NAME : NELLI PREETHI JASMINE

ROLL NO: CH.SC.U4CSE24132

DAA-2,3

BUBBLE SORTING:

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 arr[] = {1, 2, 3,7,4,0,1};
    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:

```
root@amma54:/home/amma/Downloads# gcc -o bubbleSort bubbleSort.c
root@amma54:/home/amma/Downloads# ./bubbleSort
Sorted array: 0 1 1 2 3 4 7 root@amma54:/home/amma/Downloads#
```

- Time complexity for bubble sorting is O(n^2) because of nested loops comparing elements.
- Space complexity for bubble sorting is O(1) because in place sorting only a temp variable for swapping.

Insertion Sorting:

Code:

```c
#include <stdio.h>
void insertionSort(int arr[], int n) {
    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;
    }
}

int main() {
    int arr[] = {15, 10, 12, 7, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
root@amma54:/home/amma/Downloads# gcc -o insertSort insertSort.c
root@amma54:/home/amma/Downloads# ./insertSort
Sorted array: 3 7 10 12 15 root@amma54:/home/amma/Downloads#
```

- Time complexity for Insertion sorting is O(n^2) because shifting elements takes time.
- Space complexity for insertion sorting is O(1) because in place sorting only a temp variable for swapping.

Selection Sorting:

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;
            }
        }
        temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
int main() {
    int arr[] = {54, 15, 2, 12, 1};
    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:

```
root@amma54:/home/amma/Downloads# gcc -o selectionSort selectionSort.c
root@amma54:/home/amma/Downloads# ./selectionSort
Sorted array: 1 2 12 15 54 root@amma54:/home/amma/Downloads#
```

- Time complexity for Selection sorting is O(n^2) because it always scans the remaining array to find minimum.
- Space complexity for Selection sorting is O(1) because in place sorting only a temp variable for swapping.

Bucket Sorting :

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10
void insertionSort(float arr[], int n) {
    for (int i = 1; i < n; i++) {
        float key = arr[i];
        int 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) {
    float buckets[N][n];
    int bucketCount[N];
    for (int i = 0; i < N; i++)
        bucketCount[i] = 0;
    for (int i = 0; i < n; i++) {
        int bucketIndex = (int)(arr[i] * N);
        buckets[bucketIndex][bucketCount[bucketIndex]++] = arr[i];
    }
    for (int i = 0; i < N; i++) {
        if (bucketCount[i] > 0)
            insertionSort(buckets[i], bucketCount[i]);
    }
    int idx = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < bucketCount[i]; j++) {
```

```
            arr[idx++] = buckets[i][j];
        }
    }
}
int main() {
    float arr[] = {0.68, 0.07, 0.29, 0.16, 0.62, 0.84, 0.11, 0.02};
    int n = sizeof(arr) / sizeof(arr[0]);
    bucketSort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%f ", arr[i]);
    return 0;
}
```

Output:

```
root@amma54:/home/amma/Downloads# gcc -o bucketSort bucketSort.c
root@amma54:/home/amma/Downloads# ./bucketSort
Sorted array: 0.020000 0.070000 0.110000 0.160000 0.290000 0.620000 0.680000 0.8
40000 root@amma54:/home/amma/Downloads#
```

- Time complexity for Bucket sorting is O(n+k) because all elements go into one bucket where n is number of elements and k is number of buckets.
- Space complexity for Selection sorting is O(n+k) because extra buckets are used.

Heap Sorting :

Max-Heap:

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]);
```

```c
    int n = sizeof(arr)/sizeof(arr[0]);
    heapSort(arr, n);
    printf("Ascending: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
}
```

Output:

```
40000 root@amma54:/home/amma/Downloads# gcc -o maxHeapSort maxHeapSort.c
root@amma54:/home/amma/Downloads# ./maxHeapSort
Ascending: 5 6 7 11 12 13 root@amma54:/home/amma/Downloads#
```

Min-heap:

```c
#include <stdio.h>
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;
    if (right < n && arr[right] < arr[smallest])
        smallest = right;
    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;
        minHeapify(arr, n, smallest);
    }
}
void heapSortDescending(int arr[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        minHeapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        minHeapify(arr, i, 0);
    }
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    heapSortDescending(arr, n);
    printf("Descending: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
}
```

Output:

```
root@amma54:/home/amma/Downloads# gcc -o minHeapSort minHeapSort.c
root@amma54:/home/amma/Downloads# ./minHeapSort
Descending: 13 12 11 7 6 5 root@amma54:/home/amma/Downloads#
```

- Time complexity for Heap sorting is O(n log n) because it takes log n time.
- Space complexity for Heap sorting is O(1) because in place sorting only a temp variable for swapping.

DFS:

Code:

```
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX];
int visited[MAX];
int n;
void DFS(int v) {
    visited[v] = 1;
    printf("%d ", v);
    for (int i = 0; i < n; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            DFS(i);
        }
    }
}
int main() {
    int edges, u, v;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            adj[i][j] = 0;
    printf("Enter edges (u v):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }
    printf("\nDFS Traversal starting from 0:\n");
```

```
    DFS(0);
    return 0;
}
```

Output:

```
0 root@amma29:/home/amma/Downloads# gcc -o dfs dfs.c
root@amma29:/home/amma/Downloads# ./dfs
Enter number of vertices: 5
Enter number of edges: 4
Enter edges (u v):
0 1
0 2
1 3
1 4

DFS Traversal starting from 0:
0 1 3 4 2 root@amma29:/home/amma/Downloads#
```

- Time complexity for DFS is O(V+E) because every vertex and edge is visited once.
- Space complexity for DFS is O(V) because recursive stack or explicit stack can hold upto v nodes.

BFS:

Code:

```c
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = 0, rear = 0;
int n;
void BFS(int start) {
    visited[start] = 1;
    queue[rear++] = start;
    while (front != rear) {
        int v = queue[front++];
        printf("%d ", v);
        for (int i = 0; i < n; i++) {
            if (adj[v][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
}
int main() {
    int edges, u, v;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
```

```c
            adj[i][j] = 0;
    printf("Enter edges (u v):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
        adj[u][v] = adj[v][u] = 1;
    }
    printf("\nBFS Traversal starting from 0:\n");
    BFS(0);
    return 0;
}
```

Output:

```
root@amma29:/home/amma/Downloads# gcc -o bfs bfs.c
root@amma29:/home/amma/Downloads# ./bfs
Enter number of vertices: 5
Enter number of edges: 4
Enter edges (u v):
0 1
0 2
1 3
1 4

BFS Traversal starting from 0:
0 1 2 3 4 root@amma29:/home/amma/Downloads#
```

- Time complexity for DFS is O(V+E) because every vertex and edge is visited once.
- Space complexity for DFS is O(V) because queue and visited array store vertices.