Here are some exercises for you to work through. They should help you to get a clearer idea (or confirm your understanding of the 'Big O' notation. Remember:

* These are NOT fragments of Java code - They describe the algorithm and a lot of details have been removed.
* .... and don't try to understand them, they sort of make sense but you don't need to understand the algorithm to do the analysis
* The aim is to express how the algorithm changes it's behaviour (how much time it takes) as the amount of data increases

Given the following algorithms, what is their time complexity using the Big O notation?

1.

max = 0  
For i = 0..(n-1)  
   If A[i]>max  
      max=A[i]

O(n) = n

2. Do the same with the following:

numberFound = 0  
For i = 0..(n-1)  
 For j = 0..(n-1)  
 If A[i,j] == target  
 numberFound = numberFound + 1

O(n) = n^2

3. In this example you should assume that the function BinarySearch is O(log n)

numberFound = 0  
For i = 0..(n-1)  
 For j = 0..(n-1)  
 If BinarySearch(A[i,j])  
 numberFound = numberFound + 1

O(n) = N^2 \* logn

4. This example is a little more complex:

max = 0  
For i = 0..(n-1)  
   If A[i]>max  
      max=A[i]  
  
numberFound = 0  
For i = 0..(n-1)  
 For j = 0..(n-1)  
 If A[i,j] == target  
 numberFound = numberFound + 1

O(n) = n + n^2 = n^2

5. .... and what about this one?

numberFound = 0  
For i = 0..(n-1)/2  
 For j = 0..(n-1)/2  
 If BinarySearch(A[i,j])  
 numberFound = numberFound + 1

O(n) = 0.5 \* n^2 \* logn = n^2 \* logn

6. .... and one final one. Assume that randomInt(n) returns a random integer in the range [0..(n-1)] and that A is an array of length n

i = randomInt(n)  
return (a[i])

O(n) = 1