WORKSHOP 12: §6.1-6.3 FEBRUARY 21, 2017

Name: ____

We will discuss this later, but for now you will find the following fact useful. For any $k \in \mathbb{Z}^+$,

$$(k-1)+(k-2)+\cdots+2+1=\frac{k(k-1)}{2}.$$

(1) Consider the following algorithm.

```
FindProduct

Input: $a_1$, $a_2$, ... , $a_n$
       $n$, the length of the sequence.
       $prod$, a target product.
Output: "Yes" if there are two numbers in the sequence whose
       product equals the input $prod$. Otherwise, "No".

For $i$ = 1 to $n$
     For $j$ = $i+1$ to $n$
          If ($a_i \cdot a_j$ = $prod$), Return( "Yes" )
     End-for
End-for

Return( "No" )
```

(a) Characterize the input that will cause the "If" statement in the inner loop to execute the greatest number of times.

(b) Give an asymptotic lower bound for the running time of the algorithm based on your answer to the previous question.

(c) Was it important to use the worst-case input for the lower bound? That is, would any input of a given length have resulted in the same asymptotic lower bound?

(d) Give an upper bound (using O-notation) for the time complexity of the algorithm that matches your asymptotic lower bound for the algorithm.

(2) Look back at your work on Workshop 11, Problem 1.

(a) Find the time complexity of your algorithms.
(If your algorithms are the same I had in mind, you should find that the time complexity for the first algorithm (on sorted inputs) is $\Theta(n)$, and for the second algorithm is $\Theta(n^2)$.)

(b) There is an algorithm to sort an input list called "bubble sort," commonly found in introductory programming classes because of its short code size. Bubble sort has time complexity $\Theta(n^2)$. Use Figure 6.2.2 to give the time complexity of an algorithm that first sorts an input list using bubble sort and then uses your first algorithm (on sorted inputs) for testing for duplicate entries. Is it faster than running your second algorithm directly on the unsorted list?

(c) There are other algorithms that sort an input list, including "merge sort," that have time complexity $\Theta(n \log n)$. Use Figure 6.2.2 to give the time complexity of an algorithm that first sorts an input list using merge sort and then uses your first algorithm (on sorted lists) for testing for duplicate entries. Is it faster than running your second algorithm directly on the unsorted list?