Fibonacci Recursion Quick Revision Notes

Logic of the Given Code:

- The function `fibo(int x)` is designed to compute the Fibonacci number at position `x` using recursion with memoization.
- **Base Condition**:
  - If `x == 0`, return `0` (First Fibonacci number).
  - If `x == 1`, return `1` (Second Fibonacci number).
- Memoization is used to optimize repeated calculations by storing computed results in the array `arr`.
- If the Fibonacci value for `x` is already stored in `arr`, it is directly returned to avoid redundant computations.
- Otherwise, it recursively calls `fibo(x-1) + fibo(x-2)` and stores the result in `arr[x]`.

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

long long arr[100]; // Array for memoization

// Function to compute Fibonacci numbers using recursion and memoization
long long fibo(int x){
    if(x == 0) return 0; // Base case: First Fibonacci number
    else if (x == 1) return 1; // Base case: Second Fibonacci number
    else if(arr[x] != 0) return arr[x]; // Return stored value if already computed

    return arr[x] = fibo(x-1) + fibo(x-2); // Store computed value in arr[x]
}

int main(){
    int p;
    p = fibo(55); // Compute the 55th Fibonacci number
    cout << p; // Output the result
}
```

Dry Run of the Code (For fibo(5))

Function Call	x	Return Value

fibo(5) | 5 | fibo(4) + fibo(3)
fibo(4) | 4 | fibo(3) + fibo(2)
fibo(3) | 3 | fibo(2) + fibo(1)
fibo(2) | 2 | fibo(1) + fibo(0)
fibo(1) | 1 | 1 (Base Case)
fibo(0) | 0 | 0 (Base Case)

Final Computation:

fibo(2) = 1 + 0 = 1

fibo(3) = 1 + 1 = 2

fibo(4) = 2 + 1 = 3

fibo(5) = 3 + 2 = 5

Output for fibo(5): 5

For fibo(55), the computed value is very large but optimized using memoization.

Time Complexity:

- Without memoization, recursion has an exponential time complexity: **O(2^n)**.
- With memoization (storing computed values), the time complexity reduces to **O(n)**.

Key Takeaways:

- Recursion simplifies the Fibonacci series but is inefficient without memoization.
- Memoization optimizes recursion by reducing redundant calculations.
- The approach effectively computes Fibonacci numbers in **O(n)** time.