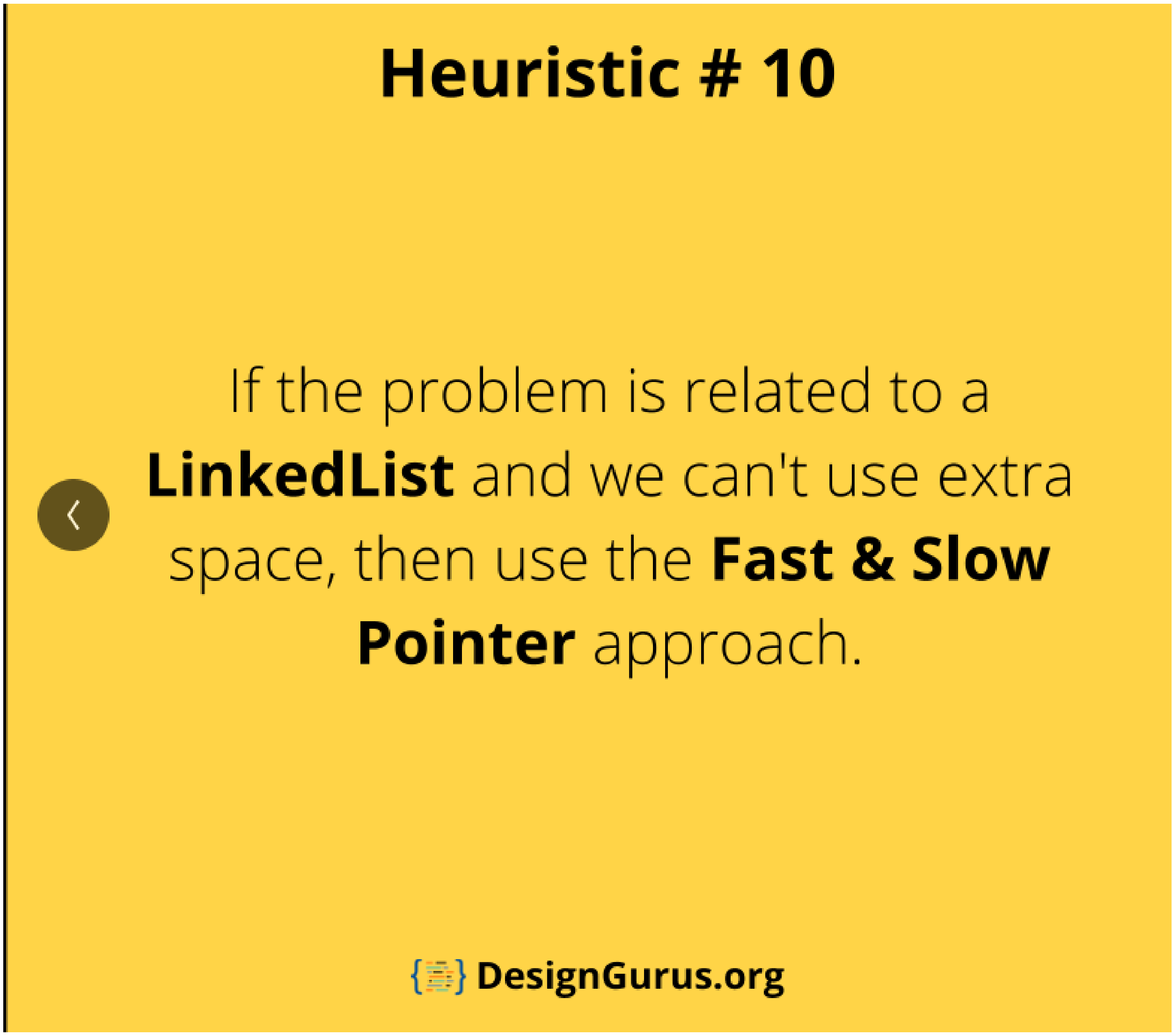
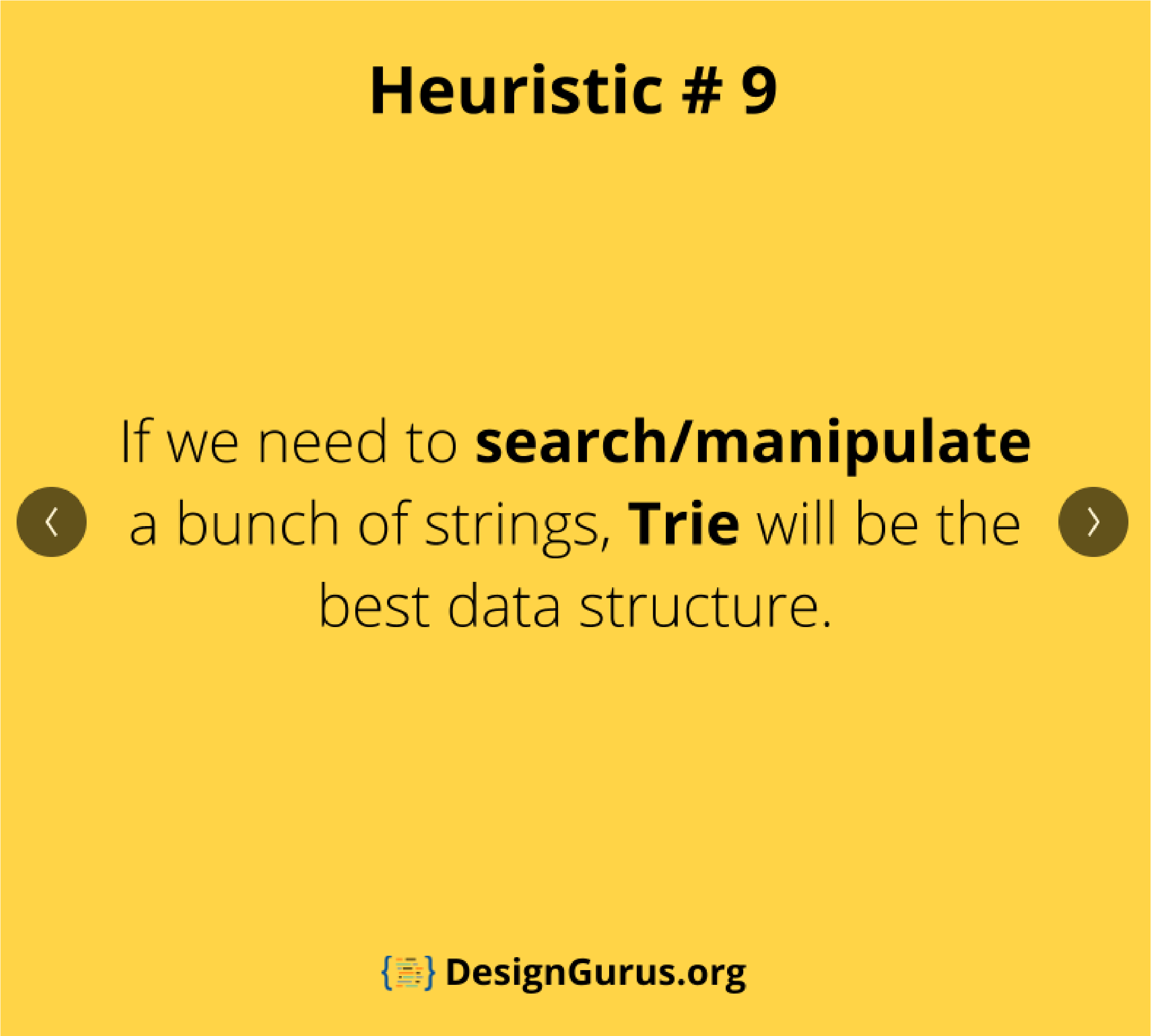


