1. **Fibonacci sequence.** The Fibonacci sequence is defined as

$$\begin{cases} F(1) = 1 \\ F(2) = 1 \\ F(n) = F(n-2) + F(n-1), \text{ when } n > 2 \end{cases}$$

(a) (1p.) Prove by induction that for all $n \geq 2$

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Hint: the zeroth power of a matrix is the identity matrix.

- At the base case: $n = 2$

Right side: $\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} F(2) \\ F(1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Left side: $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{2-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{0} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

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

- Induction steps

Assume that $\begin{bmatrix} F(k) \\ F(k-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, k \geq 2$ is correct. We need to prove that the next step

$\begin{bmatrix} F(k+1) \\ F(k) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is also correct

Right side: $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \overset{def}{=} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(k) \\ F(k-1) \end{bmatrix} = \begin{bmatrix} F(k)+F(k-1) \\ F(k) \end{bmatrix}$

Left side: $\begin{bmatrix} F(k+1) \\ F(k) \end{bmatrix} = \begin{bmatrix} F(k-1)+F(k) \\ F(k) \end{bmatrix}$ (definition of Fibonacci sequence)

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

Therefore, this statement is correct by induction proof

(b) (2p.) Use part (a) to design a recursive algorithm using the divide and conquer approach to compute $F(n), \forall n \in \mathbb{N}$. The time complexity of the algorithm should be $o(n)$, ie. faster than linear. You may assume that integer addition and multiplication take $O(1)$ time.

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
 P1 = (a11 + a22)(b11 + b22)
 P2 = (a21 + a22)b11
 P3 = a11(b12 - b22)
 P4 = a22(b21 - b11)
 P5 = (a11 + a12)b22
 P6 = (a21 - a11)(b11 + b12)
 P7 = (a12 - a22)(b21 + b22)
 c11 = P1 + P4 - P5 + P7
 c12 = P3 + P5
 c21 = P2 + P4
 c22 = P1 - P2 + P3 + P6
 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)
  M = [1 1]
      [1 0]
  Return BinExponentiation(M, n)[1][0]

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

$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

(c) (2p.) Analyse the time complexity of your algorithm.

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

$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

To find $F(n)$, 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 $O(8(n-1)) = O(8n-8) = O(n)$

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 $O(7)$ time. When n is odd, the two halves are multiplied, plus the multiplication with the remaining matrix, taking $O(14)$ time

When n is even: $T(n) = 2T(n/2) + O(7)$

When n is odd: $T(n) = 2T(n/2) + O(14)$

So on average for both cases: $T(n) = 2T(n/2) + O(10)$

According to Master Theorem: $T(n) = 2T(n/2) + O(10) = O(10\log_2 n) = O(\log_2 n)$

Therefore, the time complexity of new algorithm is $O(\log_2 n) = o(n)$, which runs in logarithmic time. It is faster than the Naïve method that runs in linear time.

2. **Individual exercise: Binomial coefficients computation.** Recall that the binomial coefficient $\binom{n}{k}$ is the number of ways to choose an unordered subset of size k from a fixed set of size n. It can be computed with the following formula:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

(a) (2p.) Binomial coefficients satisfy the following property when $n > k$ and $k \geq 1$.

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

Use this formula to design a recursive algorithm for binomial coefficients.

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

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)!}{k(k-1)!(n-k)(n-k-1)!}$$

$$\Rightarrow \binom{n}{k} = \frac{(n-1)!}{(k-1)!(n-k-1)!} \times \frac{n}{k(n-k)} = \frac{(n-1)!}{(k-1)!(n-k-1)!}\left(\frac{1}{n-k} + \frac{1}{k}\right)$$

$$\Rightarrow \binom{n}{k} = \frac{(n-1)!}{(k-1)!(n-k-1)!(n-k)} + \frac{(n-1)!}{k(k-1)!(n-k-1)!} = \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-1)!}{k!(n-k-1)!}$$

$$\Rightarrow \binom{n}{k} = \frac{(n-1)!}{(k-1)!(n-k)!} + \frac{(n-1)!}{k!(n-k-1)!} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Therefore, the recursive formula is $C(n,k) = C(n-1,k-1) + C(n-1,k)$

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);
```

(b) (1p.) Give the recurrence relation that corresponds to the time complexity of this algorithm. *Hint: You are allowed to use multiple variables in the recurrence relation.*

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

$T(n,k) = T(n-1,k) + T\left(n-1,k-1\right) + O(1)$, because addition arithmetic takes a constant time

(c) (2p.) Use the recurrence relation to compute the time complexity of the algorithm with respect to n. Prove that the complexity is in $o(n!)$.
Hint: For sufficiently large n we have the approximation

$$\binom{2n}{n} \sim \frac{2^{2n}}{\sqrt{n\pi}}.$$

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

At n – 2 level

$$T(n,k) = T(n-1,k) + T(n-1,k-1) + O(1)$$
$$\Rightarrow T(n,k) = \left(T(n-2,k) + T(n-2,k-1) + O(1)\right) + \left(T(n-2,k-1) + T(n-2,k-2) + O(1)\right) + O(1)$$
$$\Rightarrow T(n,k) = T(n-2,k) + 2T(n-2,k-1) + T(n-2,k-2) + O(3)$$

At n – 3 level

$$T(n,k) = T(n-2,k) + 2T(n-2,k-1) + T(n-2,k-2) + O(3) \Rightarrow$$
$$T(n,k) = T(n-3,k) + T(n-3,k-1) + 2\left(T(n-3,k-1) + T(n-3,k-2)\right) + T(n-3,k-2) + T(n-3,k-3) + O(7)$$
$$T(n,k) = T(n-3,k) + 3T(n-3,k-1) + 3T(n-3,k-2) + T(n-3,k-3) + O(7)$$

The sum of the coefficients for all of $T(n-i,k-j)$ is 2 to the power of the current level it is at. The Pascal Triangle below illustrate the sums as the power of 2

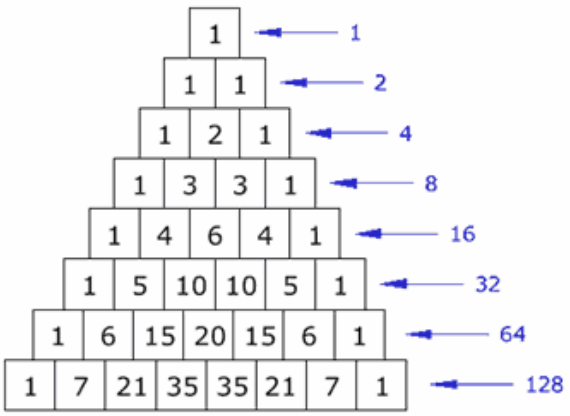

Image source: https://socratic.org/questions/how-do-i-use-pascal-s-triangle-to-expand-x-2-5

Therefore, the approximation of the recursive algorithm is $\max(T(n,k)) = T(n,n/2) = T\left(\frac{2^{2n}}{\sqrt{n\pi}}\right)$ according to the hint => The time complexity is $O\left(2^n\right)$. For the 3 first levels n = 1, 2 and 3, we have $2^n > n!$. However, starting from 4, $n!$ starts to grow asymptotically faster than 2^n. The running time of 2^n thus is strictly faster than $n!$ for sufficiently large n value.

Therefore, $T(n,k) = O\left(2^n\right) = o\left(n!\right)$ (proven)