1. Mathematical Induction

The proof technique of **Mathematical Induction** is widely applicable to prove propositions:

$$\forall n\ P(n), \quad \text{where } n \in \mathbb{N}$$

In the language of propositional logic, this technique can be stated as:

$$[\underline{P(\text{“1”})} \wedge \underline{\forall k(P(k) \implies P(k+1))}] \implies \forall n\ P(n)$$

Example: Prove $T(n) = \lg n + 1$ is the solution of the recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(\frac{n}{2}) + 1 & \text{otherwise} \end{cases}$$

Assume that n $= 2^k$, $k \geq 0$.

1. Base Case:
   Objective: show $P(\text{“1”})$ is true

2. Inductive Hypothesis:
   To-do: assume $P(k)$ is true

3. Inductive Step:
   Goal: prove $P(k+1)$ is true

4. Conclusion: We can prove by mathematical induction that ...

Hint: You might want to attempt *strong (complete) induction*[1] for Question 1 of Warmup Exercises.

[1] Strong induction makes the inductive step easier to prove by using a stronger hypothesis: one proves the statement $P(k+1)$ under the assumption that $P(m)$ holds for all natural numbers m less than $k+1$. In this form of strong induction, one still has to prove the base case, $P(\text{“1”})$, and it may even be necessary to prove extra-base cases such as $P(\text{“2”})$ before the general argument applies.

2. Recursion

The recurrence relation of the Fibonacci numbers is as follows:

$$F_n = \begin{cases} n & \text{if } n = 0 \text{ or } 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

2a. Write pseudocode to compute the n-th Fibonacci number by recursion.

FIB(n)
1 *// Base Case*
2
3
4 *// Recursive Call*
5

2b. Draw a recursion tree for n = 4, and think about how recursion is implemented in memory. [2]

2c. How to solve this recurrence?

[2] In some programming languages, the maximum size of the call stack is much less than the space available in the heap, and recursive algorithms tend to require more stack space than iterative algorithms. Consequently, these languages sometimes place a limit on the depth of recursion to avoid stack overflows, e.g. Python.