Heuristic # 10

If the problem is related to a **LinkedList** and we can't use extra space, then use the **Fast & Slow Pointer** approach.

DesignGurus.org



Heuristic # 9

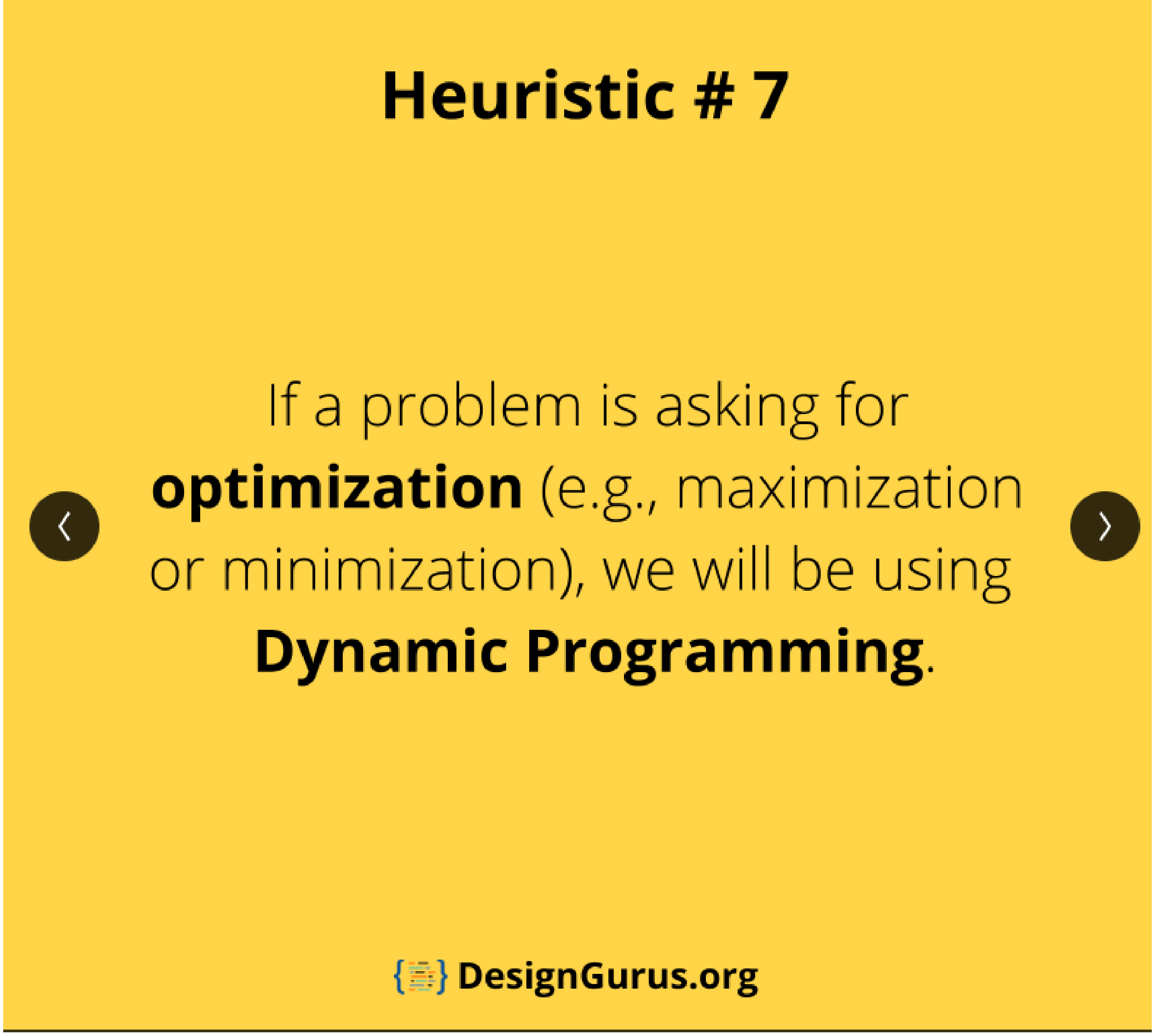
If we need to **search/manipulate** a bunch of strings, **Trie** will be the best data structure.

DesignGurus.org

Heuristic # 8

If we need to find some common **substring** among a set of strings, we will be using a **HashMap** or a **Trie**.

