CS:3330 Quiz 2, Spring 2018 (Version 1)

Let us define a problem called BalancedPartition as follows:
BalancedPartition
Input: An array $A[1..n]$ of distinct integers.
Output: A partition of the elements of A into two subarrays, A_1 and A_2 such that (a) all elements of A_1 are less than all elements of A_2 and (b) both A_1 and A_2 contain at least $n/3$ elements.
Here is a randomized algorithm that attempts to solve this problem.

```
function makeBalPart(A[1..n])
    for j ← 1 to 3 do
        p ← random(1, n)
        for i ← 1 to n do
            if A[i] ≤ A[p] then
                A_1.append(A[i])
            else
                A_2.append(A[i])
        if sizeOf(A_1) ≥ n/3 and sizeOf(A_2) ≥ n/3 then
            return (A_1, A_2)
    return Failed
```

1. What is the running time of this algorithm, in Big-Θ notation? You can assume that each integer in the array A is small, i.e., it fits into one word of memory. And you can also assume that "append" and "sizeOf" take $O(1)$ time each. (Show your work for partial credit.)

   Answer: $\Theta(n)$. The outer for loop executes $\Theta(1)$ times and the inner executes $\Theta(n)$ times, independent of the outer loop. All the statements in the code take $\Theta(1)$ time each and so the total running time is $\Theta(n)$.

2. What is the probability that this algorithm will return Failed?

   Answer: $(2/3)^3 = 8/27$. For each execution of the outer-loop, the algorithm picks a pivot uniformly at random from array A. The probability that the chosen pivot leads to an unbalanced partition (A_1, A_2) is $2/3$. For the algorithm to return "Failed" the algorithm has to pick a pivot leading to an unbalanced execution in all three of the executions of the outer loop. Since the choices of the pivot are independent between the three executions of the loop, we get the answer.

3. What small modification can you make to this algorithm to reduce its error probability, without increasing its asymptotic running time (1 sentence)?

   Answer: Increase the number of iterations of the outer loop.

4. If you could not use randomization, how would you solve this problem (in 1 sentence)? And what would the running time of your deterministic (non-randomized) algorithm be?

   Answer: Sort the array using, say the Merge Sort algorithm, in $\Theta(n \log n)$ time and the return $(A[1..n/2], A[n/2+1..n])$. This takes $\Theta(n \log n)$ time.