By: Ehsan Kourkchi (Sep., 20, 2013)

Algorithm Analysis

1. Method `'find'`

To find the `needle` in the `haystack` array, this method starts to check all the array elements to find the `haystack`. This is done in a for loop. In the worst case the last array member has the value of `needle` and in this case we have to loop over all array elements. If we double the number of the array elements, we need to spend as twice time to check all the array elements to find the `needle`. Therefore, the runtime for this method is linearly changed with the `haystack` array size and is **O(n)**.

2. Method `'find_duplicate'`

In this method, the index of the first element which is duplicated would be returned. This method uses two for loops (loop-in-loop) to cycle around the input array and compare each two elements.

The process is as following. In each step the entire process can be halted as soon as two members are found to be the same and the index of the element at which the equality happens is returned. The worst case is to compare all pair of elements in the array.

1) The first element (`index=0`) is compared with all other elements. This needs `n-1` comparisons, where `n` is the number of array elements.

2) The second element is compared with all other elements with `index>1`. Since the first and second elements have been compared in the previous step, they are not compared again in this step to save time. In this step `n-2` comparison would be done.

3) The third element would be compared with the other `n-3` elements which have `index>2`.

4) Take all the comparisons all the way to reach to the one before the last element which has the index of `n-2` and is compared with the last element (`index=n-1`).

The total number of the comparisons in the worst case then would be

$$N = (n-1) + (n-2) + (n-3) + \ldots + 1$$

$$N = 0.5 * (n-1) * (n-2) = O(n^2)$$

Therefore, the runtime for the fine_duplicate method has a quadratic relationship with the size of the input array, **O(n^2)**.