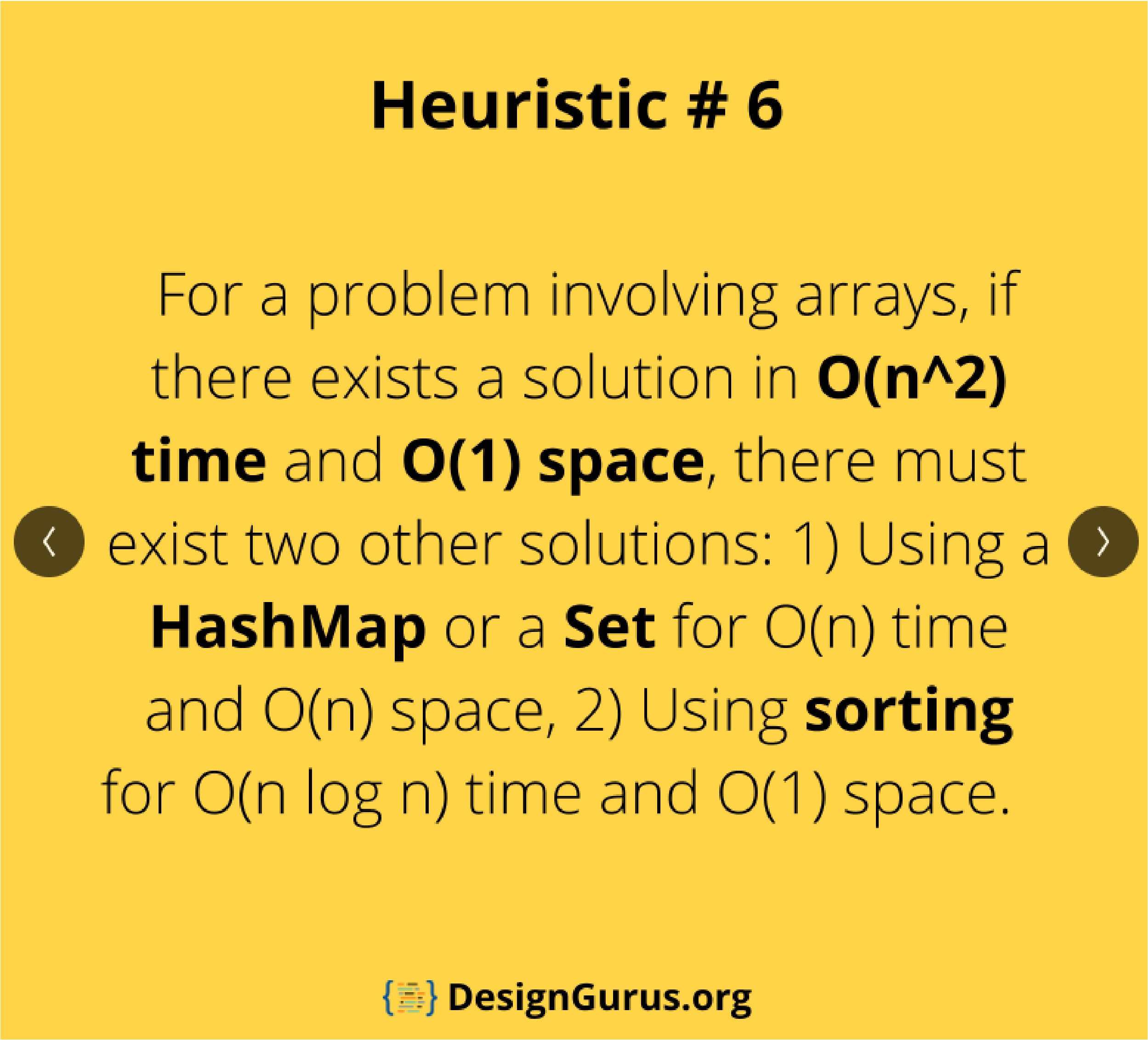
DesignGurus.org



Heuristic # 7

If a problem is asking for **optimization** (e.g., maximization or minimization), we will be using **Dynamic Programming**.

