Design Analysis and Algorithm – Lab Work

Week 2

Question 1: Bubble Sort.

Code:

```c
//Bubble Sort
#include <stdio.h>
int main() {
    int a[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - i - 1; j++) {
            if(a[j] > a[j + 1]) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm> cd "Week 2"
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc bubble.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of elements: 4
Enter elements:
23
1
45
21
Sorted array:
1 21 23 45
```

Space Complexity:

The space occupied will be 416 bytes. Out of this, 400 bytes are used to store the array of size 100, 4 bytes are used for the variable n, 8 bytes are used for the loop variables i and j, and 4 bytes are used for the temporary variable temp. Since the memory used by the program remains constant and does not depend on the number of input elements, the Space Complexity is O(1).

Time Complexity:

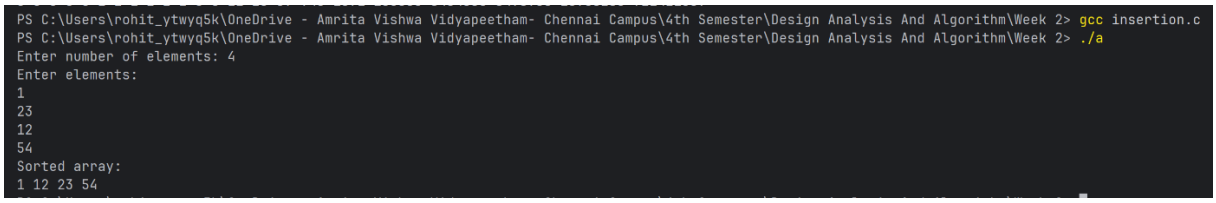
The program contains two nested loops where the outer loop runs (n−1) times and the inner loop runs (n−i−1) times. In the worst case, the total number of comparisons will be n(n−1)/2. Hence, the Time Complexity of Bubble Sort in the best case, average case, and worst case is $O(n^2)$.

Question 2: Insertion Sort.

Code:

```c
//Insertion Sort
#include <stdio.h>
int main() {
    int a[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;
        while(j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:



Space Complexity:

The space occupied will be 416 bytes. Here, 400 bytes are used for storing the array elements, and the remaining 16 bytes are used for the variables n, i, j, and key. Since only a constant amount of extra memory is used and it does not depend on the input size, the Space Complexity is O(1).

Time Complexity:

The outer loop runs (n−1) times, and the inner while loop may run up to i times in the worst case. When the array is already sorted, the inner loop executes only once for each element, resulting in linear time. Time Complexity is $O(n^2)$.

Question 3: Selection Sort.

Code:

```c
//Selection Sort
#include <stdio.h>
int main() {
    int a[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = 0; i < n - 1; i++) {
        int min = i;
        for(int j = i + 1; j < n; j++) {
            if(a[j] < a[min])
                min = j;
        }
        int temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc selection.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of elements: 4
Enter elements:
23
12
47
1
Sorted array:
1 12 23 47
```

Space Complexity:

The total space occupied will be 420 bytes. Among this, 400 bytes are used for the array of size 100, and 20 bytes are used for the variables n, i, j, min, and temp. Since the memory usage remains constant irrespective of the number of input elements, the Space Complexity is O(1).

Time Complexity:

The program consists of two nested loops which execute irrespective of the order of elements. The total number of comparisons made is $n(n-1)/2$. Hence, the Time Complexity worst case is $O(n^2)$.

Question 4:Bucket Sort.

Code:

```c
//Bucket Sort
#include <stdio.h>
#define MAX 100
int main() {
    int a[100], bucket[MAX] = {0};
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter elements (0-99):\n");
    for(int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        bucket[a[i]]++;
    }
    int index = 0;
    for(int i = 0; i < MAX; i++) {
        while(bucket[i] > 0) {
            a[index++] = i;
            bucket[i]--;
        }
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc bucket.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of elements: 4
Enter elements (0ΓÇô99):
12
1
13
3
Sorted array:
1 3 12 13
```

Space Complexity:

This method uses a two dimensional Bucket Array to sort the floating point values. Hence the Space Complexity is O(N^2).

Time Complexity:

In the worst case, all elements may fall into a single bucket and are sorted using insertion sort. Hence the Time Complexity is O(N^2).

Question 5: Min Heap Sort.

Code:

```c
//Min Heap
#include <stdio.h>
void heapify(int a[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if(left < n && a[left] < a[smallest])
        smallest = left;
    if(right < n && a[right] < a[smallest])
        smallest = right;
    if(smallest != i) {
        int temp = a[i];
        a[i] = a[smallest];
        a[smallest] = temp;
        heapify(a, n, smallest);
    }
}
int main() {
    int a[100], n;

