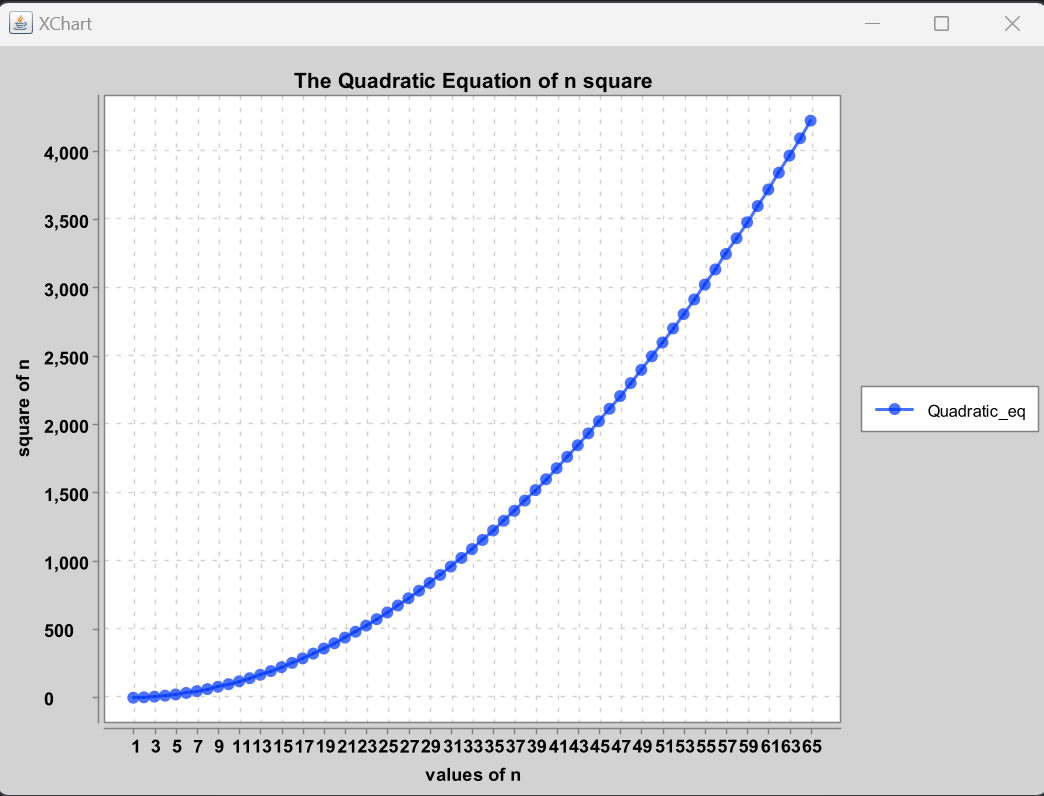
**3.Find polynomials that are upper and lower bounds on your curve from #2. From this specify a big-O, a big-Omega, and what big-theta is.**

