Text, letter

Description automatically generated

* At the base case: 

Right side: 

Left side: 

Right side equals to left side => The base case is correct.

* Induction steps

Assume that  is correct. We need to prove that the next step

 is also correct

Right side: 

Left side:  (definition of Fibonacci sequence)

Right side equals to left side => Induction steps is correct.

Therefore, this statement is correct by induction proof

Text

Description automatically generated

Matrix binary exponentiation is a faster method that can be used to find the nth element of a series defined by a recurrence relation.

This method works by dividing number of matrices into 2 equal sizes until the base case where where are only two single matrices that can be multiplied together. After that, the results are combined together (because multiplication is commutative) to achieve the final exponentiation.

*Pseudocode Algorithm* (where Strassen denotes matrix multiplication by the Strassen’s algorithm)

---------------------------------------------------------

# Only for 2x2 matrix

Strassen(mat1, mat2)

[a11, a12, a21, a22] = mat1.elements

[b11, b12, b21, b22] = mat1.elements

*P*1 = (*a*11 + *a*22)(*b*11 + *b*22)  
 *P*2 = (*a*21 + *a*22)*b*11  
 *P*3 = *a*11(*b*12 *- b*22)

*P*4 = *a*22(*b*21 *- b*11)   
 *P*5 = (*a*11 + *a*12)*b*22   
 *P*6 = (*a*21 *- a*11)(*b*11 + *b*12)   
 *P*7 = (*a*12 *- a*22)(*b*21 + *b*22)

*c*11 = *P*1 + *P*4 *- P*5 + *P*7  
 *c*12 = *P*3 + *P*5  
 *c*21 = *P*2 + *P*4  
 *c*22 = *P*1 *- P*2 + *P*3 + *P*6

matResult.elements = [c11, c12, c21, c22]

return matResult

BinExponentiation(matrix, n)

if n == 1

return matrix

else

recursedMatrix = BinExponentiation(matrix, floor(n/2))

if n mod 2 == 0

return Strassen(recursedMatrix, recursedMatrix)

else

return Strassen(Strassen(recursedMatrix \* recursedMatrix), matrix)

fibonacciSequence(n)



Return BinExponentiation(M, n)[1][0]

# F(n) is located on either exponentialMatrix[0][1] or [1][0] in this identity 



First, we can consider the initial algorithm using matrix exponentiation:



To find , we multiply the matrix by itself (n – 1) times. I will not count addition arithmetic because in the CPU, multiplication takes much more clock cycles to compute compared to addition arithmetic. Each matrix multiplication if using the Naïve approach will take 8 multiplications. In total, the running time complexity is 

The improved strategy using the binary exponentiation algorithm with the Strassen’s algorithm will have two cases: either n is even or n is odd. Each matrix multiplcation will take 7 multiplications. When n is even, the two halves are multiplied in  time. When n is odd, the two halves are multiplied, plus the multiplication with the remaining matrix, taking time

When n is even: 

When n is odd: 

So on average for both cases: 

According to Master Theorem: 

Therefore, the time complexity of new algorithm is, which runs in logarithmic time. It is faster than the Naïve method that runs in linear time.

Text, letter

Description automatically generated

We can derive the recursive formula from the direct formula as follows









Therefore, the recursive formula is 

The Recursive algorithm for binomial coefficient  
----------------------------------------  
binomialCoefficient(n, k)  
 // Base case  
 if (k > n)  
 return 0;  
 if (k == 0 || k == n)  
 return 1;

// Recursion  
 return binomialCoefficient(n - 1, k - 1) + binomialCoefficient(n - 1, k);

Text, letter

Description automatically generated

Recurrence relation verified by the time complexity of this algorithm is:

, because addition arithmetic takes a constant time

Text

Description automatically generated

We can try expanding the recurrence relation to find the pattern here:

At n – 2 level



At n – 3 level



The sum of the coefficients for all of is 2 to the power of the current level it is at. The Pascal Triangle below illustrate the sums as the power of 2

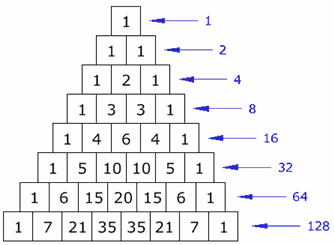


Image source: <https://socratic.org/questions/how-do-i-use-pascal-s-triangle-to-expand-x-2-5>

Therefore, the approximation of the recursive algorithm is  according to the hint => The time complexity is . For the 3 first levels n = 1, 2 and 3, we have . However, starting from 4,  starts to grow asymptotically faster than . The running time of thus is strictly faster than  for sufficiently large n value.

Therefore,  (proven)