Naive Solution (Sorting)

- Sort the array and check
  if (arr[i] == arr[i+1])
  arr[i] is the repeated element

T.C → O(nlogn)   S.C → O(1)

Optimised Solution. (Frequency map)

- Create a map and store the frequencies of elements of array
- Traverse the map and find the key with value 2
- That key is the repeated element

T.C → O(n)   S.C → O(n)

Best Solution (Turtle-hare approach)

(Commonly used to find cycles in a linked list)

Approach: $a = [\,1, 3, 4, 2, 2\,]$

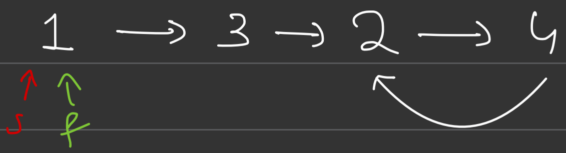
$\quad\quad\quad\quad\quad\quad\;\; 0 \quad 1 \quad 2 \quad 3 \quad 4$

Initialise 2 pointers slow & fast to a[0]

Suppose ptr is at i, ptr moves to a[i]

For above array, the ptrs follow

Step I : Initialise $s = a[0]$; $f = a[0]$

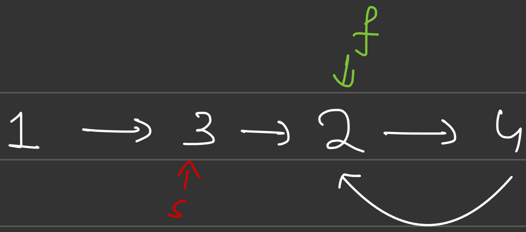


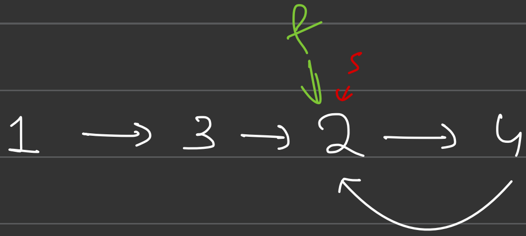
To find the cycle,

Step II : slow pointer travels 1 step ($s = a[s]$)

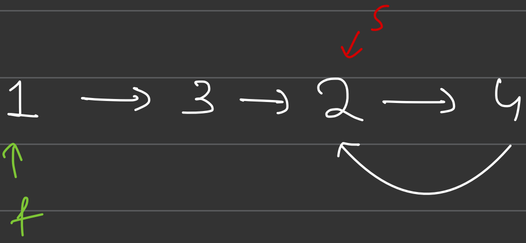
fast pointer traves 2 steps ($f = a[a[f]]$)

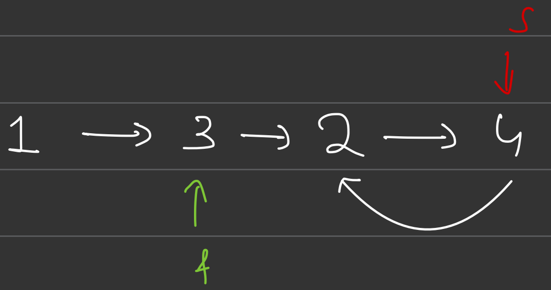
till they meet at a same point

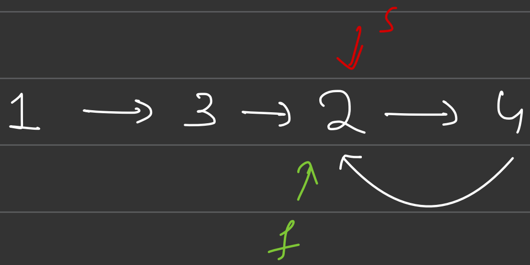




Step III : Take fast ptr to start & now both ptrs move 1 step till collision
(s = a[s] , f = a[f])







∴ 2 is the repeated element

Algorithm.

Step I: Initialisation

```
slow = a[0]; fast = a[0];
```

Step II: fast → 2x speed

```
do {
    slow = a[slow];
    fast = a[a[fast]];
} while ( slow != fast);
```

Step III: fast → 1x speed.

```
fast = a[0]

while( slow != fast) {
    slow = a[slow];
    fast = a[fast];
}
return slow;
```

T.C → O(n) ; S.C → O(1)