Judah Goldring
Professor Leff
Spring 2023
Design and Analysis

Detect Terrorist Write-Up

My DetectTerrorist algorithm uses the ideas of dividing and conquering from binary search to find the terrorist. Basically, all that has to be done is split the array consisting of all passengers into two equal parts. Then once split, weigh the two halves against each other (finding the weight in implementation is O(n) cost but for the write-up we're going to call it constant) the half with less weight is the one that contains the terrorist, so recursively call this splitting and weighing method on the half that is lighter. The base case is when only two passengers remain after logN recursive calls. At this point just weight the last two against each other and the index of the lighter half would be the terrorist. Just like binary search this divide and conquer splits an array and throws away the half it isn't using continuously splitting and throwing away as you go down the recursion. The biggest problem with this approach is an a case where the size of the elements to be split is odd, since they would be uncapable of splitting into two equal sized sets to be weighed against each other. If the size of the array was 1 I would just return the value stored at the only index of the array, otherwise, to solve this problem I implemented two protective methods. If the original array size is odd, then I weigh the first and last elements of the array against each other, if they are equal I just pass into my binary search method the zero index and the second to last index (making the array one smaller since removing the last index would have no effect on where the terrorist was). If however, they aren't equal I just return the index of the lighter of the zero and last index. The second way to guarantee equal splits is in the recursive calls, when passing back into the search method, if the side that was lighter was odd, you would just include the next closest index of the larger set for your next recursive call. Making all splits equal.

Pseudo Code:

```
Public DetectTerrorist(passenger array)
    If passenger size is odd:
        If passenger[0] different than  passenger[last]:
            Terrorist =  lighter one
        Else:
            Terrorist = binary search(passenger, 0, last – 1)
    Else:
        Terrorist = binary search(passenger, 0, last)

Public binarySearch(array, first, last)
    //base case
    If first + 1 = last:
        Weigh and return lighter of passenger[first] and passenger[last]
    Find mid point
```

```
Split and weigh left half and right half

If (left half is bigger than right half)
    If(size of right half is odd):
        Return binary search(passenger, mid, right)
    Else:
        Return binary search)passenger, mid + 1, right)

Else if (left half is smaller than right half):
    If (size of left half is odd):
        Return binary search(passenger, 0, mid + 1)
    Else :
        Return binary search)passenger, 0, mid)
```

Recurrence Relation:

$T(N) = T(N/2) + O(1)$

Since my program models binary search at design level, it is the same recurrence relation as binary search. Therefore, closed form is O(logn) at design level.

Since the edge case of having an odd n would only add 1 to the split, meaning at some splits instead of n/2 it would be n/2 + 1 and for any number N there would be very few cases where after adding a number to an odd number making it even it would further divide into odd numbers, making the + 1 negligible and keeping the recurrence equivalent to binary search.

However, since at the implementation level, comparing the weights of two sets isn't constant, but N, the doubling ratio yields a closed form O(nlogn).