Agenda for today's live session

Analysis

Algorithms : It is a combination of sequence of finite steps to solve a particular problem.

for example : Multiplication of two numbers

```
Mul() {
        1. Take two numbers (a,b)
        2. Multiply two numbers a and b and store the value of result in c
        3. return c
}
```

Properties of Algorithms :

- **It should terminate after finite time.**
- **It should produce atleast one output.**
- **It is independent of any sort of programming language.**
- **It should be unambiguous (Deterministic)**

Deterministic - For the same input same output will come always.

Not Deterministic - For the same input different output will come. And this is not at all preferrable whenever we write any sort of algorithms.

Question : Based on the above mentioned properties, can you determine whether the below program is an algorithm or not ?

```
while(true){
System.out.println("Hello World");
}
```

Hello World

Hello World

Hello World ....infinite times......not a algorithm

Steps Required to construct an algorithm :

- **Problem Definition -> what is the problem they are asking**
- **Design algorithm -> Out of existing algorithms which algorithm is more suitable to this problem statement.**

Existing algorithms - Divide and Conquer, Greedy Technique, Dynamic Programming, Backtracking and so on.

- **Draw flow chart**
- **Testing -> For every input correct output is coming or not**
- **Implementation -> Coding Part**
- **Analysis**

2 1 6 3 8 => 1 2 3 6 8

Note : Design algorithm and analysis are the two major steps.

Analysis : If any problem contains more than one solution, then best one will be decided by the analysis based on mainly two factors :

1. Time Complexity - CPU time

2. Space Complexity - Main Memory Space

Note : Time Complexity is more powerful than Space Complexity because processor cost is more costly.

Time Complexity : T(P) = C(P) + R(P)

C(P) -> Compile-time is the time at which the source code is converted into an executable code.

R(P) -> Run time is the time at which the executable code is started running.

Types of analysis :

1. Apostiary Analysis (Relative Analysis) :

- Dependent on language of compiler and the type of hardware
- Exact answer
- Different answer
- Program run fast because of the type of hardware used

2. Apriori Analysis (Absolute Analysis) :

- Independent on language of compiler and the type of hardware
- Approximate answers
- same answer
- Program run fast because of nice logic

Apriori Analysis : It is a determination of order of magnitude of a statement.

O(magnitude)

Examples for better understanding of the concepts :

Problem 1 :

```
main() {
x = y + z;                O(1) - constant time - best case time complexity
}
```

Problem 2 :

```
main() {
x = y + z;                O(1) - constant time
for(i = 1; i <= n ; i++){
x = y + z;                 O(n) time
}
}
```

Overall time complexity = O(1) + O(n) = O(n)

O(n+1) = O(n)

n = 100000000000 + 1 = 100000000000

n = 5 = 5 times

n = 10 = 10 times

n = 1000 = 1000 times and so on

i = 1 ; 1 <= 5 = true

x = y + z   1st time

i++ = i + 1 = 1 + 1 = 2 ; 2 <= 5 = true

x = y + z   2nd time

i++ = i + 1 = 2 + 1 = 3; 3 <= 5 = true

x = y + z   3rd time

i++ = i + 1 = 3 + 1 = 4; 4 <= 5 = true

x = y + z   4th time

i++ = i + 1 = 4 + 1 = 5; 5 <= 5 = true

x = y + z   5th time

i++ = i + 1 = 5 + 1 = 6; 6 <= 5 = false

no statement is executed and for loop is terminated here

Problem 3 :

```
main(){
x = y + z;            O(1) - constant time
for(i = 1; i <= n ; i++){
x = y + z;            O(n) - time
}
for(i = 1; i <= n ; i++){
        for(j = 1; j <= n; j++){
        x = y + z;            O(n^2) time
        }
}
}
```

So, overall time complexity of the above code : 1 + n + n^2 = O(n^2) time complexity.

