1. Identify some key characteristics of stacks or stack operations.
2. What methods are *required* to implement a stack?
3. What methods might be helpful in managing a stack, but are not required? You might consider methods that could be implemented as a combination of methods identified in the last question.
4. Name an example (or two) of an application for which stacks would be helpful, and explain why. Consider real-life applications, not just classic algorithms.
5. Name an example (or two) of an application for which stacks would **NOT** be helpful, and explain why. Consider real-life applications, not just classic algorithms.
6. Have you used prefix or postfix notation in the past? In what context? Did you find it useful?
7. What are some pros and cons of choosing to implement a stack using an array versus a list implementation?

Answers:

1. A stack is an ordered collection of data. Items can be deleted and added only to the top of the stack. There is no limit on the number of items (theoretically). There is also no technical limitation on the type of items the stack holds. This makes stacks a good LIFO (Last In, First Out) structure. They are also a very simple structure, easy to set up and easy to use.
2. Standard stack operations include Push (insert an item to the top of the stack), Pop (delete and return an item from the top of the stack), and Is_Empty (check if there are any items in the stack).
3. Some methods that may be helpful in managing a stack (I call them "nice-to-have's) include Copy (return the whole stack), Peek (check the value of the top item in a stack) Get_Size (return the length of the stack), and Empty_Stack (clear the contents of the stack).
4. One implementation where stacks could be helpful is in parsing delimiters. You could create a stack which checks if an equation containing parentheses is valid by pushing the open parenthesis to the stack when it is found, and popping the closed parenthesis from the stack when it is found. If the stack has a value not equal to 0 at the end of the expression, or is empty when you try to call a Pop, then the expression is invalid. Else, the expression is valid. This can also be changed to work with mixed delimeters. Stacks can also be used to evaluate a postfix expression, as postfix expressions are best solved left-to-right.
5. Stacks would not be helpful in implementing FIFO (First In, First Out) systems. Once classic example is milk at the grocery store. The milk is loaded into the fridge from the back, and is slid down to the front where the customer can grab it. This is to make sure the grocery stores are not wasting milk by keeping it in the back of the shelf and having it go bad. Stacks are not great for implementing this because they can only pull from the top element (although a stack implementation could easily be changed to support FIFO systems).

Stacks also may not be helpful for search operations (say if you wanted to find a smaller string in a large string). Since you can only look at one item at a time, it would be difficult to implement a stack which could accomplish this task.

6. I have never used prefix/postfix notation in the past, so this is new to me. I can see how it may be helpful. Parsing through expressions with parentheses is tricky because you do not do the operands in the standard left-to-right order. Converting to a prefix/postfix string allows you to evaluate the expressions in order, which could save time if you are working with a long expression. Searching the expression repeatedly for the most nested parenthesis could result in O(n^2) runtime, while converting to prefix/postfix notation would cut the theoretical time to O(n).
7. Using arrays as a stack implementation has the advantage of allowing random access. You are technically not limited to iterating through the whole array (binary search?). But linked lists have dynamic sizing, which can save memory in practical applications, since the top of the stack is rarely reached, and they will not run out of memory. But linked lists have to allocate extra memory space for a pointer, which in some implementations can make them more memory-intensive than arrays. Both require the stack to be homogeneous.