Given an input number of sections and each section has 2 plots on either sides of the road. Find all possible ways to construct buildings in the plots such that there is a space between any 2 buildings.

Example:

```
N = 1
Output = 4
Place a building on one side.
Place a building on other side
Do not place any building.
Place a building on both sides.

N = 3
Output = 25
3 sections, which means possible ways for one side are
BSS, BSB, SSS, SBS, SSB where B represents a building
and S represents an empty space
Total possible ways are 25, because a way to place on
one side can correspond to any of 5 ways on other side.

N = 4
Output = 64
```

We strongly recommend to minimize your browser and try this yourself first

We can simplify the problem to first calculate for one side only. If we know the result for one side, we can always do square of the result and get result for two sides.

A new building can be placed on a section if section just before it has space. A space can be placed anywhere (it doesn't matter whether the previous section has a building or not).

```
Let countB(i) be count of possible ways with i sections
              ending with a building.
    countS(i) be count of possible ways with i sections
              ending with a space.

// A space can be added after a building or after a space.
countS(N) = countB(N-1) + countS(N-1)

// A building can only be added after a space.
countB[N] = countS(N-1)

// Result for one side is sum of the above two counts.
result1(N) = countS(N) + countB(N)

// Result for two sides is square of result1(N)
result2(N) = result1(N) * result1(N)
```

Below is C++ implementation of above idea.

```
// C++ program to count all possible way to construct buildings
#include<iostream>
using namespace std;
```

```
// Returns count of possible ways for N sections
int countWays(int N)
{
    // Base case
    if (N == 1)
        return 4;  // 2 for one side and 4 for two sides

    // countB is count of ways with a building at the end
    // countS is count of ways with a space at the end
    // prev_countB and prev_countS are previous values of
    // countB and countS respectively.

    // Initialize countB and countS for one side
    int countB=1, countS=1, prev_countB, prev_countS;

    // Use the above recursive formula for calculating
    // countB and countS using previous values
    for (int i=2; i<=N; i++)
    {
        prev_countB = countB;
        prev_countS = countS;

        countS = prev_countB + prev_countS;
        countB = prev_countS;
    }

    // Result for one side is sum of ways ending with building
    // and ending with space
    int result = countS + countB;

    // Result for 2 sides is square of result for one side
    return (result*result);
}

// Driver program
int main()
{
    int N = 3;
    cout << "Count of ways for " << N
         << " sections is " << countWays(N);
    return 0;
}
```

Output:

```
25
```

Time complexity: O(N)

Auxiliary Space: O(1)

Algorithmic Paradigm: Dynamic Programming

Optimized Solution:
Note that the above solution can be further optimized. If we take closer look at the results, for different values, we can notice that the results for two sides are squares of Fibonacci Numbers.

N = 1, result = 4 [result for one side = 2]
N = 2, result = 9 [result for one side = 3]
N = 3, result = 25 [result for one side = 5]
N = 4, result = 64 [result for one side = 8]
N = 5, result = 169 [result for one side = 13]
.....
.....

In general, we can say

```
result(N) = fib(N+2)^2

fib(N) is a function that returns N'th
       Fibonacci Number.
```

Therefore, we can use O(LogN) implementation of Fibonacci Numbers to find number of ways in O(logN) time.