Execution of code :

```
for(i = 1; i <= n ; i++){
        for(j = 1; j <= n; j++){
        x = y + z;                    O(n^2) time
        }
}
```

n = 3   =>   3*3 = 9 times (x = y+z)

n = 10 =>   10*10 = 100times(x = y+z)

n = 100000 => 100000*100000 (x = y + z)

n^2

i = 1; 1 <= 3 = true

j = 1; 1 <= 3 = true

```
        x = y + z       1st time
```

j++ = j + 1 = 1 + 1 = 2 <= 3 = true

```
        x = y + z       2nd time
```

j++ = j + 1 = 2 + 1 = 3 <= 3 = true

```
        x = y + z       3rd time
```

----

j++ = j + 1 = 3 + 1 = 4 <= 3 = false

```
        // no statement is executed now
```

i = 2 ; 2 <= 3 = true

j = 1; 1 <= 3 = true

```
        x = y + z       4th time
```

j++ = j + 1 = 1 + 1 = 2 <= 3 = true

```
        x = y + z       5th time
```

j++ = j + 1 = 2 + 1 = 3 <= 3 = true

```
        x = y + z       6th time
```

----

j++ = j + 1 = 3 + 1 = 4 <= 3 = false

    // no statement is executed now

i = 3; 3 <= 3 = true

j = 1; 1 <= 3 = true

    x = y + z   7th time

j++ = j + 1 = 1 + 1 = 2 <= 3 = true

    x = y + z   8th time

j++ = j + 1 = 2 + 1 = 3 <= 3 = true

    x = y + z   9th time

----

j++ = j + 1 = 3 + 1 = 4 <= 3 = false

    // no statement is executed now

i++ = i + 1 = 3 + 1 = 4 ; 4 <=3 = false

//no statement is executed now

Problem 4 :

```
main(){
i = n;
while(i > 1){
i = i - 1;                    O(n-1) = O(n)
}
}
```

n = 10000000000000000 - 1 is it effect?

n = 5 (Assume)	4 times	(n-1) times
n = 10	9 times	
n = 100	99 times	

i = 5

5 > 1 = true

i = i - 1 = 5 - 1 = 4 1st time

----

i = 4

4 > 1 = true

i = i - 1 = 4 - 1 = 3 2nd time

----

i = 3

3 > 1 = true

i = i -1 = 3 - 1 = 2 3rd time

----

i = 2

2 > 1 = true

i = i - 1 = 2 - 1 = 1 4th time

----

i = 1

1 > 1 = false no execution inside while loop

//now it will not enter into the while loop statements

Problem 5 :

```
main(){
i = n;
while(i >= 1){
i = i - 2;        // statement
}
}
```

n = 10 assumption	5 times
n = 100	50 times
n = 1000	500 times
n	n/2 times

Overall time complexity of above program : O(n/2) = O(n)

i = 10

10 >= 1 true

i = 10 - 2 = 8 1st time

8 >= 1 true

i = 8-2 = 6 2nd time

6 >= 1 true

i = 6-2 = 4 3rd time

4 >= 1 true

i = 4-2 = 2 4th time

2 >= 1   true

i = 2-2 = 0   5th time

0 >= 1   false

// no statement will be executed inside the while loop

Problem 6 :

```
main(){
i = n;
while(i >= 1){
i = i - 30;                    // i = i - 35
i = i - 5;
}
}
```

n   n/35 times   O(n/35) = O(n)

Problem 7 :

```
main() {
i = 1;
        while(i < n){
                i = 2 * i;      // statement
        }
}
```

n = 64 assumption   6 time

n = 32   5 time

n log_2 n

Overall time complexity of above program : O(log_2 n)

i = 1 given

1 < 64 true

i = 2 * i = 2 * 1 = 2 1st time

2 < 64 true

i = 2 * i = 2 * 2 = 4 2nd time

4 < 64 true

i = 2 * i = 2 * 4 = 8 3rd time

8 < 64 true

i = 2 * i = 2 * 8 = 16 4th time

16 < 64 true

i = 2 * i = 2 * 16 = 32 5th time

32 < 64 true

i = 2 * i = 2 * 32 = 64 6th time

64 < 64 false

// no statement will be executed after this

Conclusion :

- Time complexity is loop only
- Not only loop but larger loop

  n + n^2 + n^3 = O(n^3)
- And if in a program there is no loop at all - O(1) - best case scenario