NAME:

Bhogapurapu sumitranand

ROLL NO:

CH.SC.U4CSE24162

(Design and Analysis of algorithms)

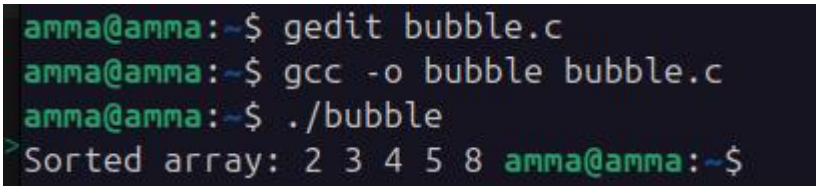
WEEk2:-

1Q) BUBBLE SORT

CODE:

```c
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    int swapped;

    for (i = 0; i < n - 1; i++) {
        swapped = 0;
        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;
                swapped = 1;
            }
        }

        if (swapped == 0)
            break;
    }
}

int main() {
    int arr[] = {5, 3, 8, 4, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

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

    return 0;
}
```

OUTPUT:



TIME COMPLEXITY:- O(n²)

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>
#include <stdlib.h>

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

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

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

    for (i = 0; i < n; i++) {
        int bucketIndex = arr[i] * n;
        buckets[bucketIndex][index[bucketIndex]++] = arr[i];
    }

    for (i = 0; i < n; i++)
        insertionSort(buckets[i], index[i]);

    int pos = 0;
    for (i = 0; i < n; i++) {
```

```
    for (i = 0; i < n; i++) {
        for (j = 0; j < index[i]; j++) {
            arr[pos++] = buckets[i][j];
        }
    }
}

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]);

    bucketSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%0.2f ", arr[i]);

    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit insertion.c
amma@amma:~$ gcc -o insertion insertion.c
amma@amma:~$ ./insertion
Sorted array: 2 3 4 5 8 amma@amma:~$
```

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 arr[], int n) {
    int i, j, minIndex, temp;

    for (i = 0; i < n - 1; i++) {
        minIndex = i;

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

        if (minIndex != i) {
            temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

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

    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit selection.c
amma@amma:~$ gcc -o selection selection.c
amma@amma:~$ ./selection
Sorted array: 11 12 22 25 64 amma@amma:~$
```

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:

```
amma@amma:~$ gedit bucket.c
amma@amma:~$ gcc -o bucket bucket.c
amma@amma:~$ ./bucket
Sorted array: 0.23 0.25 0.32 0.42 0.47 0.51 0.52 amma@amma:~$
```

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 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);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

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

    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit heap.c
amma@amma:~$ gcc -o heap.c
gcc: fatal error: no input files
compilation terminated.
amma@amma:~$ gcc -o heap heap.c
amma@amma:~$ ./heap
Sorted array: 5 6 7 11 12 13 amma@amma:~$
```

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