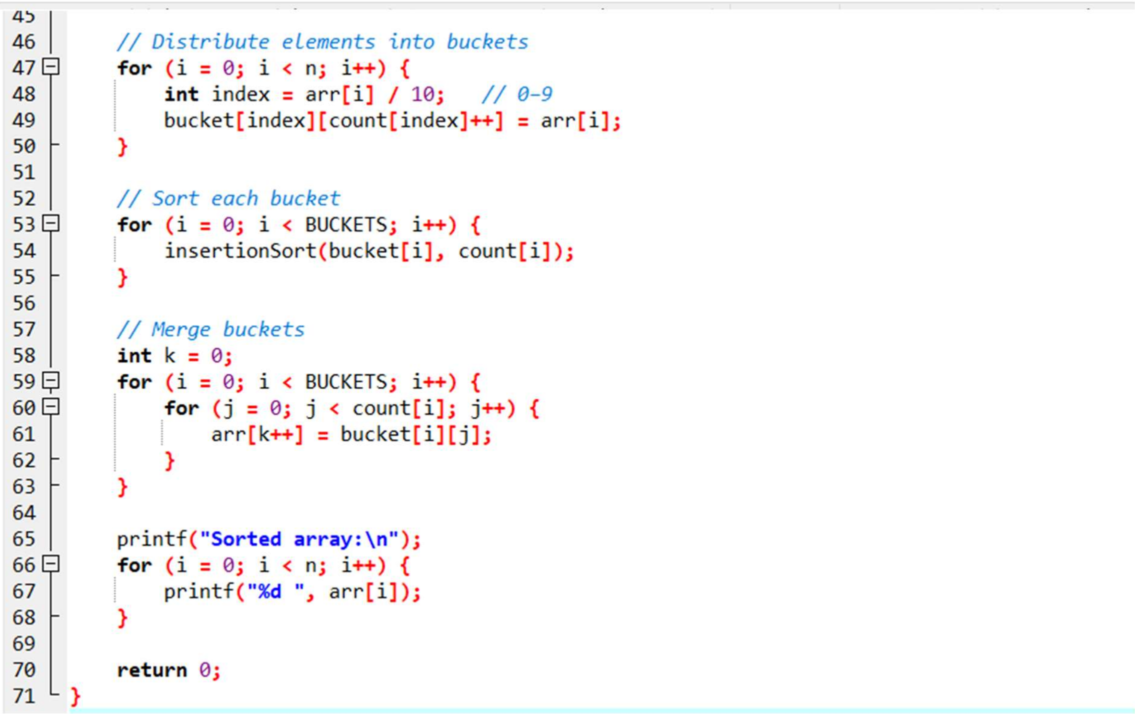
O(n^2)

Question 4: BUCKET SORT

CODE:

