Q6:

Solution:

1) Worst-case input:

A worst-case input of this program is:

- int c: 11

- target: find a^2 + b^2 = 11

More generally, a WC input is

- any number that can’t add up by two squared number (can have two same squared number)

- not including the number that can get an integer after square rooted

- any odd number as it may cause binary search method runs infinitely

[Note: other WC inputs exist]

2) Why this is a WC input and WC asymptotic analysis:

* First, we go through the for loop from line 3 to 7 and got the worst-case running time theta(sqrt(n)) as a\*a <= c can be transformed into a <= sqrt(c) while a++, so the whole for loop executed sqrt(n) times and the worst-case running time is theta(sqrt(n)).
* Second, we go through line 4 and got the worst-case running time is theta (1), because line 4 is in the for loop, so we get theta (1) \* theta(sqrt(n)) = theta(sqrt(n)).
* Third, we go through line 5 and got the worst-case running time is theta (1), because line 4 is in the for loop, so we get theta (1) \* theta(sqrt(n)) = theta(sqrt(n)). (Simply just calculated the if statement and ignored the binary search function)
* Fourth, for binary search function, we have line 2, 5 and 7 that including the if statement and they all have theta (1) worst case running time. However, a binary search function’s worst case running time can’t not be theta (1) too because it’s still in a for loop that executes theta(sqrt(n)) times. Base on the definition, a binary search function with a loop will cause its worst-case running time turns into theta(log(n)). Hence, we get the binary search function’s worst running time in the for loop is theta(sqrt(n)) \* theta(log(n)) = theta(sqrt(n) \* log(n))
* Therefore, adding all the worst-case running time and we get: theta(sqrt(n)) + theta(sqrt(n)) + theta(sqrt(n)) + theta(sqrt(n) \* log(n)) = theta(sqrt(n) \* log(n)).
* The entire code will run theta(sqrt(n) \* log(n)). More precisely, adding theta(sqrt(n) \* log(n)) + theta(sqrt(n) - 1 \* log(n)) + theta(sqrt(n) - 2 \* log(n)) +…+ theta (2 \* log(n)) + theta (1 \* log(n)) = theta(sqrt(n) \* log(n)).

3) Space Analysis:

It’s easy to know that the binary search worst space complexity is O(log(n)), best space complexity is O (1). So, the worst space complexity of the entire code is O(sqrt(n) \* log(n)) while the best space complexity is O(1) \* O(1) = O(1).