Elad Ohayon

***Algorithm*: DetectTerrorist**

Input: passengers[n], an array where all elements are equal except one which is less than the rest, and represents a terrorist

Output: the index where the terrorist lies

Solution: This algorithm employs a divide and conquer strategy to divide the input into equal sets of 2 and when the ultimately recursion unwinds, it returns the index where the terrorist lies hiding. The "sets" mentioned throughout this write-up are a custom class which contain the weight of the 2 sets combined and which indices were measured with start and end representing these indices. Since it is known the terrorist's tefillin bag is lighter, the algorithm compares these sets and finds out which one's weight is lighter than the rest. The recursive method has a base case that returns a "set" object when start-end =1 and as the recursion unwinds, we compare the 2 sets returned and return the smaller weight one if it exists or a new containing the first set's end which is the second set's start if they are equal in weight. This algorithm has two edge cases it successfully deals with: the passengers array has no terrorist and passenger.length==3. The algorithm solves this by checking how many times the weights are equal and if they are equal for every comparison then it knows there is no terrorist as all bags are of equal weight. In the case of passenger.length==3, the algorithm does an additional compare between the middle and the weight accumulated from measuring the tefillin bags in position 0 and 2 to determine the terrorist's existence.

Proof of Correctness:

Claim: My algorithm will always find the terrorist and return its index if present or -1 otherwise

Assumptions: passnegers.length≥3

1. Let P(n) be our claim.
2. We will always break up an array of length n into n-1 sets of length end-start =1 by using divide and conquer and on the merge up determine the smaller set.
3. **Base Case**: Let n=3. Then there are n-1 =2 "sets" consisting of [0,1] and [1,2]. By comparing the weight of these two sets we can determine which of the four indexes has the terrorist.
    a. **Proof**: Let A= [0,1] and B=[1,2]
        i. Case 1: The terrorist is in position 0.
            1. Set A will have a smaller weight than B which is only possible if the terrorist is in position 0 QED
        ii. Case 2: The terrorist is in position 1.
            1. The algorithm solves this edge case as explained in Algorithm: Solution
        iii. Case 3: The terrorist is in position 2.
            1. Set B will have a smaller weight than A which is only possible if the terrorist is in position 2 QED
        iv. Case 4:
            1. There is no terrorist.

2. The algorithm solves this edge case as explained in Algorithm: Solution

4. **Inductive Case** Assume this algorithm works for n=k
5. **Proof this works for n+1**
    a. There is no intrinsic difference between n+1 and the base case except for the number of steps:
        i. Logic: We recursively go down and break up the input into n-1 sets of 2. Every recursive step back up takes 2 "sets" and compares them in the way explained **3.** Thus, even with n=10000, we still only do compares similar to the base case at a time. We just do more of them.
6. As we have shown, there is no difference between n+1 and the base case n=3, proving that this algorithm holds true for [3,n].

The recurrence relationship for this algorithm is:

$$T(n) = \begin{cases} 1 & end - start = 1 \\ 2T\left(\frac{n}{2}\right) + 1 & end - start > 1 \end{cases}$$

$$T(n) = 2T(n/2) + 1$$

$$2\left(2T\left(\frac{n}{4}\right) + 1\right) + 1 \text{ Plug}$$

$$=\left(4T\left(\frac{n}{4}\right) + 2\right) + 1 \text{ Chug}$$

$$4(2T\left(\frac{n}{8}\right) + 1) + 2 + 1 \text{ Plug}$$

$$= 8T\left(\frac{n}{8}\right) + 4 + 2 + 1 \text{ Chug}$$

$$8\left(2T\left(\frac{n}{16}\right) + +1\right) + 4 + 2 + 1 \text{ Plug}$$

$$= 16T\left(\frac{n}{16}\right) + 8 + 4 + 2 + 1 \text{ Chug}$$

This takes the form of $2^k T_{\left(\frac{n}{2^k}\right)} + 2^{k-1} + 2^{k-2} \ldots 2^1 + 2^0 = \ 2^k T_{\left(\frac{n}{2^k}\right)} + 2^k - 1$. And using these intermediate steps, $\frac{N}{2^k}$=1; N=2^k; $log_2 N = k$;

Therefore: $N(T(1) + 2^k - 1 = N * 1 + 2^{log_2 N} - 1 = N + N - 1 = \boldsymbol{2N - 1 = O(n)}$.

This recurrence relationship fits the algorithm as the algorithm breaks the input into 2 sets of N/2 each recursive step and does O(1) of work on the way back, this is through the compares to determine which of the two sets weighs the least.

My algorithm is straightforwardly transformed into the API supplied by using a variety of private helper methods. These helper methods are called in the constructor and deal with the divide and conquer and logic part of the algorithm; they determine the terrorist's location in O(n) time, afterwards getTerrorist can be called in O(1) time. The doubling ratio for my implementation is ~2.0, exactly what my recurrence relation says it should be.