Parth Ray

CPE 202

Lab1 Write-Up

3.

callee	calls	returns
search([2,3,5,6,9], 9)	binary_search([2,3,5,6,9], 9, 0, 4)	4
binary_search([2,3,5,6,9], 9, 0, 4)	binary_search([2,3,5,6,9], 9, 3, 4)	4
binary_search([2,3,5,6,9], 9, 3, 4)	binary_search([2,3,5,6,9], 9, 4, 4)	4
binary_search([2,3,5,6,9], 9, 4, 4)	-	4

Once the “search” function is called, it calls its helper function “binary_search” with the two additional input arguments of starting index and ending index. Then this helper function uses recursion to find the target value within the list. It first calculates the middle index using the start and end arguments, which in this case makes the middle index equal to 3. Then the “binary_search” function sees whether the value at the middle index is equal, lower, or greater than the target value. Since 5 (middle value) is less than 9 (target value), it calls “binary_search” again but this time replacing our start argument with the middle index plus one. Again, it runs through the same checks. Since 6 (new middle value) is less than 9 (target value), it calls “binary_search” again and replace our start argument with the middle index plus one. Finally, the function runs through its checks but has now reached the base case, where the middle index value equals our target value, and the “binary_search” function will return the index of 4. This return travels up the calls until it reaches the first call of the “search” function and we get a returned value of 4. The time complexity of binary search is O(log(n)).

4.

callee	calls	returns
fib(3)	fib(2) + fib(1)	2
fib(2)	fib(1) + fib(0)	1
fib(1)	-	1
fib(0)	-	0
fib(1)	-	1

Once fib(3) is called, it checks the base cases and since they are False it will run the recursive case of fib(2) + fib(1). It will recursively call fib(2) first and this gets broken down into fib(1) + fib(0). Now since both are base cases, it will return a value. They return 1 and 0, respectively. Then the 1 and 0 travel back up the stack for fib(2) and let fib(2) return 1 (1 + 0). Now that fib(2) is returned, the function then calls our initial fib(1) call we made when we called fib(3). Since fib(1) is a base case, it returns a value of 1. Finally, the returns of fib(2) and fib(1) are summed up to 2 and 2 is returned for fib(3). The time complexity of the fib function is O(2^n).

5. When you run 1000! using the iterative function, there is no error and you get a return value. However, when you run 1000! using the recursive function, you get a “RecursionError”. This is because when you are using recursion you are saving a piece of memory in your RAM each time a recursion is called for a specific function call. This means that python has already allocated a certain amount of RAM that is available for that function call and when you exceed the amount of memory that was allocated (stack overflow), the program spits back a “RecursionError”. By default, the max number of times a recursive statement can be call is 1000. The error does not happen in the iterative form because the function only need to keep track of one value at time, it does not need to store multiple calls in its memory.