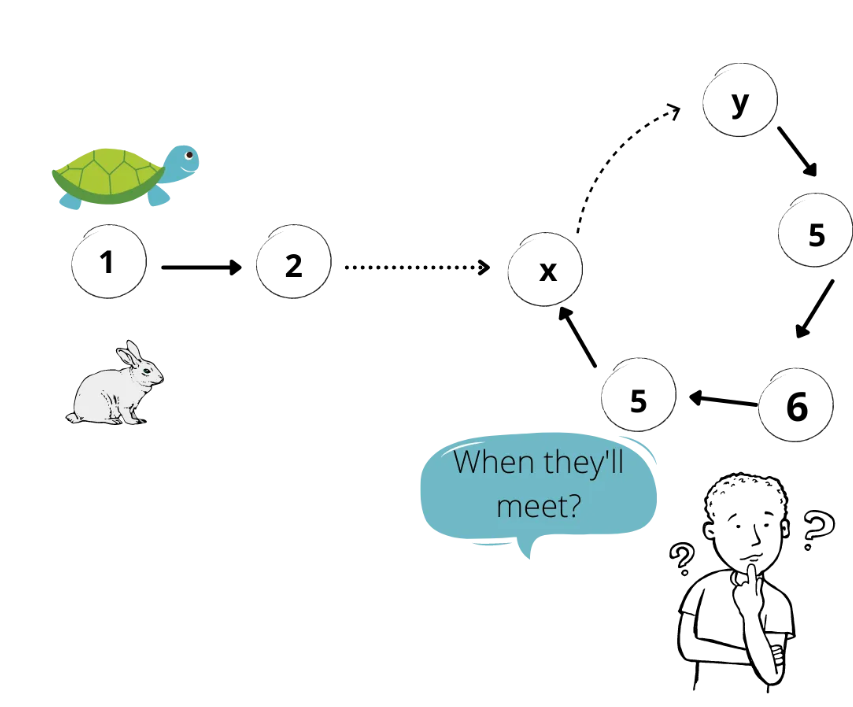
**Finding the Duplicate Number using Floyd’s Tortoise and Hare Algorithm**

Tortoise and Hare algorithm, commonly known as Floyd’s cycle detection algorithm is a pointer algorithm that uses two pointers, which move through the sequence at different pace.

One of the most used applications of it is to detect a loop or cycle in linked list. Let’s first understand this useful algorithm with a visual example:

