NAME:CH.JYOTHIRMAI

ROLL NO:CH.SC.U4CSE24107

DAA WEEK-2

1.Bubble Sort

Code:

```c
//bubble sort
#include <stdio.h>
int main() {
int a[] = {5, 2, 8, 1, 3};
int n = 5;
int i, j, temp;
for(i = 0; i < n - 1; i++) {
for(j = 0; j < n - i - 1; j++) {
if(a[j] > a[j + 1]) {
temp = a[j];
a[j] = a[j + 1];
 a[j + 1] = temp;
 }
}
}
printf("Bubble Sorted: ");
for(i = 0; i < n; i++)
printf("%d ", a[i]);
return 0;
}
```

Output:

```
0 0 256 amma@amma05:~/Documents$ gcc bubble.c -o bubble
amma@amma05:~/Documents$ ./bubble
Bubble Sorted: 1 2 3 5 8 amma@amma05:~/Documents$
```

Time Complexity:

Compares adjacent elements and swaps them. Repeats until sorted.

Time: Best O(n), Average/Worst $O(n^2)$

2.Insertion Sort

Code:

```c
//insertion sort
#include <stdio.h>
int main() {
    int a[] = {5, 2, 8, 1, 3};
    int n = 5;
    int i, key, j;
    for(i = 1; i < n; i++) {
        key = a[i];
        j = i - 1;
        while(j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
    printf("Insertion Sorted: \n");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
n
amma@amma05:~/Documents$ ./insertion
Insertion Sorted:
1 2 3 5 8 amma@amma05:~/Documents$
```

Time complexity:

Inserts each element into its correct position in the sorted part.

Time: Best O(n), Average/Worst $O(n^2)$

3.Selection Sort

Code:

```c
//selection
#include <stdio.h>
int main() {
    int a[] = {5, 2, 8, 1, 3};
    int n = 5;
    int i, j, min, temp;

    for(i = 0; i < n - 1; i++) {
        min = i;
        for(j = i + 1; j < n; j++) {
            if(a[j] < a[min])
                min = j;
        }
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }

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

    return 0;
}
```

Output:

```
Insertion Sorted:
1 2 3 5 8 amma@amma05:~/Documents$ gcc selection.c -o selection
amma@amma05:~/Documents$ ./selection
Selection Sorted: 1 2 3 5 8 amma@amma05:~/Documents$
```

Time Complexity:

Inserts each element into its correct position in the sorted part.

Time: Best **O(n)**, Average/Worst **$O(n^2)$**

4.Heap Sort:

Min heap:

Code:

```c
//min heap
#include <stdio.h>
#define MAX 10
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;
    }
    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--) {
```

```c
}
void heapSort(int arr[], int n) {
    buildMinHeap(arr, n);
    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);
        minHeapify(arr, i, 0);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original Array: \n");
    printArray(arr, n);
    heapSort(arr, n);
    printf("Sorted Array using Min-Heap: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
1 2 2 3 3 4 8 jyothirmai@jyothirmai:~/Documents/main$ gcc min.c -o min
jyothirmai@jyothirmai:~/Documents/main$ ./min
Original Array:
4 10 3 5 1
Sorted Array using Min-Heap:
10 5 4 3 1
```

Time Complexity:

Uses a heap data structure to repeatedly remove the root element.

Time: Always **O(n log n)**

Max Heap:

Code:

```c
//max heap
#include <stdio.h>
#define MAX 10
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--) {
```

```c
}
void heapSort(int arr[], int n) {
    buildMaxHeap(arr, n);
    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);
        maxHeapify(arr, i, 0);
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {4, 10, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original Array: \n");
    printArray(arr, n);
    heapSort(arr, n);
    printf("Sorted Array using Max-Heap: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
jyothirmai@jyothirmai:~/Documents/main$ gcc max.c -o max
jyothirmai@jyothirmai:~/Documents/main$ ./max
Original Array:
4 10 3 5 1
Sorted Array using Max-Heap:
1 3 4 5 10
```

Time Complexity:

Uses a heap data structure to repeatedly remove the root element.

