Count Distinct Non-Negative Integer Pairs (x, y) that Satisfy the Inequality x*x + y*y < n

Given a positive number n, count all distinct Non-Negative Integer pairs (x, y) that satisfy the inequality x*x + y*y < n.

Examples:

```
Input:  n = 5
Output: 6
The pairs are (0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (0, 2)

Input: n = 6
Output: 8
The pairs are (0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (0, 2),
              (1, 2), (2, 1)
```

A **Simple Solution** is to run two loops. The outer loop goes for all possible values of x (from 0 to √n). The inner loops picks all possible values of y for current value of x (picked by outer loop). Following is C++ implementation of simple solution.

```
#include <iostream>
using namespace std;

// This function counts number of pairs (x, y) that sati
// the inequality x*x + y*y < n.
int countSolutions(int n)
{
   int res = 0;
   for (int x = 0; x*x < n; x++)
      for (int y = 0; x*x + y*y < n; y++)
         res++;
   return res;
}

// Driver program to test above function
int main()
```

```
{
    cout << "Total Number of distinct Non-Negative pairs
         << countSolutions(6) << endl;
    return 0;
}
```

Output:

```
Total Number of distinct Non-Negative pairs is 8
```

An upper bound for time complexity of the above solution is O(n). The outer loop runs √n times. The inner loop runs at most √n times.

Using an **Efficient Solution**, we can find the count in O(√n) time. The idea is to first find the count of all y values corresponding the 0 value of x. Let count of distinct y values be yCount. We can find yCount by running a loop and comparing yCount*yCount with n.
After we have initial yCount, we can one by one increase value of x and find the next value of yCount by reducing yCount.

```
// An efficient C program to find different (x, y) pairs
// satisfy x*x + y*y < n.
#include <iostream>
using namespace std;

// This function counts number of pairs (x, y) that sati
// the inequality x*x + y*y < n.
int countSolutions(int n)
{
   int x = 0, yCount, res = 0;

   // Find the count of different y values for x = 0.
   for (yCount = 0; yCount*yCount < n; yCount++) ;

   // One by one increase value of x, and find yCount fo
   // current x.  If yCount becomes 0, then we have reac
   // maximum possible value of x.
   while (yCount != 0)
   {
       // Add yCount (count of different possible values
       // for current x) to result
       res  +=  yCount;
```

```
        // Increment x
        x++;

        // Update yCount for current x. Keep reducing yCo
        // the inequality is not satisfied.
        while (yCount != 0 && (x*x + (yCount-1)*(yCount-1
          yCount--;
    }

    return res;
}

// Driver program to test above function
int main()
{
    cout << "Total Number of distinct Non-Negative pairs
         << countSolutions(6) << endl;
    return 0;
}
```

Output:

```
Total Number of distinct Non-Negative pairs is 8
```

Time Complexity of the above solution seems more but if we take a closer look, we can see that it is O(√n). In every step inside the inner loop, value of yCount is decremented by 1. The value yCount can decrement at most O(√n) times as yCount is count y values for x = 0. In the outer loop, the value of x is incremented. The value of x can also increment at most O(√n) times as the last x is for yCount equals to 1.