DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK-2

NAME : M. Charitha Varshini

ROLL.NO: CH.SC.U4CSE24128

DEPARTMENT: CSE-B

Sorting Techniques

1) Bubble Sort

```c
#include<stdio.h>
int main(){
     int n,i,j,temp;
    printf("Enter the no.of elements:");
    scanf("%d",&n);
    int a[n];
    printf("Enter the %d elements:\n",n);
    for(i=0;i<n;i++){
        scanf("%d",&a[i]);
    }

    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("Sorted elements are:\n");
    for(i=0;i<n;i++){
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```

```
amma@amma25:~$ gcc -o bubblesort bubblesort.c
amma@amma25:~$ ./bubblesort
Enter the no.of elements:5
Enter the 5 elements:
23 90 12 67 10
Sorted elements are:
10 12 23 67 90
amma@amma25:~$
```

Time Complexity:

The outer loop executes n−1 times.

For each pass, the inner loop compares adjacent elements.

Total number of comparisons is proportional to n^2.
Time Complexity = O(n^2)

Space Complexity:

Variables used:
int n, i, j, temp- 4*4=16 bytes

No additional data structures are used.

Memory usage does not depend on input size.
Space Complexity = O(1)

2) Insertion sort

```c
#include <stdio.h>
void insertionSort(int a[], int n) {
    int i, j, key;
    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;
    }
}
int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    insertionSort(a, n);
```

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

    return 0;
}
```

```
amma@amma25:~$ gcc -o insertion insertion.c
amma@amma25:~$ ./insertion
Enter number of elements: 5
Enter 5 elements:
7 10 1 6 2
Sorted array: 1 2 6 7 10
amma@amma25:~$
```

Time complexity:

Outerloop- n times

Ineerloop-n times

Time complexity= O(n^2)

Space complexity:

Int n-4 bytes

Int i- 4 bytes

Int j- 4 bytes

Int key- 4 bytes

Total bytes= 16 bytes

Space complexity=O(1)

3) Bucket sort

```c
#include <stdio.h>
void bucketSort(int a[], int n) {
    int i, j;
    int max = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    int bucket[max + 1];
    for (i = 0; i <= max; i++)
        bucket[i] = 0;
    for (i = 0; i < n; i++)
        bucket[a[i]]++;
    j = 0;
    for (i = 0; i <= max; i++)
        while (bucket[i] > 0) {
            a[j++] = i;
            bucket[i]--;
        }
}
```

```c
int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    bucketSort(a, n);

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

    return 0;
}
```

```
amma@amma25:~$ gcc -o bucketsort bucketsort.c
amma@amma25:~$ ./bucketsort
Enter number of elements: 4
Enter 4 elements:
10 1 22 2
Sorted array: 1 2 10 22
amma@amma25:~$
```

Time complexity:

Elements are distributed into buckets in linear time

Each bucket is then sorted individually

Time complexity= O(n)

Space complexity= O(n)

4) Heap sort

```c
#include <stdio.h>

void heapify(int a[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;

    if (left < n && a[left] > a[largest])
        largest = left;

    if (right < n && a[right] > a[largest])
        largest = right;

    if (largest != i) {
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;

        heapify(a, n, largest);
    }
}

void heapSort(int a[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(a, n, i);
```

```c
    for (int i = n-1; i >= 0; i--) {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;

        heapify(a, i, 0);
    }
}
int main() {
    int n, i;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

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

    return 0;
}
```

```
amma@amma25:~$ gcc -o heapsort heapsort.c
amma@amma25:~$ ./heapsort
Enter number of elements: 6
Enter 6 elements:
12 90 45 89 23 2
Sorted array: 2 12 23 45 89 90
amma@amma25:~$
```

Time complexity:

Heap – O(n)

Heapify – n times

Each heapify takes logn times

Time Complexity = O(n log n)

Space complexity:

Int n- 4 bytes

Int i- 4 bytes

Int temp- 4 bytes

Space Complexity = O(1)

5) Selection sort

```c
#include <stdio.h>
void selectionSort(int a[], int n) {
    int i, j, minIndex, temp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (a[j] < a[minIndex])
                minIndex = j;
        }
        temp = a[i];
        a[i] = a[minIndex];
        a[minIndex] = temp;
    }
}
int main() {
    int n, i;

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

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

    selectionSort(a, n);
```

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

    printf("\n");
    return 0;
}
```

```
amma@amma25:~$ gcc -o selectionsort selectionsort.c
amma@amma25:~$ ./selectionsort
Enter number of elements: 5
Enter 5 elements:
6 10 24 1 0
Sorted array: 0 1 6 10 24
amma@amma25:~$
```

Time complexity:

Outer loop – (n-1) times

Inner loop – n times

n*n

Time complexity= O(n^2)

Space complexity:

Variables used
int n, i, j, min, temp- 4*5=20 bytes

Space complexity =O(1)

Graph Traversals

6)BFS

```c
#include <stdio.h>
#define MAX 20
int main() {
    int n;
    int adj[MAX][MAX];
    int visited[MAX] = {0};
    int queue[MAX];
    int front = 0, rear = -1;
    int start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    visited[start] = 1;
    queue[++rear] = start;
    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = queue[front++];
        printf("%d ", current);
```

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

            if (adj[current][i] != 0 && !visited[i]) {
                visited[i] = 1;
                queue[++rear] = i;
            }
        }
    }

    return 0;
}
```

```
Enter number of vertices: 3
Enter adjacency matrix:
0 1 1
1 0 0
0 1 0
Enter starting vertex: 1
BFS Traversal: 1 0 2
```

Time complexity:

Each vertex is visited once

For each vertex, all n vertices are visited

Total = n*n

Time complexity: $O(n^2)$

Space complexity:

Int adj[Max][Max] – fixed size

Int visited[Max]- fixed size

Int queue[Max]- fixed size

Space complexity: O(1)

7)DFS

```c
#include <stdio.h>
#define MAX 20
int adj[MAX][MAX];
int visited[MAX] = {0};
void dfs(int vertex, int n) {
    visited[vertex] = 1;
    printf("%d ", vertex);
    for (int i = 0; i < n; i++) {
        if (adj[vertex][i] != 0 && !visited[i]) {
            dfs(i, n);
        }
    }
}
int main() {
    int n;
    int start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
```

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

    printf("DFS Traversal: ");
    dfs(start, n);
    printf("\n");
    return 0;
}
```

```
Enter number of vertices: 4
Enter adjacency matrix:
0 1 1 1
0 0 1 0
1 1 0 1
1 1 0 0
Enter starting vertex: 0
DFS Traversal: 0 1 2 3
```

Time complexity:

Each vertex is visited once

For each vertex, all n vertices are visited

Total = n*n

Time complexity: $O(n^2)$

Space complexity:

Uses visited array and recursion stack

Space complexity= O(n)