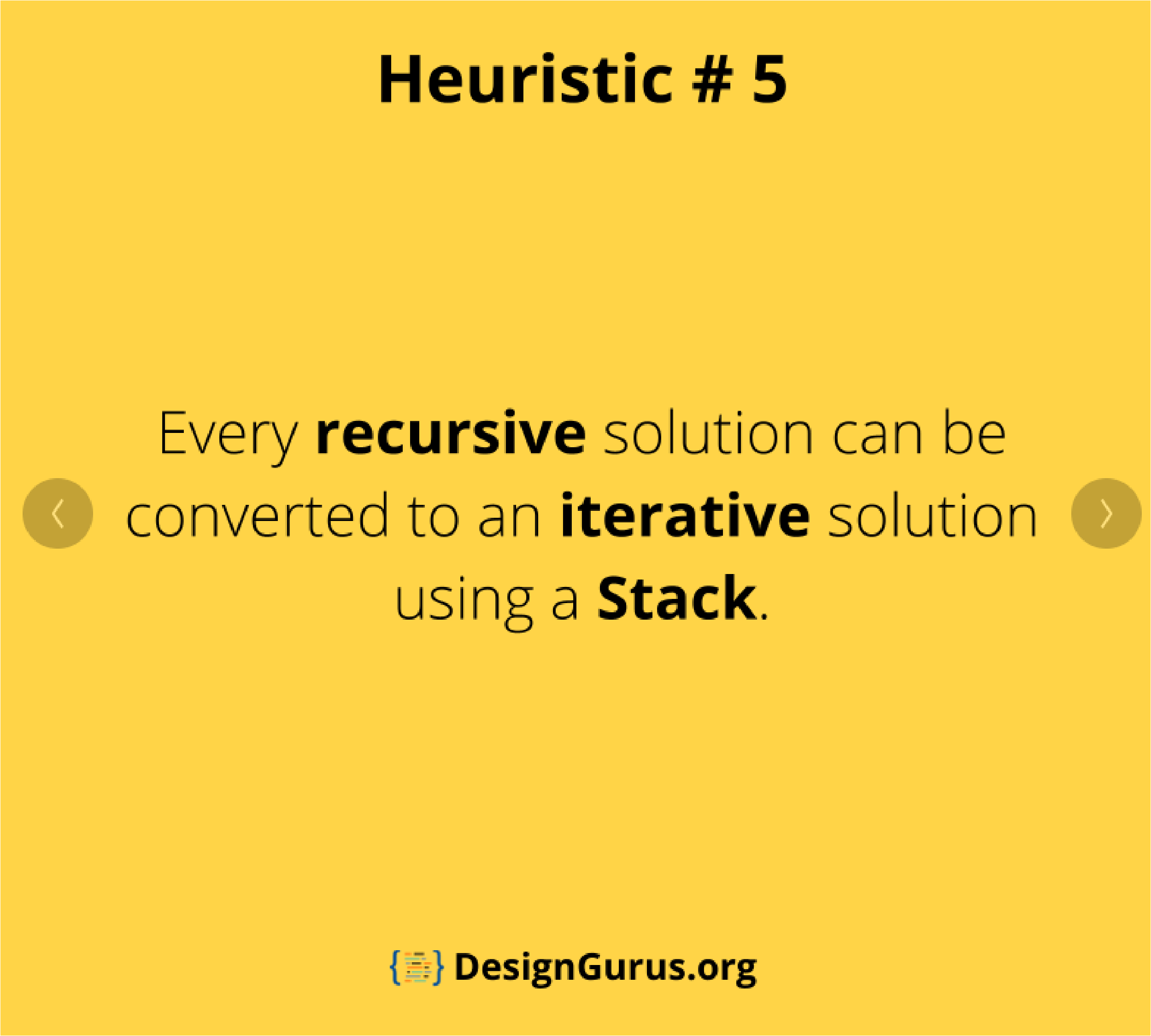
DesignGurus.org



Heuristic # 6

For a problem involving arrays, if there exists a solution in **O(n^2) time** and **O(1) space**, there must exist two other solutions: 1) Using a **HashMap** or a **Set** for O(n) time and O(n) space, 2) Using **sorting** for O(n log n) time and O(1) space.