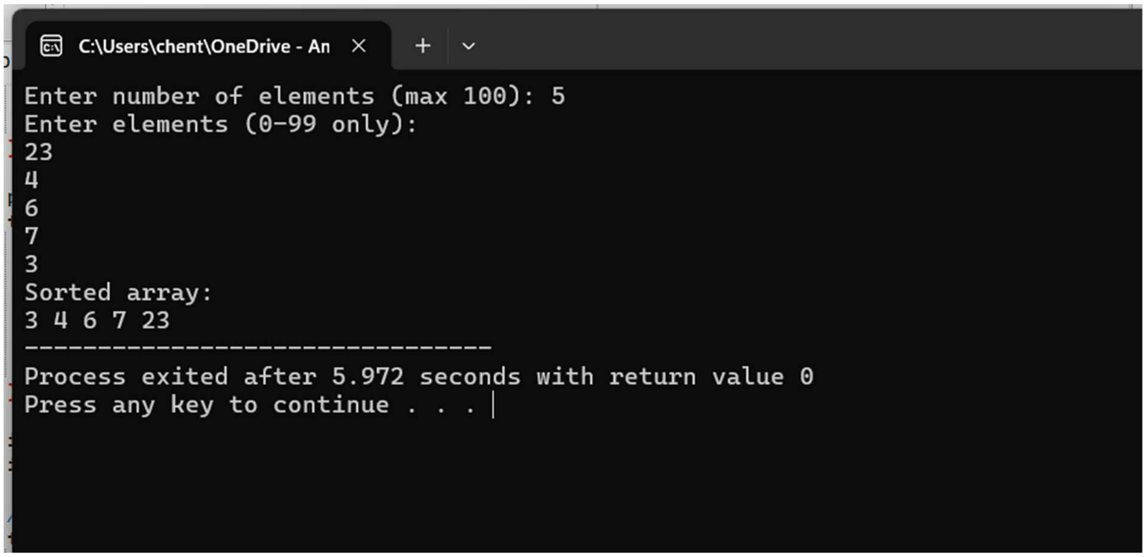


```c
// CH.SC.U4CSE24103
#include <stdio.h>

#define MAX 100
#define BUCKETS 10

void insertionSort(int bucket[], int n) {
    int i, key, j;
    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;
    }
}

int main() {
    int n, i, j;
    int arr[MAX];

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

    if (n <= 0 || n > MAX) {
        printf("Invalid array size!\n");
        return 1;
    }

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

        if (arr[i] < 0 || arr[i] > 99) {
            printf("Invalid input! Elements must be between 0 and 99.\n");
            return 1;
        }
    }

    int bucket[BUCKETS][MAX];
    int count[BUCKETS] = {0};
```



```c

    // Distribute elements into buckets
    for (i = 0; i < n; i++) {
        int index = arr[i] / 10;   // 0-9
        bucket[index][count[index]++] = arr[i];
    }

    // Sort each bucket
    for (i = 0; i < BUCKETS; i++) {
        insertionSort(bucket[i], count[i]);
    }

    // Merge buckets
    int k = 0;
    for (i = 0; i < BUCKETS; i++) {
        for (j = 0; j < count[i]; j++) {
            arr[k++] = bucket[i][j];
        }
    }

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

    return 0;
}
```

