Task – Tower of Hanoi

The Tower of Hanoi is a classic mathematical puzzle or game that was invented by the French mathematician Édouard Lucas in 1883. The game consists of three towers, and a number of disks of different sizes, which can slide onto any tower. The puzzle starts with the disks in a neat stack in ascending order of size on one tower, the smallest at the top, making a conical shape.

Objective:

The objective of the puzzle is to move the entire stack to another tower, following these simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or an empty tower.
- Disk can only be placed on top of another disk of a larger size, or an empty tower.

The challenge is to accomplish the objective in the fewest possible moves. As the number of disks increases, the problem becomes exponentially more complex.

Below is an UML class diagram to create a Tower of Hanoi game.

```
Tower
-----------------------------
- disks: list
-----------------------------
+ Tower()
+ push_disk(disk: int)
+ pop_disk(): int
+ peek_top_disk(): int
+ __str__(): str
```

```
TowerOfHanoi
-----------------------------
- towers: dict
- n: int
- step: int = 1
-----------------------------
+ TowerOfHanoi(n: int)
+ get_tower(tower: str): Tower
+ get_n(): int
+ reset_step()
+ get_step(): int
+ reset_towers()
+ print_towers()
+ check_win(): bool
+ move_disk(src: str, dst: str)
+ solve(n: int)
```

Implement the classes based on the following descriptions. Lastly, create a game menu to facilitate user interaction:
[You may assume that all inputs are valid]

```
Please choose one of the following options:
1. Start/Reset a new game
2. Move a disk
3. Auto-solve game
4. Quit
```

Attributes/Methods	**Description**
`Tower Class`	
`- disks: []`	`Tower` class is a Stack data structure, which uses a list to store `int` values representing the size of the disks. E.g: `[3, 2, 1]` Represents that lowest disk has a size of `3`, and top disk has a size of `1`.

`+ push_disk(disk: int)`
`+ pop_disk(): int`
`+ peek_top_disk(): int`
	`push`, `pop` and `peek` operation for the stack of disks.

`+ __str__(): string`	String method for the class, to simply cast the list of int values to a str. E.g: `[3, 2, 1]`
`TowerOfHanoi Class`	
`- n: int`	`n` refers to the number of disks of the first tower.
`- step: int = 1`	`step` is to track the number of steps taken by the user.
`- towers = {"A": Tower(), "B": Tower(), "C": Tower()}`	`towers` is a dict, storing 3 `Tower` objects, to be referenced by the strings of `"A"`, `"B"`, `"C"` respectively.
`+ get_tower(tower: str): Tower`	Use the string `"A"`, `"B"`, `"C"` to access the `Tower` objects.

`+ get_n(): int`
`+ get_step(): int`
	Getter for `n` and `step`.

`+ reset_step()`	Reset the `step` value back to `1`.
`+ reset_towers()`	Reset 3 towers to new `Tower` objects, and based on value of `n`, add a list of disks to `Tower A`. e.g. when `n = 3`, `Tower A` should have the following disks inside: `[3, 2, 1]`

`+ print_towers()`	Print the 3 towers in the following format:
	`Tower A: [3, 2, 1]`
	`Tower B: []`
	`Tower C: []`

`+ check_win(): bool`	Check if the players has won the game. If so, print:
	`You win!`
	`Total steps taken: 7`
	`Optimal steps expected (2^3 - 1): 7`

`move_disk(src: str, dst: str)`	Move disk from src to dst tower.
	1. check if src tower is empty. Print error msg if it is empty.
	2. check if top disk of src can be moved to dst:
	- if dst is empty, it's ok to move
	- if top of dst is larger than top disk of src, then it's ok to move.
	- if it is not a valid move, print error msg.
	3. move disk from src to dst
	4. print step count and movement in the following format:
	`Step 1`
	`Moving disk from A to C`
	`Tower A: [3, 2]`
	`Tower B: []`
	`Tower C: [1]`
	5. check if game is won
	6. increment the step value

`+ solve()`	1. reset the towers
	`Solving for 3 disks`
	`Starting arrangement:`
	`Tower A: [3, 2, 1]`
	`Tower B: []`
	`Tower C: []`
	
	2. solve the game automatically, by printing:
	
	`Step 1`
	`Moving disk from A to C`
	`Tower A: [3, 2]`
	`Tower B: []`
	`Tower C: [1]`
	
	`Step 2`
	`Moving disk from A to B`
	`Tower A: [3]`
	`Tower B: [2]`
	`Tower C: [1]`
	
	…
	
	`Step 7`
	`Moving disk from A to C`
	`Tower A: []`
	`Tower B: []`
	`Tower C: [3, 2, 1]`
	
	`You win!`
	`Total steps taken: 7`
	`Optimal steps expected (2^3 - 1): 7`