Checkpoint 1

Study the recursive definition below which computes the $n$th Fibonacci number. Draw the complete activation record hierarchy that results from the function call $\text{fib}_a(4)$. This problem is complex to analyze exactly, but find a reasonable upper bound on the number of calls to $\text{fib}_a$ as a function of $n$.

```c
int fib_a(int n) {
    assert (n >= 0);
    if (n == 0) return 1;
    if (n == 1) return 1;
    return fib_a(n-1) + fib_a(n-2);
}
```

To complete this checkpoint: Show a TA your diagram and analysis.

Checkpoint 2

Complete the implementation of the driver function for this alternate recursive definition for Fibonacci:

```c
int fib_b_helper(int n, int count, int prev_fib, int current_fib) {
    if (n == count) return current_fib;
    return fib_b_helper(n, count+1, current_fib, current_fib+prev_fib);
}

int fib_b(int n) {
    assert (n >= 0);
    return fib_b_helper( /* COMPLETE THIS FUNCTION CALL */ );
}
```

Draw the activation record hierarchy for a call to $\text{fib}_b(4)$ and give the order notation for the number of operations. Using this second version as a guide, write Fibonacci iteratively (that is, using a for or while loop instead of recursion).

Note: The differences between the activation record hierarchies of the recursive versions of Fibonacci illustrate the fact that not all recursive functions can be easily re-written as iterative functions.

To complete this checkpoint: Show a TA your diagram, analysis and iterative function.

Checkpoint 3

Checkpoint 3 will be available in lab. To prepare, study the Merge Sort and Nonlinear Word Search code we developed in Lecture 10 which has been posted on the webpage.