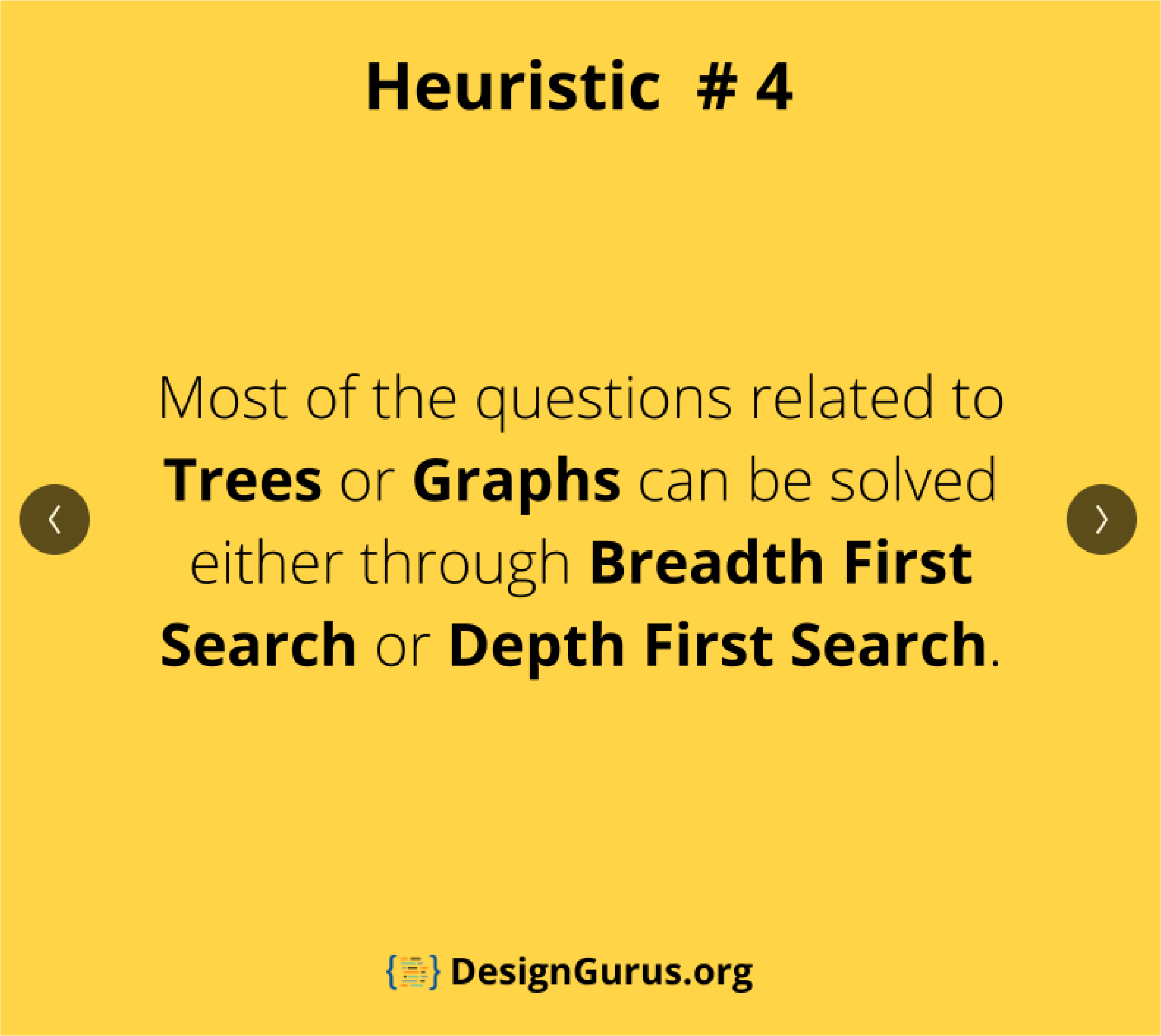
DesignGurus.org



Heuristic # 5

Every **recursive** solution can be converted to an **iterative** solution using a **Stack**.

