**Name: Patrick O’Brien**

For each of the below questions, write a short sentence or two to express (in your own words) your answer. Keep the answers short, but use complete, correct, English sentences.

If it helps to clarify the questions, feel free to mentally prefix all the questions with the phrase "According to the video…"

1. After you’ve watched all the videos, please answer this question:  
   Of all the videos that you watched, if you could pick one video to be re-recorded by the instructor outside of class which would you choose? Why?  
   (Keep in mind the recording outside of class will omit any pauses from the instructor answering student questions, have less hemming and hawing, etc, and generally be more concise)

|  |
| --- |
| < Write your answer here > |

**VIDEO: Tracing through recursion**

1. In the video, there are the two things that you're expected to do (in regards to recursion) in this class. What is the first of these things (that this video talks about)?

|  |
| --- |
| Given some recursive code – figure out what it should do. What exactly is it doing, what it should be doing, how to fix it.  Write some recursive code. |

1. What makes a function/method recursive?

|  |
| --- |
| The fact that the code at some point stops and calls itself, which restarts the method with a brand new set of variables. |

1. When a function calls itself, what happens to all the variables in that second call?

|  |
| --- |
| It restarts the function and provides it with a completely new set of variables. |

1. What makes it particularly tricky to keep track of all the variables in all the function calls?

|  |
| --- |
| All the variables have the same name, despite having different values after each recursive call. |

1. On a separate sheet of paper, follow along with the video. You should draw the same picture that is being drawn in the video, and you should completely understand why everything is being drawn.

|  |
| --- |
| x |

1. What is a "stack frame"? What is an "activation record"?

|  |
| --- |
| Stack frames are a block of memory on the stack that keep track of the method calls. An activation record is the same thing as a stack frame with a different name. |

1. In the video, what are the numbers in the circles used to keep track of?

|  |
| --- |
| The position on the stack frame, how many times its been called. |

1. How might you keep track of the recursive function's execution when the recursive function calls itself in multiple, separate places (instead of just calling itself in just one place, as this example demonstrated)?

|  |
| --- |
| Return up to the original call, then go down to the other call, return up, repeat for each separate call of the foo function. |

**VIDEO: Writing recursive code: Basic approach**

1. How is a recursive problem like an onion? Specifically, what are the two things that you reduce the recursive problem to?  
   (Hint: A student in the video gave the *wrong* answer to the question "How is recursion like an onion?" when the student answered "They both make you cry?" ☺ )

|  |
| --- |
| 1. Reducing the problem to a smaller version of the problem, plus some work that you can do in the current iteration of the problem.  2. Once the problem is reduced enough, you can take that final bit of the problem and work it out very quickly. |

1. If you wanted to calculate 23, what could you reduce this problem to?

|  |
| --- |
| 2^3 🡪 (2^2) \* 2 🡪 ((2^1) \* 2) \* 2) = 2^3 🡪 2 return 🡪 4 return 🡪 8 return |

1. Similarly, how could you break 22 down?

|  |
| --- |
| (2^1) \* 2 🡪 2 return   * 4 return * end |

**VIDEO: Writing recursive code: PowR example**

1. What is the point of having both a public and private version of the method? Are both methods recursive?

|  |
| --- |
| The public method is the method you call when you want to do something once.  The private method is recursive and handles all the recursion. The public method is not recursive. |

1. What is the public version of the method responsible for doing (before calling the private version)?

|  |
| --- |
| The public version of the method is responsible for: handling things that you want to do once, calling the private method that handles recursion. |

1. What is the "base case"? How does it differ from the recursive case?

|  |
| --- |
| The base case is: where you don’t do any recursion. Anything that stops the recursion is the base case, otherwise the program will infinitely recurse.  A recursive case differs because it takes the base and makes a recursive check. |

1. If your recursive function does not have a base case, what will happen?

|  |
| --- |
| It will run infinitely with no base case to check it. |

1. In the private method, what are the two base cases?   
   ( Hint: There's one if statement per base case )

|  |
| --- |
| If (b == 0)  Return 0;  If (e == 0)  Return 1; |

1. For this function, what is the C# code the implements the "smaller version of itself" part of the recursion?

|  |
| --- |
| Private int PowR\_priv(int b, int e)  {  If (b == 0)  Return 0;  If (e == 0)  Return 1;  Int smallerProblem = PowR\_priv(b, e -1);  smallerProblem = smallerProblem \* b;  return smallerProblem;  } |

1. For this function, what is the C# code the implements the "some work" part of the recursion?

|  |
| --- |
| smallerProblem = smallerProblem \* b;  return smallerProblem; |

1. What is the C# code that does BOTH the "smaller version of itself" and the "some work" parts of the recursion all on a single line?

|  |
| --- |
| Return PowR\_priv(b, e – 1) \* b; |