Staircase Problem Recursion Quick Revision Notes

Logic of the Given Code:

- The function `stair(int x)` calculates the number of ways to reach the `x`th step using recursion.
- **Base Condition**:
  - If `x == 1`, return `1` (Only 1 way to reach step 1: a single step).
  - If `x == 0`, return `1` (1 way to stay at ground level: doing nothing).
- **Recursive Case**:
  - To reach step `x`, we can either come from step `x-1` (taking 1 step) or from step `x-2` (taking 2 steps).
  - Thus, `stair(x) = stair(x-1) + stair(x-2)`, which follows the Fibonacci sequence.

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

// Function to calculate the number of ways to reach the x-th step
int stair(int x){
    if(x == 1) return 1; // Base case: Only 1 way to reach step 1
    else if(x == 0) return 1; // Base case: Only 1 way to stay at step 0
    else return stair(x-1) + stair(x-2); // Recursive case
}

int main(){
    int p;
    p = stair(10); // Compute the number of ways to reach step 10
    cout << p; // Output the result
}
```

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

Function Call	x	Return Value
stair(5)	5	stair(4) + stair(3)
stair(4)	4	stair(3) + stair(2)
stair(3)	3	stair(2) + stair(1)
stair(2)	2	stair(1) + stair(0)
stair(1)	1	1 (Base Case)
stair(0)	0	1 (Base Case)

Final Computation:

stair(2) = 1 + 1 = 2

stair(3) = 2 + 1 = 3

stair(4) = 3 + 2 = 5

stair(5) = 5 + 3 = 8

Output for stair(5): 8

For stair(10), the computed value is much larger but follows the Fibonacci pattern.

Time Complexity:

- The given implementation has **O(2^n)** time complexity due to repeated calculations.
- Using **memoization (Dynamic Programming)**, this can be optimized to **O(n)**.

Key Takeaways:

- The problem follows the Fibonacci sequence logic.
- The given recursive solution is inefficient for large values due to redundant calculations.
- Optimized versions use **Dynamic Programming** or **Memoization** to reduce time complexity.