Time: Always O(n log n)

5.Bucket Sort:

Code:

```c
//bucket sort
#include <stdio.h>
void bucketSort(int arr[], int n) {
    int bucket[10] = {0};
    for (int i = 0; i < n; i++) {
        bucket[arr[i]]++;
    }
    int index = 0;
    for (int i = 0; i < 10; i++) {
        while (bucket[i] > 0) {
            arr[index++] = i;
            bucket[i]--;
        }
    }
}
int main() {
    int arr[] = {4, 2, 2, 8, 3, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    bucketSort(arr, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
jyothirmai@jyothirmai:~/Documents/main$ gcc bucket.c -o bucket
jyothirmai@jyothirmai:~/Documents/main$ ./bucket
1 2 2 3 3 4 8 jyothirmai@jyothirmai:~/Documents/main$
```

Time Complexity:

Distributes elements into buckets, sorts them, then combines.

Time: Best/Average O(n + k), Worst O(n^2)

6.BFS:

Code:

```c
//bfs
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int adj[MAX][MAX] = {
    {0, 1, 1, 0, 0, 0},
    {1, 0, 1, 1, 0, 0},
    {1, 1, 0, 0, 1, 0},
    {0, 1, 0, 0, 1, 1},
    {0, 0, 1, 1, 0, 1},
    {0, 0, 0, 1, 1, 0}
};
void BFS(int start) {
    bool visited[MAX] = {false};
    int queue[MAX], front = -1, rear = -1;
    visited[start] = true;
    queue[++rear] = start;
    while (front != rear) {
        int node = queue[++front];
        printf("Visited %d\n", node);
        for (int i = 0; i < MAX; i++) {
            if (adj[node][i] == 1 && !visited[i]) {
                visited[i] = true;
                queue[++rear] = i;
```

```
                queue[++rear] = i;
            }
        }
    }
}
int main() {
    printf("BFS starting from node 0:\n");
    BFS(0);
    return 0;
}
```

Output:

```
jyothirmai@jyothirmai:~/Documents/main$ gcc bfs.c -o bfs
jyothirmai@jyothirmai:~/Documents/main$ ./bfs
BFS starting from node 0:
Visited 0
Visited 1
Visited 2
Visited 3
Visited 4
Visited 5
```

Time Complexity:

Explores graph level by level using a queue.

Time: O(V + E)

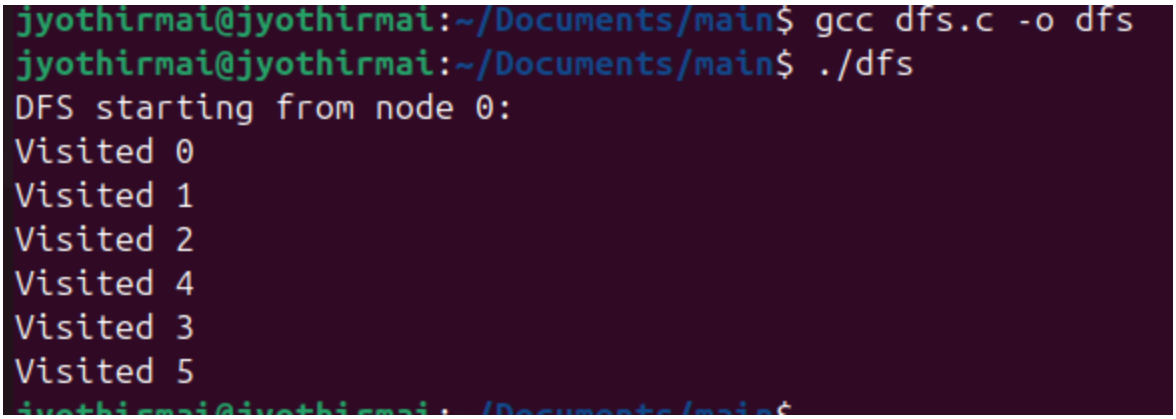
7.DFS:

Code:

```c
//dfs
#include <stdio.h>
#include <stdbool.h>
#define MAX 6
int adj[MAX][MAX] = {
    {0, 1, 1, 0, 0, 0},
    {1, 0, 1, 1, 0, 0},
    {1, 1, 0, 0, 1, 0},
    {0, 1, 0, 0, 1, 1},
    {0, 0, 1, 1, 0, 1},
    {0, 0, 0, 1, 1, 0}
};
void DFS(int node, bool visited[]) {
    visited[node] = true;
    printf("Visited %d\n", node);

    for (int i = 0; i < MAX; i++) {
        if (adj[node][i] == 1 && !visited[i]) {
            DFS(i, visited);
        }
    }
}
int main() {
    bool visited[MAX] = {false};
    printf("DFS starting from node 0:\n");
    DFS(0, visited);
    return 0;
```

Output:



Time Complexity:

Goes deep into a graph before backtracking.

Time: O(V + E)