Zig-Zag Recursion Quick Revision Notes

Logic of the Given Code:

- The function `zig(int n)` prints numbers in a "zig-zag" pattern using recursion.
- **Base Condition**:
  - If `n == 0`, return `0` to stop further recursion.
- **Recursive Calls**:
  - First, print `n` before calling `zig(n-1)` (pre-recursive call).
  - After returning from the first recursive call, print `n` again (mid-print).
  - Then call `zig(n-1)` again (post-recursive call).
  - Finally, print `n` once more after the second recursive call.
- This results in a symmetrical "zig-zag" pattern of numbers.

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

// Function to print zig-zag pattern using recursion
int zig(int n){
    if(n == 0) return 0; // Base case: Stop recursion when n reaches 0

    cout << n;  // 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); // Calling function 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:

- Each function call makes two more recursive calls.
- This results in **O(2^n)** time complexity.

Key Takeaways:

- This function prints numbers in a symmetrical "zig-zag" order.
- It follows a **pre, mid, and post recursion** pattern.
- The number of recursive calls grows exponentially, making it inefficient for large values.