Analysis of TwoSum Algorithm

1) Characterize Input: Input size is based on how many elements are in the nums vector when the algorithm is called, so n is the size of the nums vector.
2) The Basic Op: The basic operation would be something in the $if\ (map.find(target\ -\ num)\ !=\ map.end())$ line as it will execute on each loop. I will choose the equivalency check "!=" as my basic op.
3) Same Count?: The worst, average, and best cases will be different depending on if there exists two numbers that add to the sum and where they exist in the nums vector. If no sum exists, then, the basic op is executed on each loop, or n times. If the nums that add to the sum are right at the beginning, then the basic op is only executed twice (once for first num, first num added to map, then once more for second num before being terminated). Worst case would be n, average case would be n/2, and best case would be 2.
4) Summation Exp.: A summation expression that characterizes the number of times the basic operation executes would be $\sum_{i=0}^{n-1} 1$.
5) Closed Form: Finding a closed form gives $n - 1 - 0 + 1$, which is simply n.
6) Growth Function: $n \in O(n)$, or linear

The algorithm I choose to use is not a brute-force algorithm. The brute force way to solve this problem is to choose the first number and then compare it against every other number to see if they add up. This results in a n^2 growth. However, with a little insight into the problem, this can be brought down to n. If there is a num 4 and the target is 10, then 10 - 4 must be in the list if 4 and some other num add up to the target. I know I didn't word it well, but essentially all we need to do is keep track of the nums we have encountered. We can then check to see if the target minus the current num is in our list of encountered nums. If it is, then we got a match. Of course, if we keep track of these encountered nums in a plain old vector or similar structure, we have linear search time as we check it for our particular difference. As such, a map with constant access time is used. It doesn't matter what the value is, we just have to check if the difference exists as a key or not, which is constant with the find function of a map.

The analysis of a brute force algorithm differs slightly.

1) Characterize Input: Input size is based on how many elements are in the nums vector when the algorithm is called, so n is the size of the nums vector.
2) The Basic Op: I didn't write a brute force algorithm, but the basic op would be something along of the likes of $if\ A[i] + A[j] == target$. It would be an equivalency check in the nested loops.

3) Same Count?: The worst, average, and best cases will be different. If the numbers that add up to the target are in the front of the vector, then the best would be 1. The worst would be having to check each number against every other number, so $n * n = n^2$. The average would be $n^2/2$.
4) Summation Exp.: A summation expression that characterizes the number of times the basic operation executes would be $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$. I didn't write a brute force algorithm, but ideally one would write it with first loop from $i = 0 \textit{ to } n - 2$ and the second with $j = i + 1 \textit{ to } n - 1$ to prevent checking the same numbers twice.
5) Closed Form: Finding a closed form gives $n(n - 1)/2$ We have done this one in class before, so I didn't show the work.
6) Growth Function: $n(n - 1)/2 \in O(n^2)$, or quadratic