**Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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)?

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

1. What makes a function/method recursive?

|  |
| --- |
| < Etc. > |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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)?

|  |
| --- |
|  |

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

|  |
| --- |
|  |

1. Similarly, how could you break 22 down?

|  |
| --- |
|  |

**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?

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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?

|  |
| --- |
|  |