↳ Not an intuitive question. (Floyds)

Q 11 Find the duplicate number in an array of n+1 integers where numbers are from [1, n].

i/p → {1, 4, 4, 2, 4}

o/p → 4

* The simplest approach can be like running 2 nested loops and then finding a number which is equal to number present at i^{th} index & j^{th} index (starting from $i+1$) but this has time complexity of $O(n^2)$.

* We can make a frequency / count array and then store the frequency of each number in that array & if we find any number having frequency > 1, that will be the answer. It has time & space complexity of $O(n)$.

* Now the approach of constant space can be just sort the array & then compare the adjacent elements & if found equal, then simply return that number. Time complexity of the solution will be $O(n \log n)$.

Dry run

1) Sort the array

{1, 2, 4, 4, 4}

2) Start from $i = 1$ till size of array &

compare $(i-1)^{th}$ index element and i^{th} index element. If found equal store it & simply come out of loop by break.

(i) $i = 1$, ans = arr[0] = 1
arr[i-1] = arr[0] = 1
arr[i] = arr[1] = 2
$1 \neq 2$ hence simply move to next element

(ii) $i = 2$, ans = arr[0] = 1
arr[i-1] = arr[1] = 2
arr[i] = arr[2] = 4
$2 \neq 4$ hence simply move to next element

(iii) $i = 3$, ans = arr[0] = 1
arr[i-1] = arr[2] = 4
arr[i] = arr[3] = 4
$4 = 4$ hence ans = arr[i] = 4 & break

3) Simply return ans i.e 4 in this case.

Code

```
int findDuplicate (vector <int>& nums) {
    //Sort the vector
    sort (nums.begin(), nums.end());
    //Random guess
    int ans = nums[0];
    //Comparing the adjacent elements
for (int i=1; i<nums.size(); i++) {
        if (nums[i-1] == nums[i]) {
            ans = nums[i];
            break; // 1 duplicate is there
```

```
}
return ans; // Duplicate element is returned
}
```

Time complexity = O(n log n)
Space complexity = O(1)

Note→ In the question there were 2 constraints,
(i) no modification in array
(ii) no extra space to be used

The above 2 constraints can be made sure with fast & slow pointer approach which we will be discussing in linked list.

Floyd's detection (Cycle detection)

0	1	2	3	4
1	3	4	2	2

Think of every value as the pointer.
arr[0] = 1
pointing to 1st index.

