

NAME:

UDATHA VENKATA ACHYUTH KRISHNA

ROLL NO:

CH.SC.U4CSE24148

(Design and Analysis of algorithms)

WEEK2:-

1Q) BUBBLE SORT

CODE:

```
#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:

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

TIME COMPLEXITY:- $O(n^2)$

JUSTIFICATION:-

Outer loop runs n times.

Inner loop runs up to n times.

Total operations $\approx n \times n$.

SPACE COMPLEXITY:- $O(1)$

JUSTIFICATION:-

int $n \rightarrow 4$ bytes

int $i \rightarrow 4$ bytes

int $j \rightarrow 4$ bytes

int temp $\rightarrow 4$ bytes

Only constant extra space is used.

2Q) INSERTION SORT

CODE:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void insertionSort(float arr[], int n) {
5     int i, j;
6     float key;
7     for (i = 1; i < n; i++) {
8         key = arr[i];
9         j = i - 1;
10        while (j >= 0 && arr[j] > key) {
11            arr[j + 1] = arr[j];
12            j--;
13        }
14        arr[j + 1] = key;
15    }
16 }
17
18 void bucketSort(float arr[], int n) {
19     int i, j, index[n];
20     float buckets[n][n];
21
22     for (i = 0; i < n; i++)
23         index[i] = 0;
24
25     for (i = 0; i < n; i++) {
26         int bucketIndex = arr[i] * n;
27         buckets[bucketIndex][index[bucketIndex]++] = arr[i];
28     }
29
30     for (i = 0; i < n; i++)
31         insertionSort(buckets[i], index[i]);
32
33     int pos = 0;
34     for (i = 0; i < n; i++) {
```

```

34     for (i = 0; i < n; i++) {
35         for (j = 0; j < index[i]; j++) {
36             arr[pos++] = buckets[i][j];
37         }
38     }
39 }
40
41 int main() {
42     float arr[] = {0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51};
43     int n = sizeof(arr) / sizeof(arr[0]);
44
45     bucketSort(arr, n);
46
47     printf("Sorted array: ");
48     for (int i = 0; i < n; i++)
49         printf("%.2f ", arr[i]);
50
51     return 0;
52 }
53

```

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 $\approx n \times n$.

SPACE COMPLEXITY:- $O(1)$

JUSTIFICATION:-

int n \rightarrow 4 bytes

int i \rightarrow 4 bytes

int j \rightarrow 4 bytes

int key \rightarrow 4 bytes

Only constant extra space is used.

3Q) SELECTION SORT

CODE:

```

1 #include <stdio.h>
2
3 void selectionSort(int arr[], int n) {
4     int i, j, minIndex, temp;
5
6     for (i = 0; i < n - 1; i++) {
7         minIndex = i;
8
9         for (j = i + 1; j < n; j++) {
10             if (arr[j] < arr[minIndex]) {
11                 minIndex = j;
12             }
13         }
14
15         if (minIndex != i) {
16             temp = arr[i];
17             arr[i] = arr[minIndex];
18             arr[minIndex] = temp;
19         }
20     }
21 }
22
23 int main() {
24     int arr[] = {64, 25, 12, 22, 11};
25     int n = sizeof(arr) / sizeof(arr[0]);
26
27     selectionSort(arr, n);
28
29     printf("Sorted array: ");
30     for (int i = 0; i < n; i++)
31         printf("%d ", arr[i]);
32
33     return 0;
34 }

```

OUTPUT:

```

Sorted array: 11 12 22 25 64
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 $\approx n \times n$.

SPACE COMPLEXITY:- $O(1)$

JUSTIFICATION:-

int n \rightarrow 4 bytes

int i \rightarrow 4 bytes

int j \rightarrow 4 bytes

int key \rightarrow 4 bytes

Only constant extra space is used.

4Q) BUCKET SORT

CODE:


```

#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:~$
amma@amma:~$

```

TIME COMPLEXITY:- $O(n + k)$

JUSTIFICATION:-

Elements are distributed into k buckets.

Each bucket is sorted individually.

Average case operations $\approx n + k$.

SPACE COMPLEXITY:- $O(n + k)$

JUSTIFICATION:-

Buckets array \rightarrow extra space

int $n \rightarrow 4$ bytes

int $i \rightarrow 4$ bytes

Additional space for buckets is required.

5Q) HEAP SORT

CODE:

```
1 #include <stdio.h>
2
3 void heapify(int arr[], int n, int i) {
4     int largest = i;
5     int left = 2 * i + 1;
6     int right = 2 * i + 2;
7
8
9     if (left < n && arr[left] > arr[largest])
10         largest = left;
11
12
13     if (right < n && arr[right] > arr[largest])
14         largest = right;
15
16
17     if (largest != i) {
18         int temp = arr[i];
19         arr[i] = arr[largest];
20         arr[largest] = temp;
21
22         heapify(arr, n, largest);
23     }
24 }
25
26 void heapSort(int arr[], int n) {
27
28     for (int i = n / 2 - 1; i >= 0; i--)
29         heapify(arr, n, i);
30
31
32     for (int i = n - 1; i > 0; i--) {
33         int temp = arr[0];
34         arr[0] = arr[i];
35         arr[i] = temp;
36
```

```

36
37     heapify(arr, i, 0);
38 }
39 }
40
41 int main() {
42     int arr[] = {12, 11, 13, 5, 6, 7};
43     int n = sizeof(arr)/sizeof(arr[0]);
44
45     heapSort(arr, n);
46
47     printf("Sorted array: ");
48     for (int i = 0; i < n; i++)
49         printf("%d ", arr[i]);
50
51     return 0;
52 }
53

```

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 $\approx n \log n$.

SPACE COMPLEXITY:- $O(1)$

JUSTIFICATION:-

int n \rightarrow 4 bytes

int i \rightarrow 4 bytes

int temp \rightarrow 4 bytes

Sorting is done in-place.

6Q)BFS

CODE:

```

#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 $\rightarrow O(V)$

Each edge is explored once $\rightarrow O(E)$

Total operations $\approx V + E$

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:

```

#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 $\rightarrow V$

Each edge visited once $\rightarrow E$

SPACE COMPLEXITY:- $O(V)$

JUSTIFICATION:-

Recursion stack $\rightarrow V$

Visited array $\rightarrow V$