Heuristic Function:

Price of State's closeness to the goal

The heuristic *h(n)* measures how far the current state *n* is from the goal by calculating the price of **incorrectly** placed marbles.
This function reflects how much 'effort' or 'cost' remains to transform current state to goal state.

$$h(n) = \sum_{i \in \text{Marbles}} \begin{cases} w_i & \text{if marble } i \text{ is not in its correct position} \\ 0 & \text{otherwise} \end{cases}$$

Where:

- h(n) : Heuristic value for the state n .
- w_i : Weight (cost) of moving marble i .

Intuitive Interpretation

- Each marble that is not in its correct position must be moved **at least once** to reach its goal position.
- The heuristic assumes that the **minimum** possible effort is sufficient – meaning it only counts the first required move for each incorrectly placed marble.
- If a marble needs to move multiple times, the heuristic does not account for those additional moves.
- As a result, the heuristic function *h(n)* **underestimates** the total cost to the goal *opt(n)*, but it is guaranteed to be **at least** *0* and **never greater than** *opt(n)*.
  - If all marbles are in their correct places, *h(n) = 0*, indicating the goal state.
  - If marbles are misplaced, the sum of their weights provides **a lower bound** on the true cost.

Admissibility

- The heuristic *h(n)* is admissible because it never overestimates the true cost *h(n)*.
- Since *h(n)* only accounts for the minimum effort (a single move per incorrectly placed marble), it provides a lower bound on *h(n)*.

$$h(n) \leq h^*(n)$$

Consistency

- Moving one marble reduces the heuristic value *h* by **at least** the marble's weight w_i , and the cost of this move is exactly w_i .
- Thus, the condition $h(n) \leq c(n, n') + h(n')$ holds for every successor n'.

This ensures that *h(n)* is both **admissible** and **consistent**, making it suitable for algorithms like A*, IDA*, and DFBnB.

Complexity Analysis

The heuristic function *h(n)* iterates over the entire state (the board) to check for incorrectly placed marbles and sums their weights.

- Time Complexity:
    - For a board with *N* positions (e.g., a 3x3 board where *N = 9*), the function requires *O(N)* time to evaluate the heuristic.
    - This is because each position must be checked to determine whether the marble is in its correct place.
- Space Complexity:
    - The space complexity is *O(1)*, as no additional data structures are used to evaluate the heuristic beyond a constant number of variables.
    - The function directly operates on the state representation.

Java Code

```java
public int calculateHeuristic(State goal) {
    int heuristic = 0;

    for (int i = 0; i < board.length(); i++) {
        char marble = board.charAt(i);

        // Ignore empty or blocked cells
        if (marble == '_' || marble == 'X') {
            continue;
        }

        // Add the cost of the marble if it's not in the correct position
        if (marble != goal.board.charAt(i)) {
            heuristic += calculateMarbleCost(marble);
        }
    }
    return heuristic;
}
```