1 Divisor Game

This document was created by Ali Bozkurt. Github

1.1 Problem Statement

Given an integer n, the task is to determine the winner of the Divisor Game.
For full description click

1.2 Approach

Let n be the current integer representing the game state.

Consider a recursive function $f(n)$ that determines if the current player (e.g., Alice) wins the Divisor Game when the current number is n.

The logic for $f(n)$ in mathematical terms:

$$f(n) = \begin{cases} \text{true} & \text{if there exists at least one factor } x \text{ of } n \text{ such that } f(n-x) = \text{false} \\ \text{false} & \text{if for all factors } x \text{ of } n,\ f(n-x) = \text{true} \end{cases}$$

This recursive function $f(n)$ evaluates the game outcome based on the factors of n. If there is at least one factor x that leads to a state where the opponent loose ($f(n-x) = \text{false}$), the current player wins. Otherwise, if all possible moves lead to the opponent's win ($f(n-x) = \text{true}$), the current player loses.

This logic determines the winner of the Divisor Game based on the recursive nature of the game strategy.

Cases

- **Base Case (n = 1):** Alice definitely loses since there's no available factor. Since Alice loose $f(1) = \text{false}$
- **Case for n = 2:** The only factor is 1. Alice's state will be determined by the outcome of $f(2-1)$. If $f(2-1)$ is true Bob wins and Alice loose. Otherwise, Alice wins. As stated in the previous case $f(2-1) = \text{false}$, so Alice wins. Since Alice wins $f(2) = \text{true}$
- **Case for n = 3:** The only factor is 1. Alice's decision is based on $f(3-1)$. If $f(3-1)$ is false Alice wins; otherwise, Bob wins. As stated in the previous case $f(3-1) = \text{true}$, so Alice loose. Since Alice loose $f(3) = \text{false}$
- **Case for n = 4:** There are two factors : 1 and 2. Lets evaluate them:

  - **Choosing 1:** If the result of the recursive function $f(4-1)$ (which evaluates to $f(3)$) returns false, then Alice wins decisively. However, if it returns true, we cannot determine the outcome definitively because we must also consider the situation for the other factor, which is 2(choosing2).

- **Choosing 2:** If the result of the recursive function $f(4-2)$ (which evaluates to $f(2)$) returns false, then Alice wins definitively without considering factor 1. However, if it returns true, the situation where factor 1 is selected should be evaluated.

  If both situations are true, Bob definitely wins, and consequently, Alice loses.

  Since $f(3)$ return false, without looking other case of $f(2)$, Alice wins definitely. Furthermore, in question it is stated that *'both players play optimally'*

1.3 Brute Force Way

you can find the brute force solution below written by csharp language:

```
// Solution.cs
  public bool recursive(int n)
  {
     // base case
     if (n == 1) return false;
     else if (n < 1) return false;
     for (int i = 1; i < n; i++)
     {
        if (n % i == 0)
        {
           if (recursive(n - i) == false) return true;
        }
     }
     return false;
  }
```

1.4 Memoization Way

you can find the solution below written by csharp with memoization technique

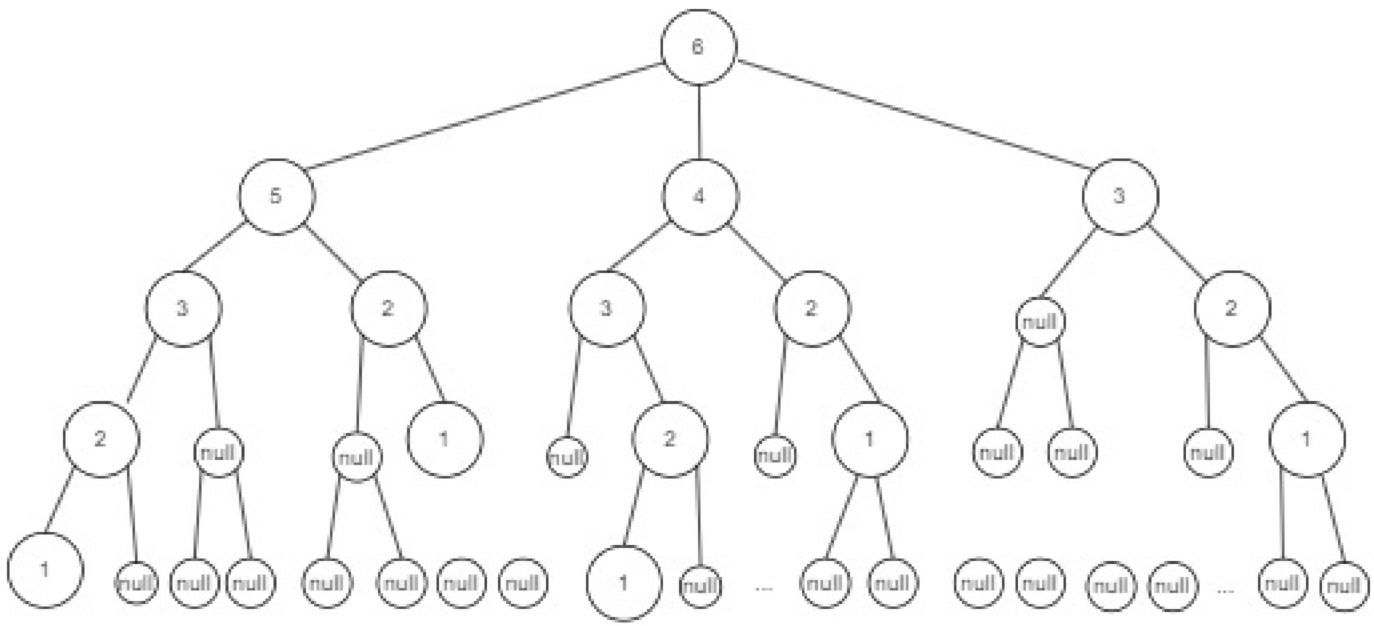
```
// Solution.cs
  public bool recursive(int n, Dictionary<int, bool> MemoArray)
  {
     // base case
     if (n == 1) return false;


     else if (n < 1) return false;
     if (MemoArray.ContainsKey(n))
        return MemoArray[n];
```

```
    for (int i = 1; i < n; i++)
    {
       if (n % i == 0)
       {
          if (recursive(n - i, MemoArray) == false)
          {
             if (!MemoArray.ContainsKey(n - i))
                MemoArray.Add(n - i, true);
             return true;
          }
       }
    }
    MemoArray.Add(n, false);
    return false;
 }
```

1.5 Tree Visualization

For n = 6 you can see the tree visualisation below(For better resolution click).



1	1.2
3	3
6	3.2
12	3.2^2
24	3.2^3
.	
.	
.	
.	

Number of leaf nodes: **τ(n)(2ⁿ)**

τ(n) is takes a positive integer as its input and gives the number of positive divisors

1.6 Complexity Analysis

- Time Complexity for Brute Force : $O(n^2)$
- Space Complexity for Brute Force : $O(n)$
- Time Complexity for Memoization : $O(\tau(n) \times n)$
- Space Complexity for Memoization : $O(n)$

Note: The function $\tau(n)$ counts how many divisors n has