Azoacha Forcheh (20558994)
CS 234 Fall 2017
Assignment 01, Problem 4

Part (a)

```
function has_duplicate(A):
    arr <- merge_sort(A)
    n <- length(A)
    for i <- 0 to n:
        if A[i] == A[i+1]:
            return True
    return False
```

Analysis:
A is first sorted via merge sort, which has $O(n \log n)$ worst-case time.

This sorted array is stored in *arr* (which uses $O(n)$ space as it is length n).

In the worst case, the array has no duplicates. Then, the for loop goes through all n numbers in its range, i.e. iteration occurs n times. Hence, the whole loop has $T(n) = n * 1 \epsilon O(n)$ worst case time since the code within the loop are basic operations and hence only require constant time.

The return and assignment statements require constant time.

Hence, the entire algorithm has worst case time:

$$T(n) = n \, log \, n + n \leq n \log n + n \, log \, n = 2n \, log \, n \, \epsilon \, O(n \log n)$$

Part (b)

```
function has_duplicate(A):
    lst <- [0] * k
    for val in A:
        lst[val] <- lst[val] + 1
        if lst[val] > 1:
            return True
    return False
```

Analysis:
A new integer list of length k is first created - since k is $O(n)$, this uses at most $O(n)$ space. This assignment is a basic operation and takes constant time, but the creation of the list itself takes $O(n)$ time since k is $O(n)$.

As before, in the worst case, the array has no duplicates. Then, the for loop over A goes through every element. The statements within the loop are basic operations and are hence constant. Therefore, in the worst case, the whole for loop runs in $O(n)$ time.

Therefore, the entire algorithm has worst case time:

$$T(n) = n + n = 2n \epsilon O(n)$$