Foundation Session - 1

Agenda

1. Asymptotic Analysis
2. Rules to follow
3. Big-O
4. Big-Theta
5. Big-Omega
6. Best-Case v/s Average-Case v/s Worst-Case Complexity
7. Auxiliary Space v/s Space Complexity
8. String Sorting Analysis
9. Recursive Runtimes

2 rules:

1. Drop-off the constants
2. Drop-off the irrelevant terms

```
For (i =0; i < n; i++) {
        For (int j = 0; j < m; j++) {
                Cout << i + j << endl;
        }
}
```

TC => O(n * m)
Aux Space => O(1)
SC => O(1)

```
For (i =0; i < n; i++) {
        Cout << i << endl;
}
For (int j = 0; j < m; j++) {
        Cout << j << endl;
}
```

TC => O(n + m)
Aux Space => O(1)
SC => O(1)

```
arr => matrix of strings
For (int i =0; i < n; i++) {
        For (int j = 0; j < m; j++) {
                Cout << arr[i][j] << endl;
        }
}
```

TC => O(n * m * max_len_of_any_string_in_matrix)
Aux Space => O(1)
SC => O(n * m * max_len_of_any_string_in_matrix)

Printing an Integer => O(1)
32-bit integer => [0, 0, 1, 1 …..]

Printing a String => O(length of String)

Question:
You have an array of Strings.
Array size = n
Length of all the strings = m
TODO: Sort each of the strings in the array individually and then sort the complete array.

Input = [“acb”, “dfg”, “bca”, “ddb”]

Step-1: [“abc”, “dfg”, “abc”, “bdd”]
Step-2: [“abc”, “abc”, “bdd”, “dfg”]

Prerequisites:

- Sorting an array of integer => O(n*log(n))
- Comparing two strings => O(m)

TC: O(n * m * log(m) + m * n * log(n)) = O(nm * (log(n) + log(m)))

Step-1:

Sorting a single string -> O(m * log(m))
Sorting n number of strings -> O(n * m * log(m))

Step-2:

O(m * n * log(n))

<u>Recursive Analysis:</u>

```
Int fib(int n) {
    If (n <= 1) return n;
    Return fib(n-1) + fib(n-2);
}
```

TC: O(2^n)
AS: O(n)

```
Int fun(int n) {
        If (n <= 1) return n;
        For (int i = 0; i < n; i++) { … do something O(1) …}
        Return fun(n-1) + fun(n-2);
}
```

TC: O(n * 2^n)
AS: O(n)

```
Int fun(int n, arr[n]) {
        If (n <= 1) return n;
        … do something with the array …
        For (int i = 0; i < n; i++) { … do something O(1) …}
        Return fun(n-1, arr) + fun(n-2, arr);
}
```

Time:
If array is passed by value -> O(n * 2^n)
If array is passed by ref -> O(n * 2^n)

Aux Space:
If array is passed by value -> O(n^2)
If array is passed by ref -> O(n)