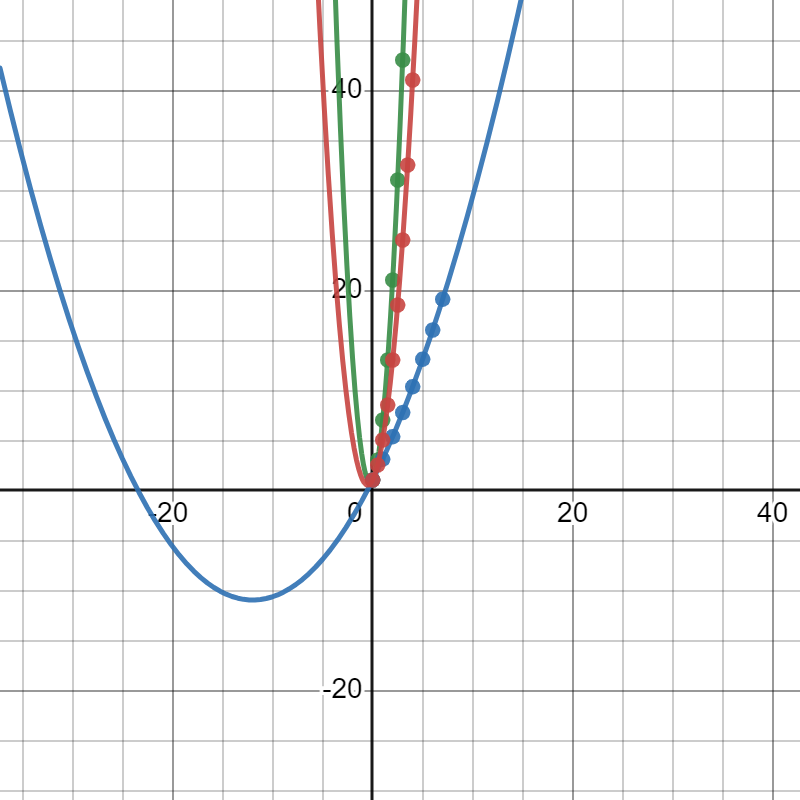


The above graph is a quadratic equation of 2n^2+2n+1

So let the upper bound of the equation be 4n^2+2n+1

And lower bound of the equation be (1/12)n^2+2n+1

Then the graph :



The red color shows the equation of 2n^2+2n+1

The green color shows the equation of 4n^2+2n+1 which is a upper bound

The blue color shows the equation of (1/12)n^2+2n+1 which is the lower bound

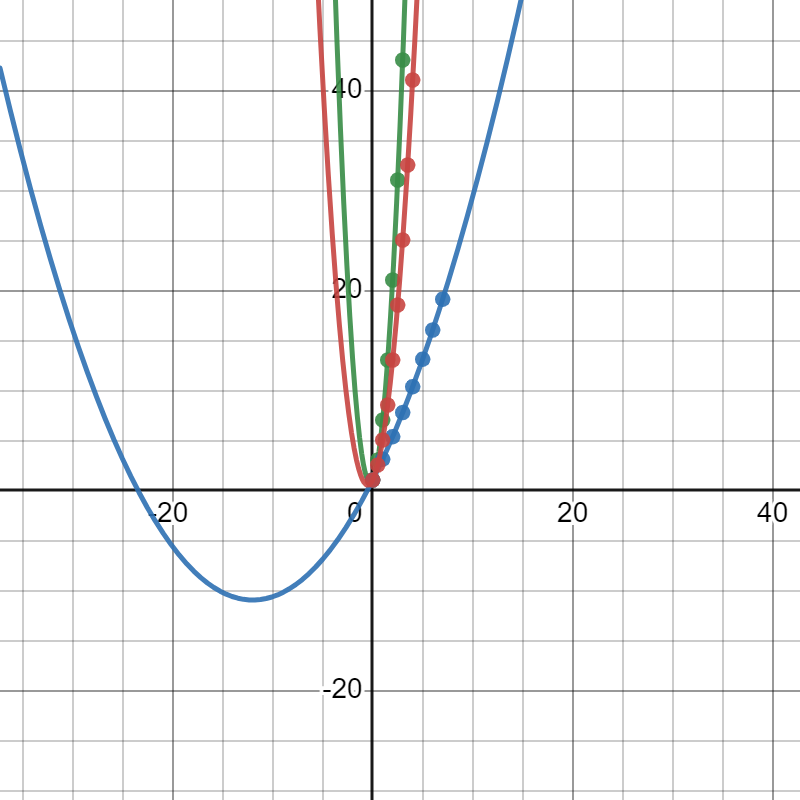
For the quadratic equation

big-O= O(n^2), which is the worst case of the quadratic equation

big-Omega=O(n^2), which is the best case of the quadratic equation

big-theta =O(n^2), ), which is the average case of the quadratic equation.

**4.Find the approximate (eye ball it) location of "n\_0" . Do this by zooming in on your plot and indicating on the plot where n\_0 is and why you picked this value. Hint: I should see data that does not follow the trend of the polynomial you determined in #2.**



The graph meets at the point of 0.5

So n\_0 is 0.5

**5. If I modified the function to be:**

**x = f(n)**

**x = 1;**

**y = 1;**

**for i = 1:n**

**for j = 1:n**

**x = x + 1;**

**y = i + j;**

**Will this increate how long it takes the algorithm to run (e.x. you are timing the function like in #2)?**

Yes as the loop changes the runtime of the algorithm will also change

Before the nested loop the three statements runs one time each

For the first for loop it takes n times

For second for loop it takes n x n times because it is inner loop so, for this loop it is n^2.

Then after we have two statements they are x=x+1and y=i+j these will run n(n+1) times

So as a result the runtime of the given code will be t(n)=O(n^2).

**6. Will it effect your results from #1?**

Yes, it will change according to the values of the quadratic equation.

But for all values of the quadratic equation, the runtime complexity will be the same that is O(n^2).