Zig-Zag Recursion Quick Revision Notes (Hinglish)

Logic of the Given Code:

- Function `zig(int n)` ek "zig-zag" pattern print karta hai recursion use karke.
- **Base Condition**:
  - Agar `n == 0`, toh recursion stop ho jata hai aur function return ho jata hai.
- **Recursive Calls**:
  - Pehle `cout << n;` use karke `n` print hota hai (pre-recursive print).
  - Phir `zig(n-1);` call hota hai (first recursive call).
  - Jab recursion return hota hai, tab dubara `cout << n;` print hota hai (mid-print).
  - Phir dusri baar `zig(n-1);` call hota hai (second recursive call).
  - Finally, function return hone se pehle `cout << n;` firse print hota hai (post-recursive print).
- Iss wajah se ek symmetrical "zig-zag" pattern banta hai.

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

// Zig-zag pattern print karne ke liye recursive function
int zig(int n){
    if(n == 0) return 0; // Base case: Jab n 0 ho, recursion stop kar do

    cout << n;  // Pehla print (pre-recursive print)
    zig(n-1);   // First recursive call
    cout << n;  // Mid-recursive print
    zig(n-1);   // Second recursive call
    cout << n;  // Post-recursive print
}

int main(){
    zig(4); // Function call for n = 4
}
```

Dry Run of the Code (For zig(2))

Function Call	Output
zig(2)	2 (pre)
zig(1)	1 (pre)

zig(0) | - (base case)
        | 1 (mid)
zig(0) | - (base case)
        | 1 (post)
        | 2 (mid)
zig(1) | 1 (pre)
zig(0) | - (base case)
        | 1 (mid)
zig(0) | - (base case)
        | 1 (post)
        | 2 (post)

Final Output for zig(2):

2 1 1 1 2 1 1 1 2

Time Complexity:

- Har function call do aur recursive calls karta hai.
- Is wajah se time complexity **O(2^n)** hoti hai.

Key Takeaways:

- Yeh function ek symmetric "zig-zag" pattern print karta hai.
- Yeh **pre, mid, aur post recursion** pattern follow karta hai.
- Recursion calls exponential growth karti hain, jo ise bade values ke liye slow bana deti hain.