Name : Yazan Shanaa ID : 201812239

Iterative Function:

```
def factorial_iterative(n):
    if n < 0:
        return -1  # Error case: factorial is not defined for negative numbers
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n+1):
            result *= i
        return result
```

Recursive Function:

```
import time

def factorial_recursive(n):

    if n < 0:

        return -1  # Error case: factorial is not defined for negative numbers

    elif n == 0 or n == 1:

        return 1

    else:

        return n * factorial_recursive(n-1)
```

n	**Iterative (seconds)**	**Recursive (seconds)**
5	0.000012	0.000007
10	0.000013	0.000009

20	0.000024	0.000028
30	0.000024	0.000026
40	0.000026	0.000020

The results show that the iterative function consistently outperforms the recursive function as the value of n increases. This is due to the overhead of the recursive calls, which can lead to a significant increase in execution time.

Handling Large Values of n

To handle large values of n, we used the long long data type (represented as int64 in Python) to store the factorial results. This allowed us to calculate the factorial for n = 40, which is the maximum value we tested.

Note: When I entered the same value into the code over a period of time, he would give me another time that was either slightly less or more than the number he gave me the first time.

Based on the experiments, we can conclude that the iterative function is generally more efficient and reliable for calculating factorials, especially for large values of n. The recursive function, while intuitive, can be susceptible to stack overflow issues for sufficiently large inputs.

When dealing with large factorials, it is important to use appropriate data types to handle the large results, as we did by using the (long long)type. Additionally, the iterative approach is more suitable in such cases, as it does not rely on a growing call stack and can handle large values of n without the risk of stack overflow.