# Comparing & Contrasting Implementations

**The goal for this exercise** is to review recursion, and to think about how one can apply the “Big Oh” notation to not just time consumed, but also space consumed by an algorithm.

Given the code sample below, first figure out what Pow1 & Pow2 will print (and more importantly, *how* they will print). Make sure that you find the running time of the two implementations, in O(N) notation, where N is the **exponent** parameter. Also, see if you can figure out how much space each implementation uses, again using the O(N) notation.

|  |
| --- |
| class Two\_Pow\_Implementations  {  public void RunExercise()  {  Console.WriteLine(" \*\*\*\*\*\* Pow1 Tests: \*\*\*\*\*\*\n");  Console.WriteLine("3^0: {0}", Two\_Pow\_Implementations.Pow1(3, 0));  Console.WriteLine("3^1: {0}", Two\_Pow\_Implementations.Pow1(3, 1));  Console.WriteLine("3^3: {0}", Two\_Pow\_Implementations.Pow1(3, 3));  Console.WriteLine("2^6: {0}", Two\_Pow\_Implementations.Pow1(2, 6));  Console.WriteLine("17^2: {0}", Two\_Pow\_Implementations.Pow1(17, 2));  Console.WriteLine("-2^3: {0}", Two\_Pow\_Implementations.Pow1(-2, 3));  Console.WriteLine(" \*\*\*\*\*\* Pow2 Tests: \*\*\*\*\*\*\n");  Console.WriteLine("3^0: {0}", Two\_Pow\_Implementations.Pow2(3, 0));  Console.WriteLine("3^1: {0}", Two\_Pow\_Implementations.Pow2(3, 1));  Console.WriteLine("3^3: {0}", Two\_Pow\_Implementations.Pow2(3, 3));  Console.WriteLine("2^6: {0}", Two\_Pow\_Implementations.Pow2(2, 6));  Console.WriteLine("17^2: {0}", Two\_Pow\_Implementations.Pow2(17, 2));  Console.WriteLine("-2^3: {0}", Two\_Pow\_Implementations.Pow2(-2, 3));  Console.WriteLine();  }  // This method calculates bas raised to the power of exponent  // (aka bas ^ exponent)  // It does this iteratively  public static double Pow1(double bas, uint exponent)  {  double retVal = 1;  for (int i = 0; i < exponent; i++)  {  retVal \*= bas;  }  return retVal;  }  // This method also calculates bas raised to the power of exponent  // (aka bas ^ exponent)  // It does this recursively  public static double Pow2(double bas, uint exponent)  {  if (exponent == 0)  return 1;  if (exponent == 1)  return bas;  // Assume that IsPowerOfTwo returns true when  // exponent is an even power of 2 – 2, 4, 8, 16, 32, 64, etc  // and false when exponent is anything else  // Assume that it will run in constant time (if you’ve seen  // Big Oh notation – if not, ignore this last line ? )  if ( IsPowerOfTwo(exponent) != true)  return bas \* Pow2(bas, exponent - 1);  else  return Pow2(bas \* bas, exponent / 2);  }  } |

**What you need to do for this exercise:**

1. In the provided starter solution, there should be a project named something like **03\_PCE\_StudentCode** project. In that project, in the file named Student\_Answers.cs, there is a class named Two\_Pow\_Implementations. In a comment inside that class (and at the top of the class – there’s a comment that’s been left there for you to modify), write your answer to the following questions:
   1. Describe what Pow**1** will calculate, and *how* it calculates it.   
      You should be able to describe what this will accomplish in general terms (i.e., feel free to use a specific example or two, but make sure to spell out what the function does generally)
   2. Describe what Pow**2** will calculate, and *how* it calculates it.  
      You should be able to describe what this will accomplish in general terms (i.e., feel free to use a specific example or two, but make sure to spell out what the function does generally)
   3. Clearly list the running time of both Pow1, and Pow2, using the O() notation.
   4. As a ‘stretch’, see if you can figure out how to describe the amount of memory used by the two algorithms, using the “Big Oh” notation. Try approaching this by figuring out how many local variables each version will use, at most. Then try to describe that using the “Big Oh” notation. Make sure to include a quick explanation about why your guess seems right to you.   
      Overall – you should give this last question your best effort, but don’t worry about getting it ‘exactly right’