OUTPUT:



```
Enter number of elements (max 100): 5
Enter elements (0-99 only):
23
4
6
7
3
Sorted array:
3 4 6 7 23
--------------------------------
Process exited after 5.972 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: O(n + k)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes

Buckets used:

- k buckets (arrays or lists)

Array used:

- int arr[size] → size × 4 bytes

Auxiliary space:

- Buckets require extra memory proportional to number of elements and buckets

Overall space usage:

- Input array → O(n)
- Buckets → O(k)

Total space complexity:

- O(n + k)

Time Complexity

Distribution into buckets:

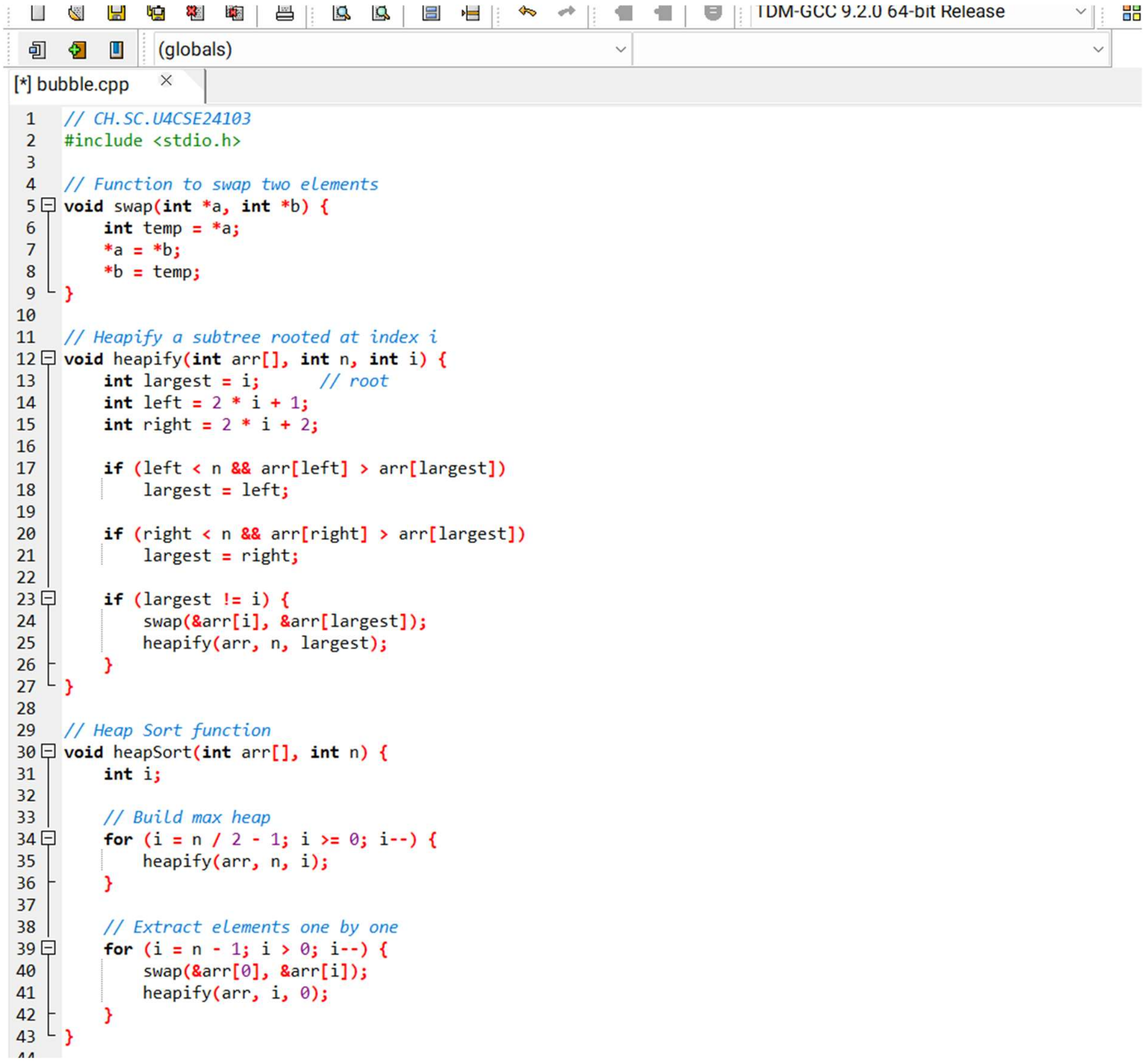
- Each element placed into a bucket once
- Time → O(n)

Overall time complexity:

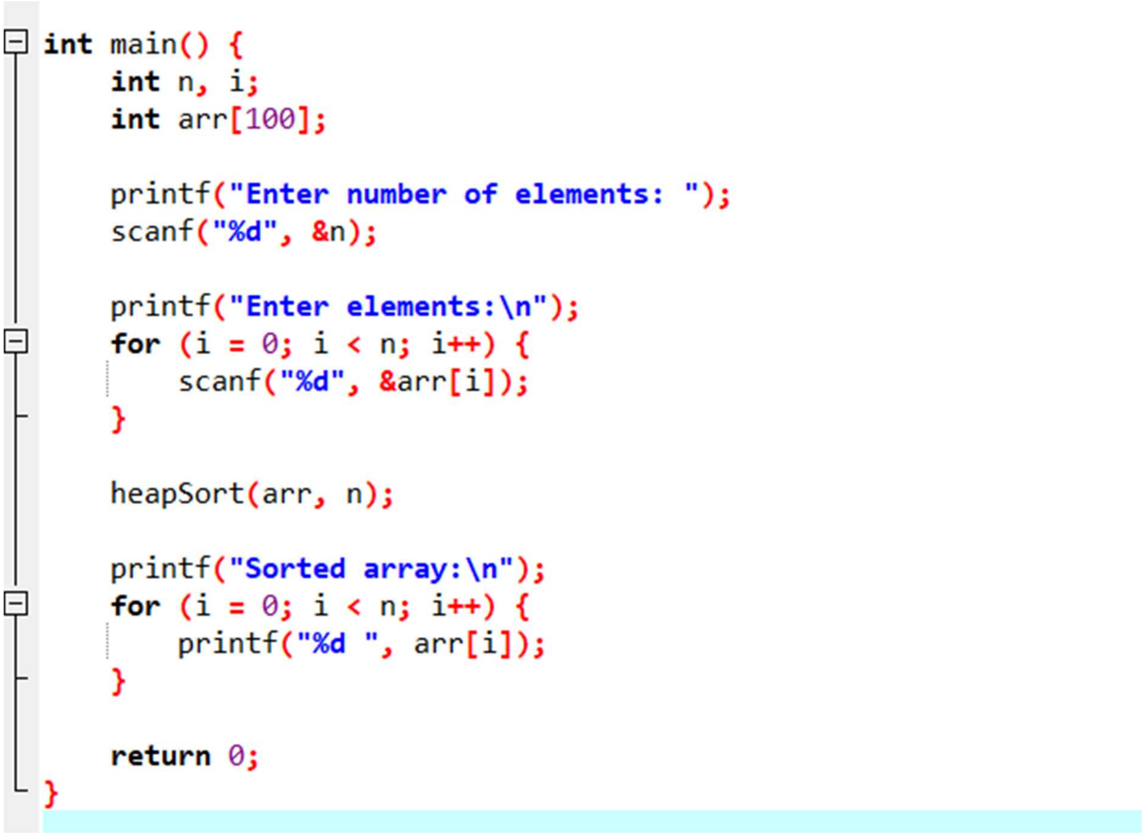
- Best / Average → O(n + k)
- Worst → O(n²)

Question 5: HEAP SORT

CODE:

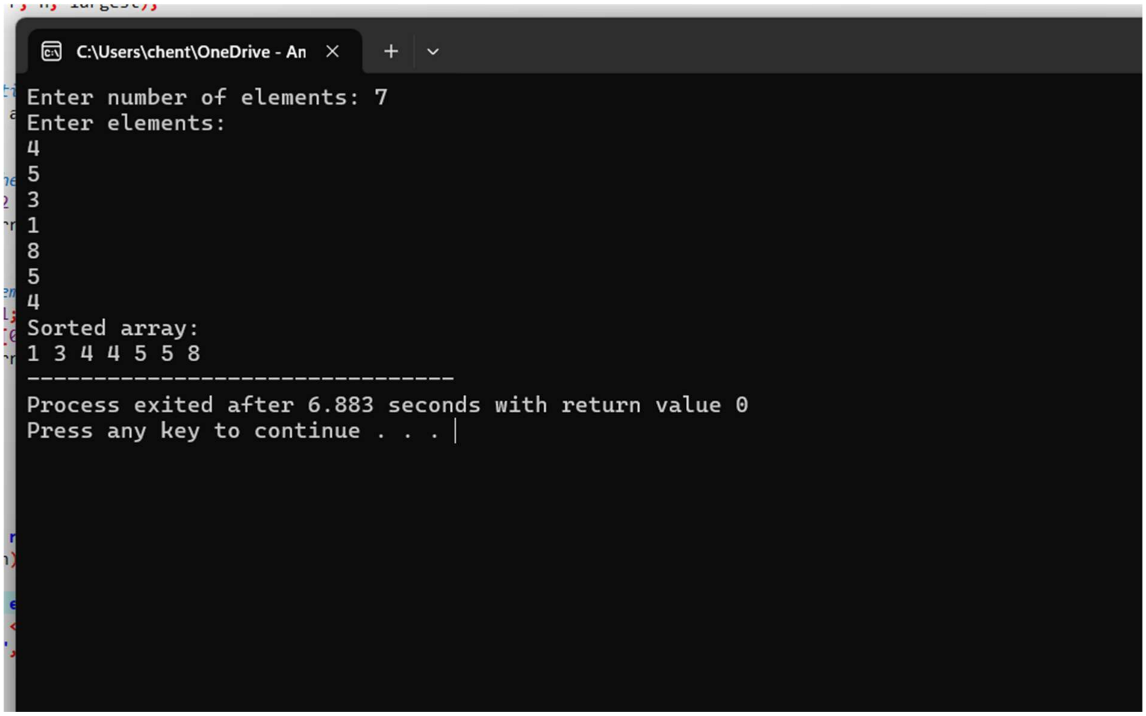