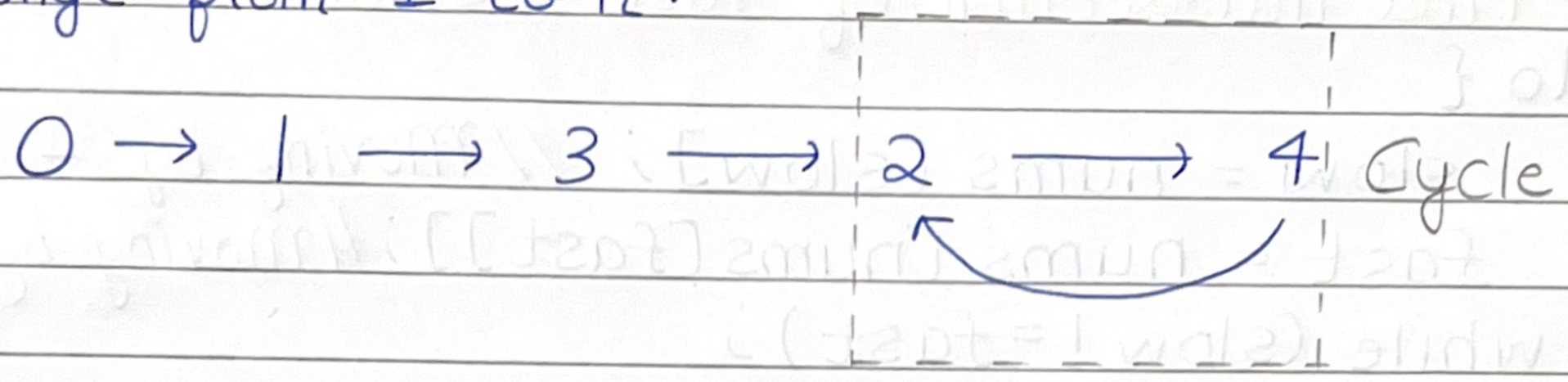
arr[1] = 3
pointing to 3rd index

arr[3] = 2
pointing to 2nd index

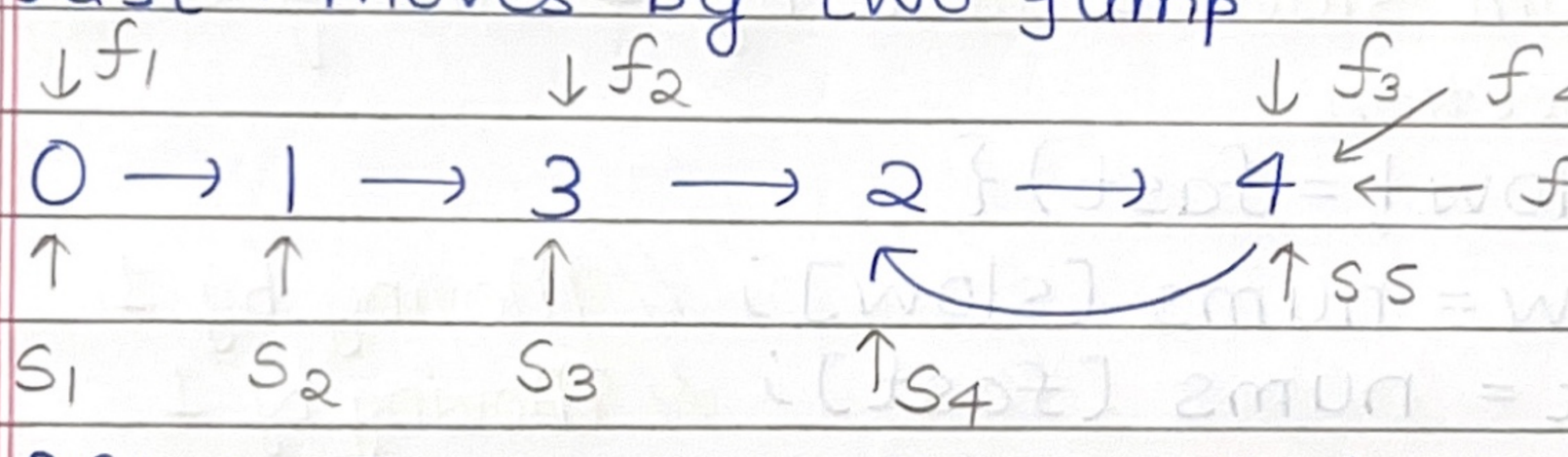
arr[2] = 4
pointing to 4th index

arr[4] = 2
again pointing to 2nd index & hence here

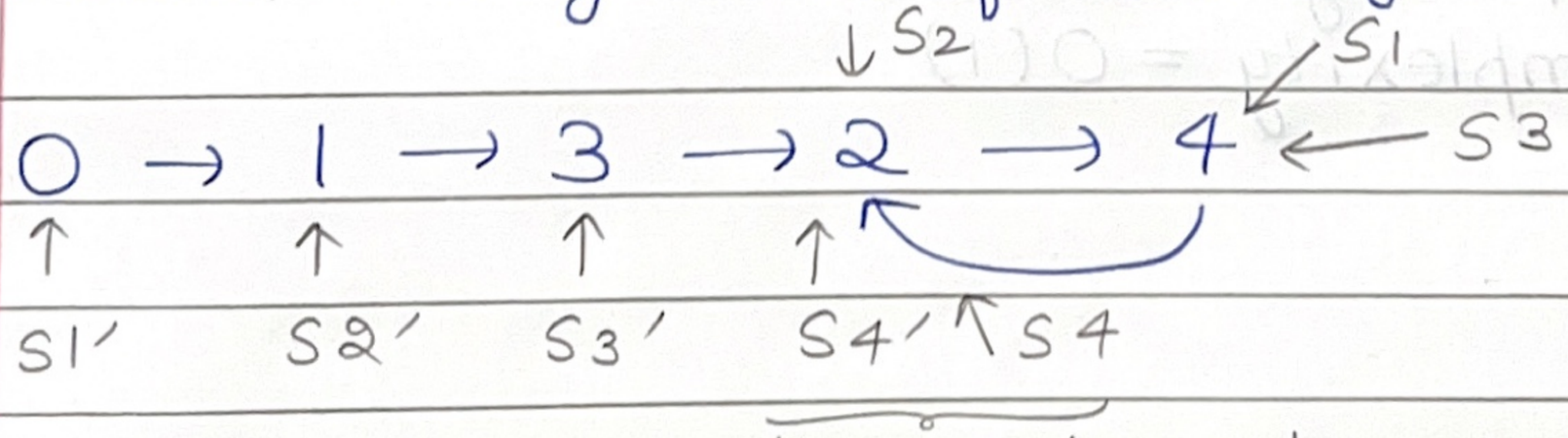
we got the duplicate value.
Also we can say that we won't be getting any value pointing to 0th index as numbers are in range from 1 to n.



slow and fast pointer
slow → moves by one jump
fast → moves by two jump



After 5 jumps of slow & 10 jumps of fast, they both intersect & hence this is the intersection point.
Now take 2nd slow pointer at the start & Start moving both of them by 1.



Hence we found the starting point of the cycle.

Code

```
int findDuplicate (vector <int> &nums) {
```

↳ Range → [1, n]

```
// 0th index not pointed by anyone
int slow = 0;
int fast = 0;
// Find intersection of fast & slow
do {
    slow = nums [slow]; //Moving by 1
    fast = nums [nums [fast]]; //Moving by 2
} while (slow != fast);
//Place slow to 0.
slow = 0
//Move both slow & fast by 1 and find
the intersection
while (slow != fast){
    Slow = nums [slow]; //Moving by 1
    fast = nums [fast]; //Moving by 1
}
return slow; //fast can be returned
}
```

Time complexity = O(n)
Space complexity = O(1)