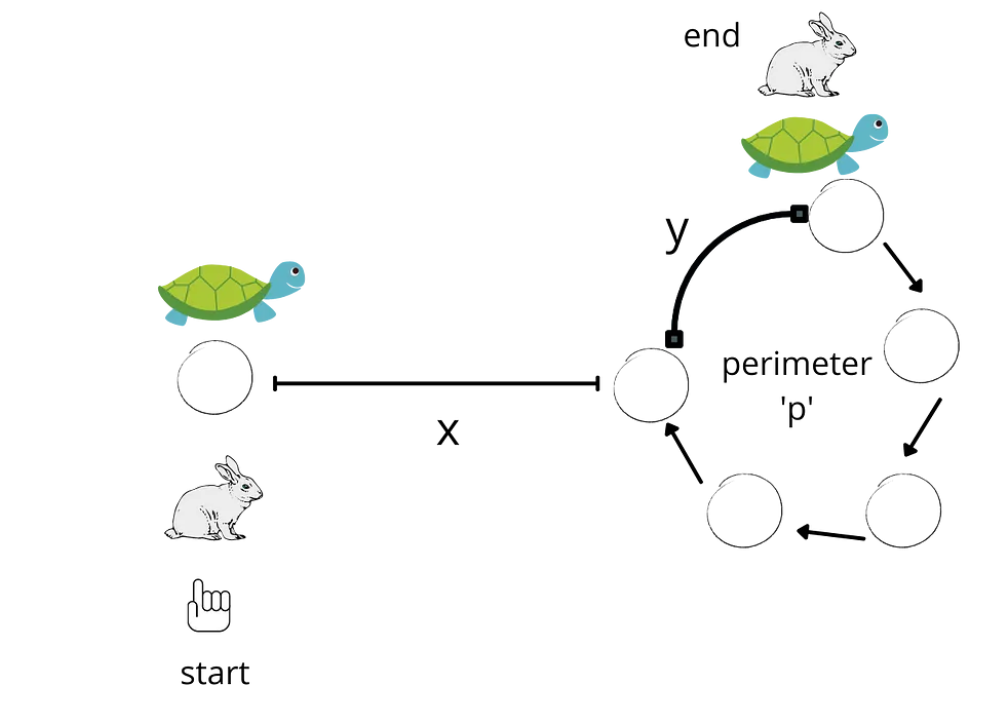


In a given sequence, there could be two possibilities, either there is a cycle or not.

# ****Detect the Cycle****

Assuming there is a hare and a tortoise at the beginning and the hare moves 2 steps at one time while the tortoise moves 1 step. If there is a cycle in the sequence, they both will meet finally at some point. Or if there’s no such a cycle, they will both reach the end.

# ****Where does the cycle start from?****



Initially, the hare and tortoise are at the start of the sequence. The hare moves 2 steps at a time while the tortoise moves 1 step. Assuming they meet after k iterations, the length of the non-cyclic part is x, circumference of the cycle is p, and the distance between the beginning of the cycle to the place they meet is y, we have:

* k = x + a \* p + y. a is a non-negative integer representing the number of loops that tortoise traveled, and k is the distance that the tortoise moved.
* 2 \* k = x + b \* p + y. b is another non-negative integer representing the number of loops that hare traveled, and 2 \* k is the distance that the hare moved.

On solving the above two equations, we get: **k = (b — a) \* p**

Now we put the hare back to the beginning and both hare and tortoise move 1 step at a time.

The place where they meet is the start of the cycle. This is because when the hare travelled another x, the tortoise is at the k + x = (b — a) \* p + x.

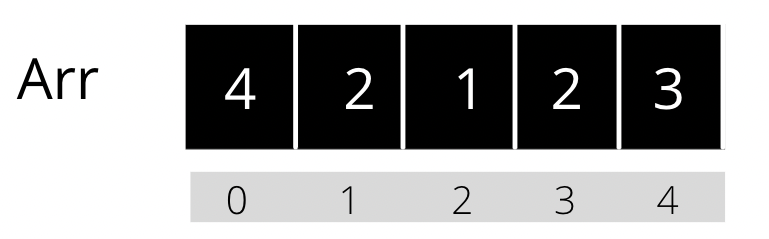
Now, we have understood the working of this algorithm, we can move forward to the code.

Given an array of integers named **Arr** containing n+1 integers where each integer is in the range [1,n] inclusive. There is only **one repeated number** in **Arr**, return this duplicate number.

int findDuplicateNum(vector<int>& Arr) {  
   
 int slow = Arr[0];  
 int fast = Arr[0];  
 do {  
 slow = Arr[slow];  
 fast = Arr[Arr[fast]];  
 }while(slow != fast);  
   
 fast = Arr[0];  
 while(slow != fast) {  
 slow = Arr[slow];  
 fast = Arr[fast];  
 }  
 return fast;  
   
 }

Here, the slow pointer resembles the tortoise moves and the fast pointer is resembling hare moves.

Let’s take an example:



Here, slow = Arr[0] = 4, and fast = Arr[0] = 4. At the end of do-while loop, slow = 2, and fast = 2. After this, we reset fast to point to 1st element of the sequence. The slow pointer won’t be reset. We will loop through the sequence until slow != fast. At the end, we will get the duplicate number.

# Complexities

The time complexity of this algorithm is Linear: O(N)

The space complexity is constant: O(1)