```
// CH.SC.U4CSE24103
#include <stdio.h>

// Function to swap two elements
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Heapify a subtree rooted at index i
void heapify(int arr[], int n, int i) {
    int largest = i;      // root
    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]);
        heapify(arr, n, largest);
    }
}

// Heap Sort function
void heapSort(int arr[], int n) {
    int i;

    // Build max heap
    for (i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // Extract elements one by one
    for (i = n - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}
```



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

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

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

    heapSort(arr, n);

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

    return 0;
}
```

OUTPUT:



```
Enter number of elements: 7
Enter elements:
4
5
3
1
8
5
4
Sorted array:
1 3 4 4 5 5 8
--------------------------------
Process exited after 6.883 seconds with return value 0
Press any key to continue . . . |
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: O(1)

Variables in main() :

- int i → 4 bytes
- int j → 4 bytes
- int size → 4 bytes
- int temp → 4 bytes
- int parent → 4 bytes
- int left → 4 bytes
- int right → 4 bytes
- int largest → 4 bytes

Array used:

- int arr[size] → size × 4 bytes

Total memory used by variables (excluding input array):

- 4 + 4 + 4 + 4 + 4 + 4 + 4 + 4 = 32 bytes

Auxiliary space:

- Constant extra space used
- O(1)

Time Complexity: O(n log n)

Building max heap:

- Heapify runs in O(n) time

Sorting phase:

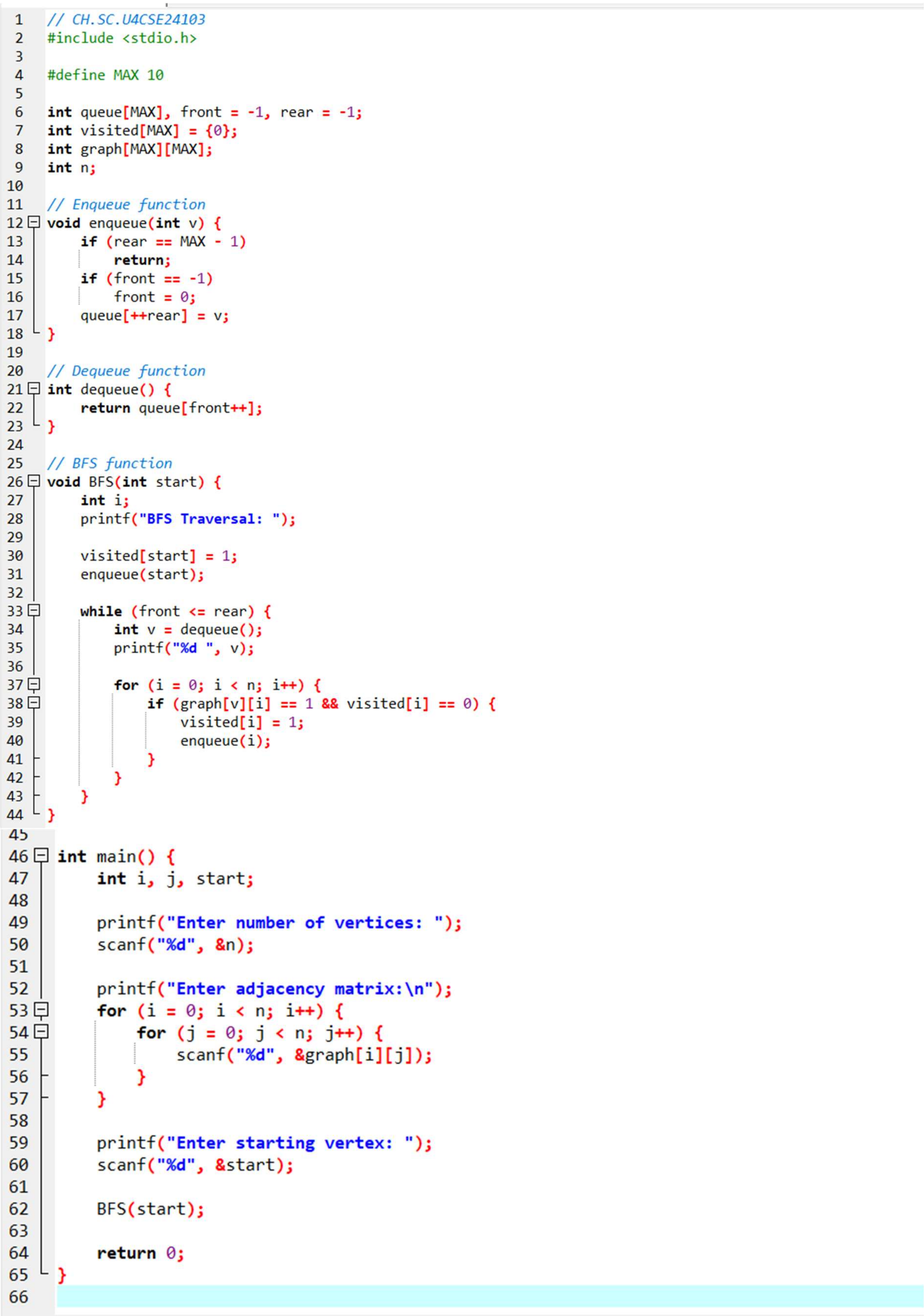
- n – 1 deletions from heap
- Each heapify operation takes O(log n)

Total time:

- O(n) + O(n log n)

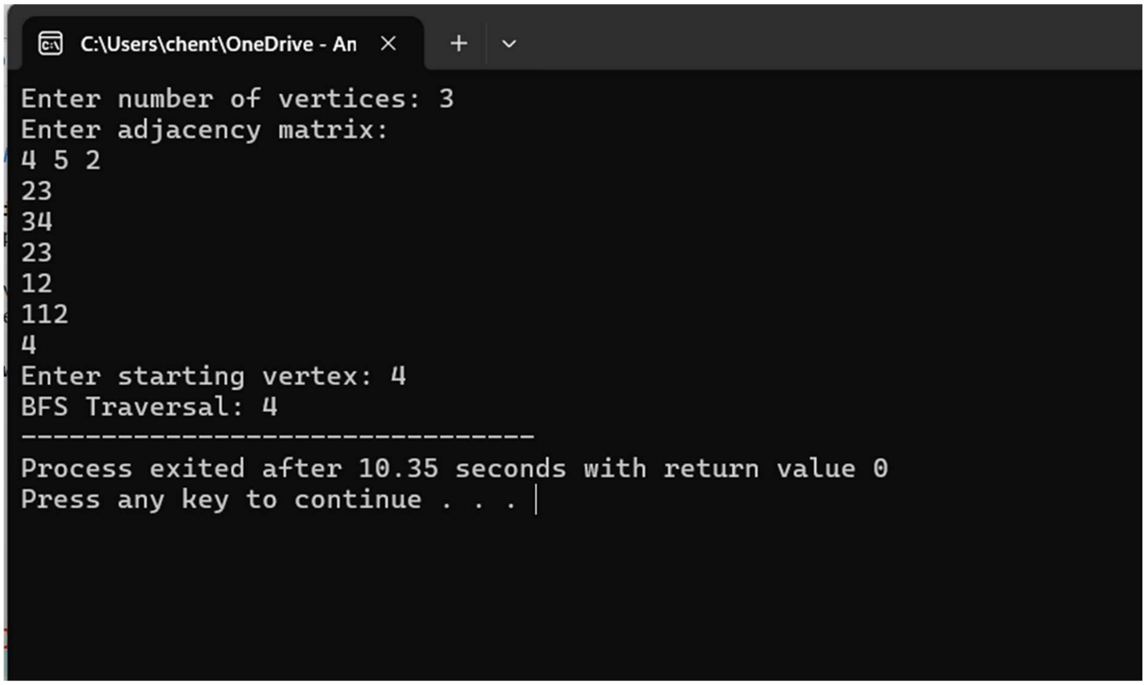
Question 6: BFS

CODE