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

    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    for(int i = n - 1; i > 0; i--) {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(a, i, 0);
    }
    printf("Sorted array:\n");
```

```
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc minheap.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of elements: 4
Enter elements:
2
3
13
4
Sorted array:
13 4 3 2
```

Space Complexity (Worst Case):

The worst case space occupied includes 400 bytes for the array and 24 bytes for the variables used in heap operations. Additional space is required for recursive calls up to the height of the heap. Hence, the worst case Space Complexity is O(log n).

Time Complexity (Worst Case):

The worst case occurs when heapify is performed for all elements and each heapify operation takes logarithmic time. Therefore, the worst case Time Complexity is O(n log n).

Question 6:Max Heap Sort.

Code:

```
//Max Heap
#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);
    }
}
int main() {
    int a[100], n;
    printf("Enter number of elements: ");
```

```
    scanf("%d", &n);
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for(int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);
    for(int i = n - 1; i > 0; i--) {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(a, i, 0);
    }
    printf("Sorted array:\n");
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc maxheap.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of elements: 4
Enter elements:
2
7
3
1
Sorted array:
1 2 3 7
```

Space Complexity (Worst Case):

The worst case space occupied consists of 400 bytes for the array and 24 bytes for variables. Additional memory is used by recursive heapify calls which depend on the height of the heap. Therefore, the worst case Space Complexity is O(log n).

Time Complexity (Worst Case):

The worst case occurs when heapify operations are required for every element. Since each heapify takes logarithmic time, the total time becomes proportional to n log n. Therefore, the worst case Time Complexity is O(n log n).

Question 7: Breadth First Search.

Code:

```c
#include <stdio.h>
int main() {
    int n, i, j, start;
    int adj[10][10], visited[10] = {0};
    int queue[10], front = 0, rear = 0;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);
    printf("BFS Traversal: ");
    queue[rear++] = start;
    visited[start] = 1;
    while(front < rear) {
        int v = queue[front++];
        printf("%d ", v);

        for(i = 0; i < n; i++) {
            if(adj[v][i] == 1 && visited[i] == 0) {
                queue[rear++] = i;
                visited[i] = 1;
            }
        }
    }
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of vertices: 5
Enter adjacency matrix:
5
0 1 1 0 0
1 0 1 1 0
1 1 0 0 1
0 1 0 0 1
0 0 1 1 0
Enter starting vertex (0 to 4): BFS Traversal: 0 2 3 1 4
```

Space Complexity:

The adjacency matrix adj[10][10] requires $O(n^2)$ space for n vertices

Time Complexity:

BFS processes each of n vertices once, scanning the full row of n elements in the adjacency matrix for each vertex, resulting in $O(n \times n) = O(n^2)$ time.

Question 8: Depth First Search.

Code:

```c
#include <stdio.h>
#include <stdio.h>
int adj[10][10], visited[10] = {0}, n;
void dfs(int v) {
    int i;
    visited[v] = 1;
    printf("%d ", v);

    for(i = 0; i < n; i++) {
        if(adj[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}
int main() {
    int i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("Enter starting vertex (0 to %d): ", n - 1);
    scanf("%d", &start);
    printf("DFS Traversal: ");
    dfs(start);
    return 0;
}
```

Output:

```
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> gcc dfs.c
PS C:\Users\rohit_ytwyq5k\OneDrive - Amrita Vishwa Vidyapeetham- Chennai Campus\4th Semester\Design Analysis And Algorithm\Week 2> ./a
Enter number of vertices: 5
Enter adjacency matrix:
0 1 1 0 0
1 0 1 1 0
1 1 0 0 1
0 1 0 0 1
0 0 1 1 0
Enter starting vertex (0 to 4): 0
DFS Traversal: 0 1 2 4 3
```

Space Complexity :

The worst case space occupied includes the adjacency matrix adj[10][10], which requires 10 × 10 × 4 = 400 bytes, and the visited array visited[10], which requires 10 × 4 = 40 bytes. In addition to this, space is required for the recursive call stack during DFS traversal. In the worst case, the recursion depth can go up to n vertices, requiring 4n bytes. Therefore, the worst case Space Complexity is $O(n^2)$ due to the adjacency matrix representation.

Time Complexity :

The worst case occurs when all vertices are visited during the DFS traversal. For each vertex, the algorithm scans all n vertices in the adjacency matrix to check for edges. Hence, the total number of operations becomes proportional to n × n. Therefore, the worst case Time Complexity is $O(n^2)$.