Data Structures and Algorithms:

Data Structures-Structures used to store the data like array,linked list,trees,graphs,queues,stacks,hashmaps etc.

Algorithms-processes/operations performed on the data stored in data structures like searching,sorting,finding the shortest path from source to destination.

Importance of DSA: To crack technical Interviews(give examples where you were asked or you asked DSA in Interviews or mocks etc.)

Uses of DSA:
Tries-search engines
(google search)

Graphs-Google maps
Linked List -Music player
Trees-files and folders

Time Complexity:No. of Iterations or how many elements are touched or disturbed and how many times.

Time Complexity!=Time taken to excecute the code as time taken is machine dependent. example-Old windows and new Macbook might excecute the same code in different times.

Different Time Complexities:

Constant TC:fixed no. of iterations or no iteration
Example if-else code snippet
Print statement
Etc.

Linear TC:
Single loop

```
let n = 10;  // You can set n to any positive integer

for (let i = 1; i <= n; i++) {
    console.log(i);
}
```

For i=1,1 time till now it has run
i=2,till now 2 times it has run
.
.
.
i=n,n times till now it has run

TC:O(n) where is n is the input given and total n iterations happened means n elements are disturbed one time each.
or,

multiple loops independent of each other

```
let n = 10;  // Set n to any positive integer
let m = 5;   // Set m to any positive integer

// Loop from 1 to n
console.log('Loop from 1 to n:');
for (let i = 1; i <= n; i++) {
    console.log(i);
}
```

For i=1,1 time till now it has run
i=2,till now 2 times it has run
.
.
.
i=n,n times till now it has run

```
// Loop from 1 to m
console.log('Loop from 1 to m:');
for (let j = 1; j <= m; j++) {
    console.log(j);
}
```

For i=1,1 time till now it has run
i=2,till now 2 times it has run
.
.
.
i=m,m times till now it has run

TC:O(n+m) where in n,m is the input given and total n iterations happened means n,m elements are disturbed one time each.

Squared TC:

```
let n = 5;  // You can set n to any positive integer
let m = 3;  // You can set m to any positive integer

for (let i = 1; i <= n; i++) {
```

```
    for (let j = 1; j <= m; j++) {
        console.log(`i: ${i}, j: ${j}`);
    }
}
```

TC:O(n*m)
n elements are iterated or disturbed m times each.

For i=1,m iterations are done
i=2,m iterations are done
.
.
i=n,m iterations are done

Logarithmic TC:

```
let n = 64;  // You can set n to any positive integer

while (n > 1) {
    n = Math.floor(n / 2);  // Divide n by 2 and round down to the nearest integer
}
```

N=1,0 iterations
N=2,1 iteration(2/2=1 then no further iteration)
N=3,1 iteration(3/2=1 then no further iteration)
N=4(2 iterations(4/2,2/2, then no further iteration)
.
.
.
N=8(3 iterations,8/2,4/2,2/2, then no further iteration)

Each time we are reducing i to its half that is n/2->n/4->n/8 —-------n/n until it becomes 1.
So the time is O(log base2 of n) as in log base2 n steps we can reduce n to 1.

Underroot Time Complexity:

```
let n = 100;  // You can set n to any positive integer

for (let i = 1; i * i <= n; i++) {
    console.log(i);
}
```

TC:O(root n)

Rules to write time complexity:
1.write the overall complexity of code
2.Erase all lower complexities leaving the higher TC only
3.Ignore constants

Why Erase all lower complexities leaving the higher TC only?
Example: N^2+10N

Input size	Iterations	% of lower complexity in total overall complexity
N=10	10^2+10(10) 100+100	100/200*100=50%
N=100	100^2+10(100) 10000+1000	1000/11000*100=9% something
N=1000	1000^2+10(1000) 1000000+10000	10000/1001000*100=0.something

Each time the input increases the affect of lower TC's is decreasing

Why Ignore constants?

2n and 3n is both linear no matter n is getting multiplied by 2 or 3.
Also,

```
for (let i = 1; i <= n; i++) {
   console.log(i);
}
```

In this for every i,three operations are fixed i.e. checking the condition,excecuting statements inside curly brackets and incementing i.

So n*3 is the complexity if n changes from 10 to 50 stills these 3 operations are fixed.
So,constant never changes with increase in input.

Space Complexity:
Given Space:Given to you in problems itself.
Auxiliary Space:That you use on your own to solve a problem.
Examples etc.