Time Complexity

Analyzing Simple Algorithms:

Algorithm analysis is a fundamental skill in computer science. Algorithm analysis refers to the process of evaluating the performance of an algorithm in terms of time and space complexity. Algorithm analysis is essential for understanding the efficiency and scalability of algorithms.

Loop Analysis for time complexity

Iteration count:

Iteration is a single pass through the body of a loop. Whenever the loop's code executes once, it counts as one iteration. Iterations are the actions performed by the loop until a condition is met.

Counting operation :

The time complexity of a loop is typically measured by the number of operations it performs in an algorithm.

Loop Analysis

1. Single loops :

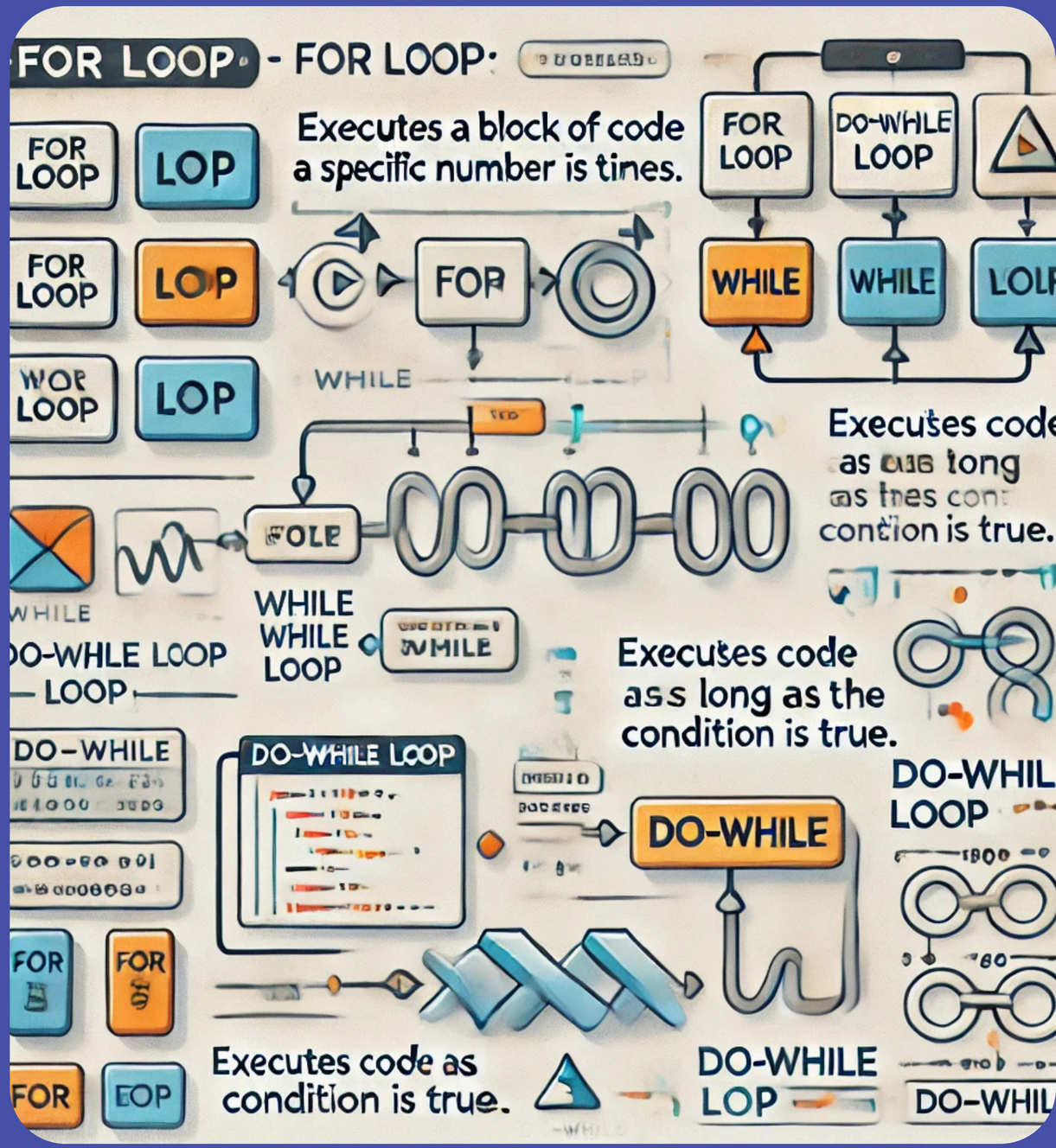
A single loop refers to a loop that executes a block of code repeatedly for a certain number of times or until a condition is met, and it only uses one loop structure.

Types of single loop:

- For loop
- do while loop
- While loop

Time Complextiy of singleloop

A single loop that iterates n times has a time complexity of O(n).



2. Nested loops :

A nested loop is a loop inside another loop. the inner loop is executed completely every time the outer loop runs once.

Nested loops time complexity:

Nested loops where both outer and inner loop iterate n times have a time complexity of O(n^2).