```c
// CH.SC.U4CSE24103
#include <stdio.h>

#define MAX 10

int queue[MAX], front = -1, rear = -1;
int visited[MAX] = {0};
int graph[MAX][MAX];
int n;

// Enqueue function
void enqueue(int v) {
    if (rear == MAX - 1)
        return;
    if (front == -1)
        front = 0;
    queue[++rear] = v;
}

// Dequeue function
int dequeue() {
    return queue[front++];
}

// BFS function
void BFS(int start) {
    int i;
    printf("BFS Traversal: ");

    visited[start] = 1;
    enqueue(start);

    while (front <= rear) {
        int v = dequeue();
        printf("%d ", v);

        for (i = 0; i < n; i++) {
            if (graph[v][i] == 1 && visited[i] == 0) {
                visited[i] = 1;
                enqueue(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", &graph[i][j]);
        }
    }

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

    BFS(start);

    return 0;
}
```

OUTPUT:



```
Enter number of vertices: 3
Enter adjacency matrix:
4 5 2
23
34
23
12
112
4
Enter starting vertex: 4
BFS Traversal: 4
--------------------------------
Process exited after 10.35 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: O(n)
Variables in main() :

- int n → 4 bytes
- int i, j → 8 bytes
- int start → 4 bytes
- int front, rear → 8 bytes

Arrays used:

- int visited[n] → n × 4 bytes
- int queue[n] → n × 4 bytes
- int graph[n][n] → $n^2 \times 4$ bytes

Auxiliary space:

- Queue + visited array → O(n)
- Adjacency matrix → $O(n^2)$

Overall space complexity:

- $O(n^2)$ (due to adjacency matrix)

Time Complexity: $O(n^2)$

Traversal logic:

- Each vertex is visited once

Adjacency matrix scan:

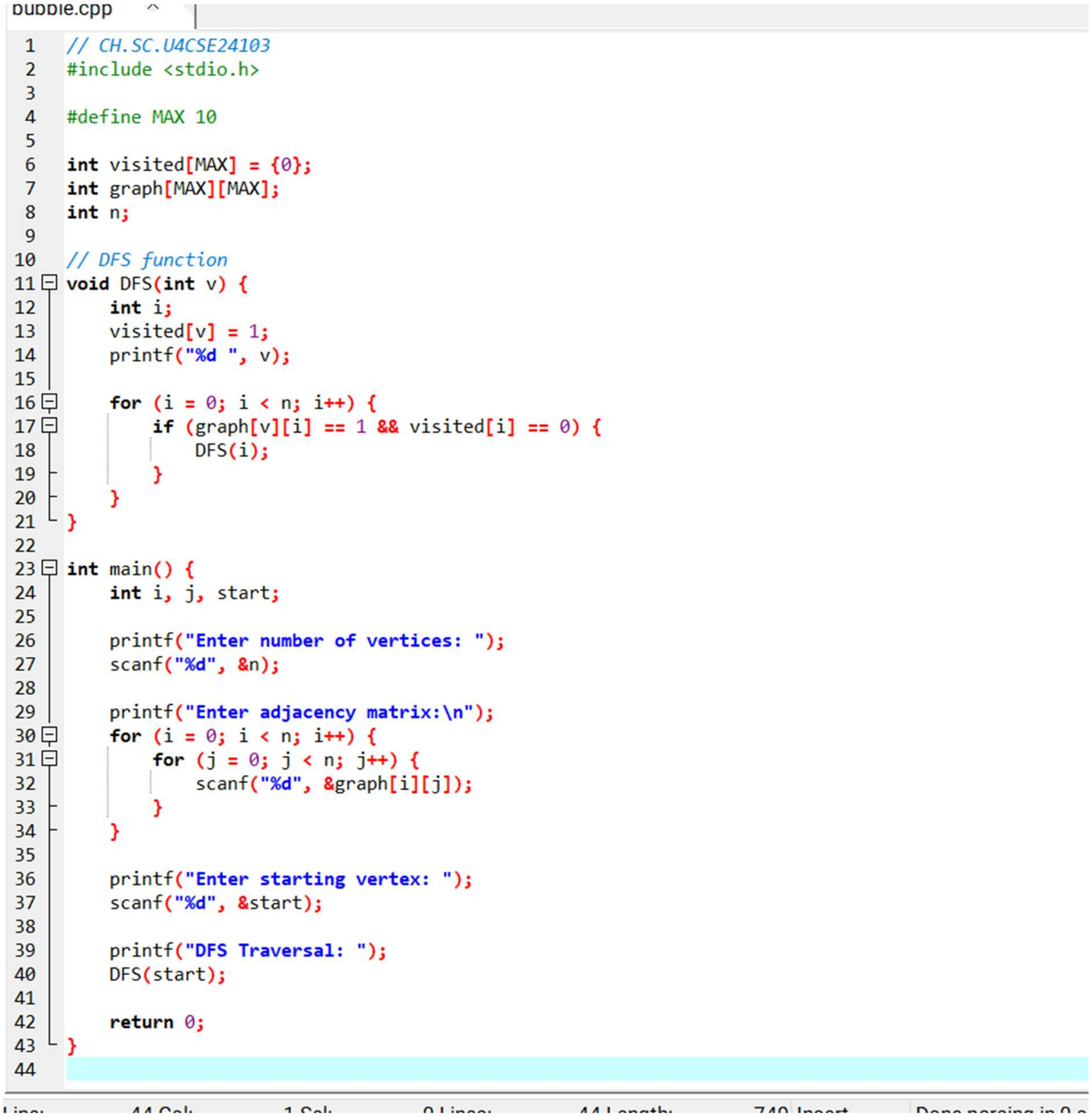
- For each vertex, all n vertices are checked