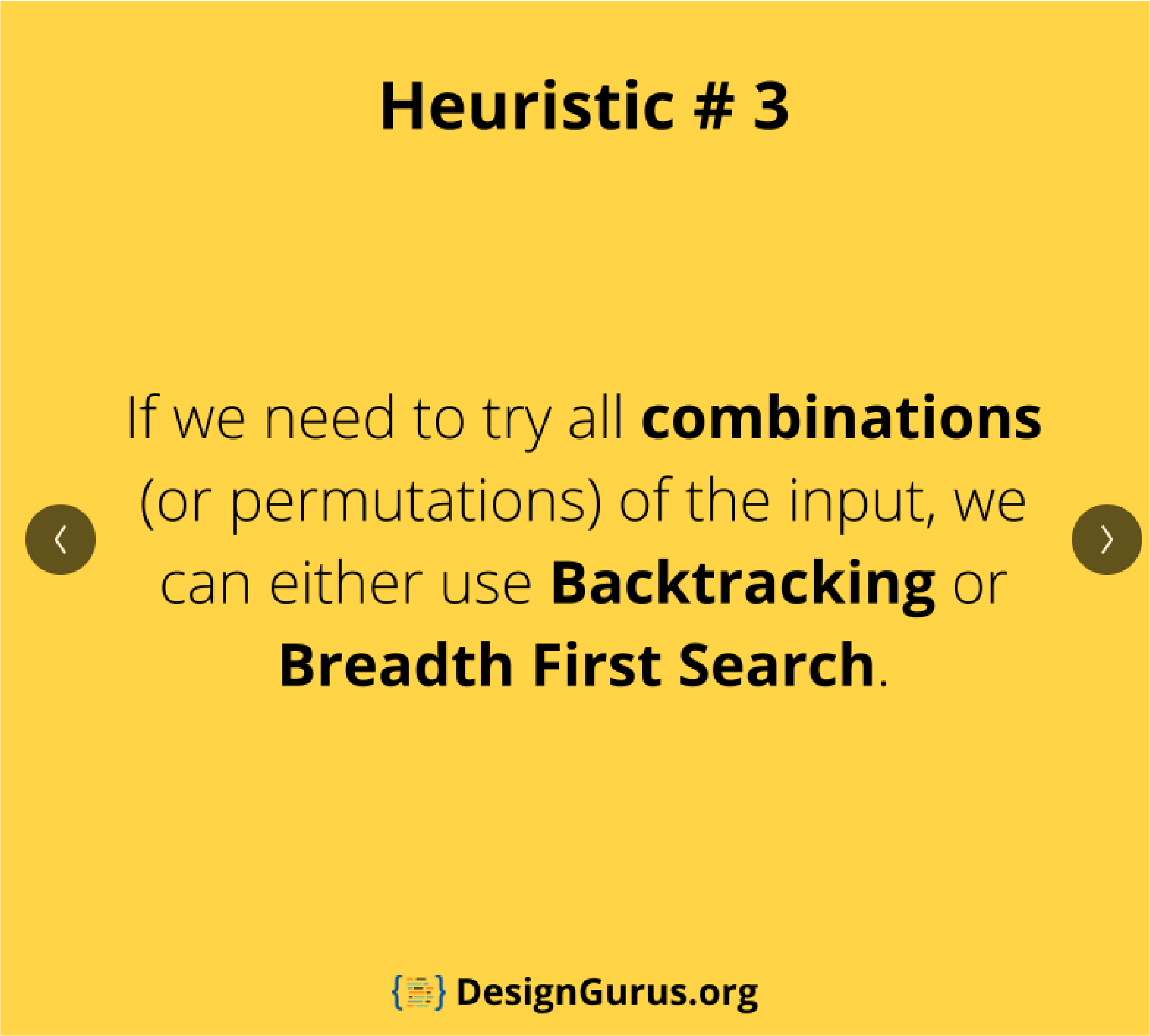
DesignGurus.org



Heuristic # 4

Most of the questions related to **Trees** or **Graphs** can be solved either through **Breadth First Search** or **Depth First Search**.