Code of nested loops:

```
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 10; i++) { // outer
loop
        for (int j = 1; j <= 10; j++) { // inner
loop
            cout << i * j ;
        }
        cout << endl;
    }
    return 0;
}
```

Recursion :

Recursion is a programming concept where a function calls itself directly or indirectly into its function to solve a problem.

Features of recursion :

Base Case::

Every recursive function must have a base case to stop the recursion. It is the condition under which the recursion stops.

Recursive Case :

Recursive case is a case where a function calls itself inside a function .

Example of recursive case:

```
int GCD(int A, int B) {
 if (A % B == 0)
  return B;
 return GCD(B, A % B);
}
```

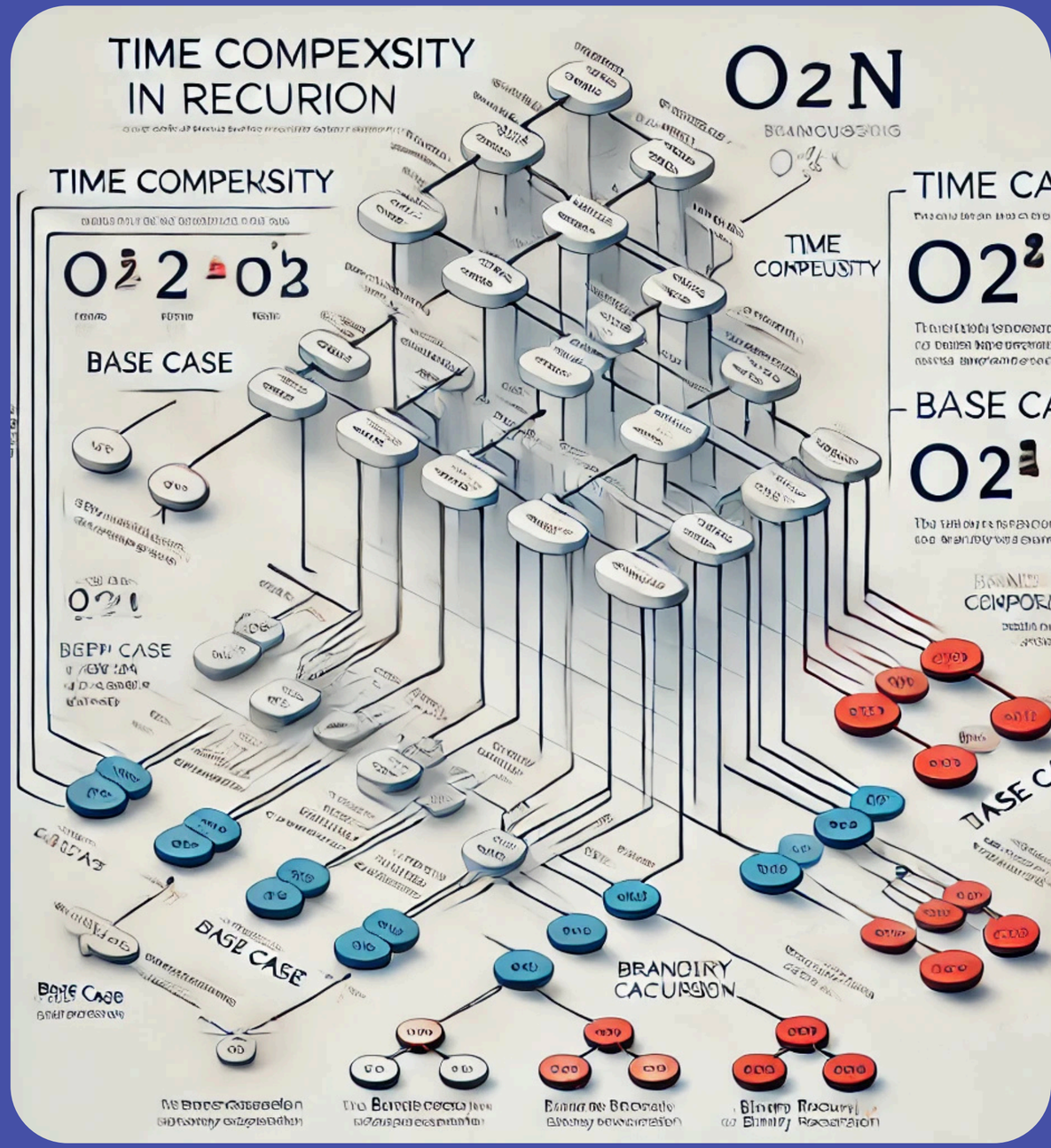
Further examples of recursion and their time complexity

Code of factorial :

```
long long factorial(int n)
 {
   if (n <= 1) return 1;  // base case
   return n * factorial(n - 1);  // recursive call
}
```

Time complexity of factorial

At **each recursive call, the function performs one multiplication and makes one recursive call.**
Since there are n calls, the time complexity is: O(n) .



Code of factorial :

```
long long factorial(int n)
 { if (n <= 1) return 1; // base case
return n * factorial(n - 1); // recursive call
 }
```

Time complexity of factorial

Recursive fibonacci makes two recursive calls for each call, resulting in a binary recursion tree. The total number of calls grows exponentially, leading to a time complexity of O(2^n).