Total operations:

- n × n comparisons

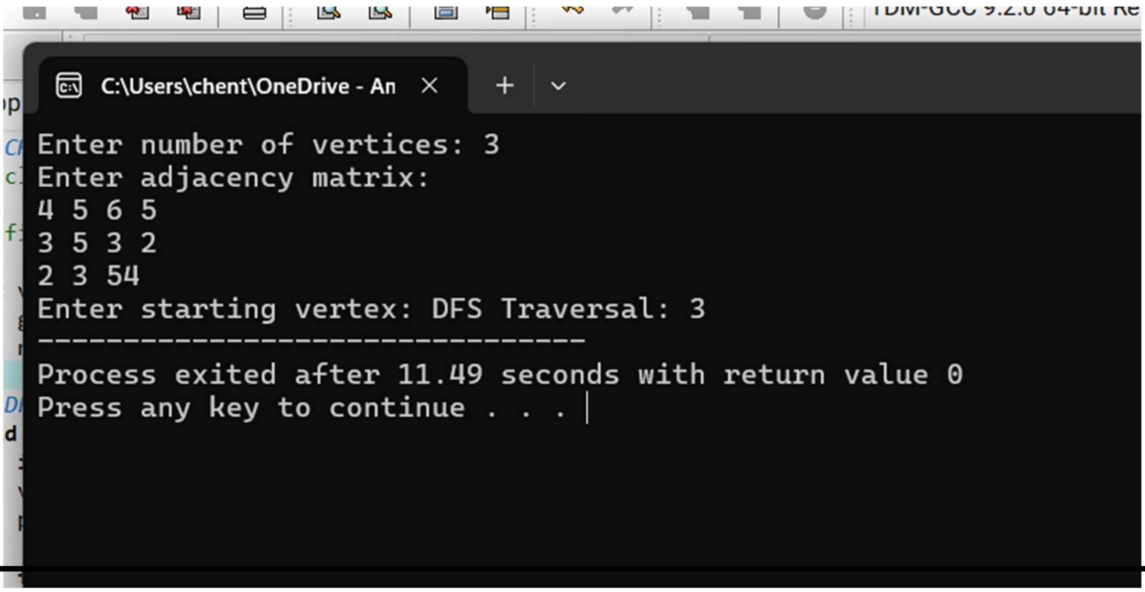
Question 7: DFS

CODE:



```c
// CH.SC.U4CSE24103
#include <stdio.h>

#define MAX 10

int visited[MAX] = {0};
int graph[MAX][MAX];
int n;

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

    for (i = 0; i < n; i++) {
        if (graph[v][i] == 1 && visited[i] == 0) {
            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", &graph[i][j]);
        }
    }

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

    printf("DFS Traversal: ");
    DFS(start);

    return 0;
}
```

OUTPUT:



```
Enter number of vertices: 3
Enter adjacency matrix:
4 5 6 5
3 5 3 2
2 3 54
Enter starting vertex: DFS Traversal: 3
--------------------------------
Process exited after 11.49 seconds with return value 0
Press any key to continue . . .
```

TIME AND SPACE COMPLEXITY JUSTIFICATION

Space Complexity: $O(n^2)$

Variables used:

- int n → 4 bytes
- int i, j → 8 bytes
- int start → 4 bytes

Arrays used:

- int visited[n] → n × 4 bytes
- int graph[n][n] → n^2 × 4 bytes

Recursion stack:

- Maximum depth = n
- Stack space → $O(n)$

Overall space complexity:

- $O(n^2)$ (dominant adjacency matrix)

Time Complexity: $O(n^2)$

Traversal logic:

- Each vertex visited once

Adjacency matrix scan:

- For each vertex, all n vertices are checked

Total operations:

- n × n comparisons