Fines Explanation

Samuel Kent 22704037

Given an array priorities[ ] of integers of size N ships.

Initialize an int to hold the length of the given array N. Done in O(1) as we are only accessing and assigning one element.

Initialize a long variable to hold the total amount of fines and assign it 0. Done in O(1).

Set the key position initially as 1 and set the value of priorities[1] as the key in O(1). For all cases, don't set the key to priorities[0] as there are no elements before it in the array.

We then traverse the array backward starting from and including the element directly behind the key position. As we traverse the array backward we compare the value of each element to the key value. If the key value is greater than the current element it is because the current element has failed to give way to the element at key position so we iterate our total amount of fines count.

Once we reach the start of the array after traversing backward we iterate the key elements position forwards and set this as the new key. We then traverse backward again checking each element behind the key position. We continue to do this until we reach the end of the array.

So initially when key = priorities[1], we traverse backwards and only compare key to priorities[0]. Next we set key = priorities[2] and traverse backwards comparing it to priorities[1] then priorities[0]. Then key = priorities[3] and we traverse backward again. We continue this until after our final backward traversal when key = priorities[N - 1].

In this way at the start of the process when we traverse backward we are only examining 1 element costing O(1) time complexity. Next when the key position is iterated we examine 2 elements behind it for O(2). We continue this until key position is the final element and we are comparing key to N-1 or O(N) elements. As such we have done N operations of log N time complexity so all the backward traversals performed cost O(N logN) time.

At the same time we are also traversing the array forward and each time we do we perform a backward traversal, as such we are performing N backward traversals of cost O(N logN).

Therefore, the overall time complexity of this solution is O(N^2) when the Log N is due to it not being the dominant term.