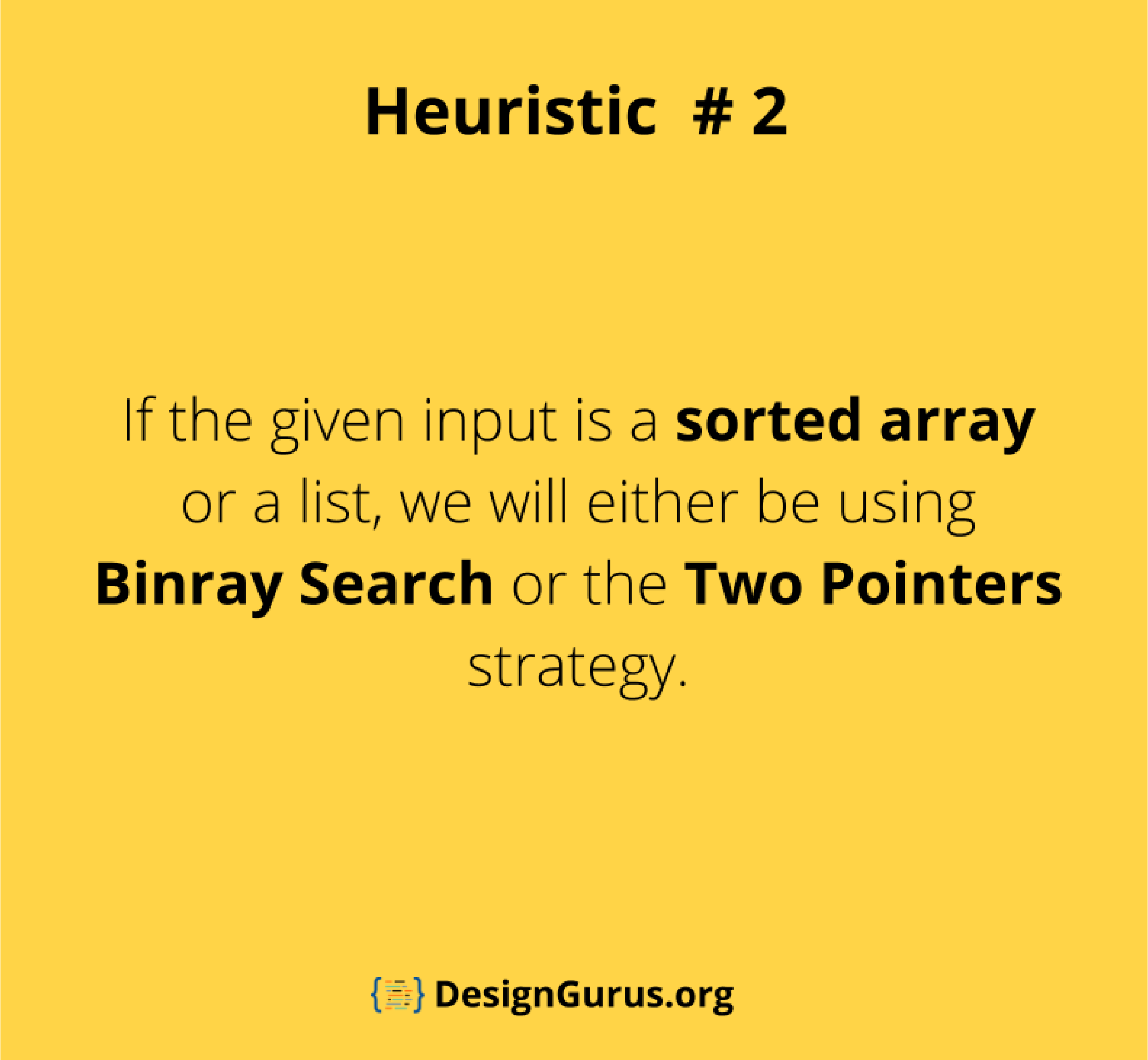
DesignGurus.org



Heuristic # 3

If we need to try all **combinations** (or permutations) of the input, we can either use **Backtracking** or **Breadth First Search**.

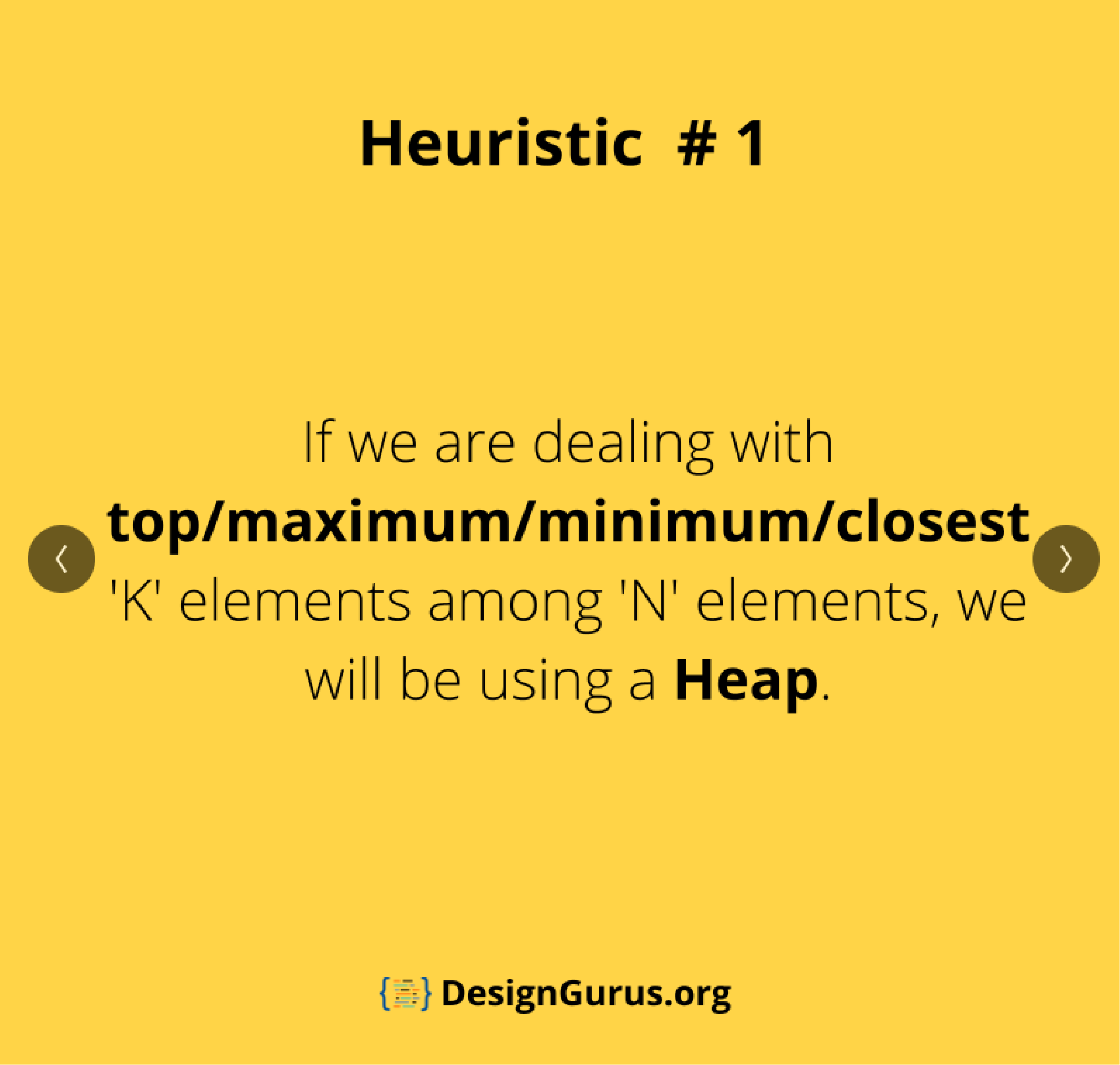
DesignGurus.org



Heuristic # 2

If the given input is a **sorted array** or a list, we will either be using **Binray Search** or the **Two Pointers** strategy.

DesignGurus.org



Heuristic # 1

If we are dealing with **top/maximum/minimum/closest** 'K' elements among 'N' elements, we will be using a **Heap**.

DesignGurus.org

10 Golden Rules

for

Solving Coding Questions

DesignGurus.org