Feedback — Homework 2

Thank you. Your submission for this homework was received.

You submitted this homework on **Fri 26 Feb 2016 7:36 PM PST**. You got a score of **80.00** out of **100.00**. You can attempt again, if you'd like.

"In order to understand recursion, one must first understand recursion." - Anonymous

Question 1

Examine the behavior of this recursive program that models the story of the "Cat in the Hat" by Dr. Suess.

How many calls to the function `clean_up` are made by the program?

You entered:

```
25
```

Your Answer		Score	Explanation
25	✖	0.00	
Total		0.00 / 10.00	

Question 2

Consider the following Python function

```
def add_up(n):
    if n == 0:
        return 0
    else:
        return n + add_up(n - 1)
```

If n is non-negative integer, enter a math expression in n for the value returned by `add_up(n)`.

You entered:

(1+n)*n/2

Preview   Help

Your Answer	Score	Explanation
(1+n)*n/2	✔ 10.00	Correct. `add_up` just computes the sum of the numbers from 0 to n.
Total	10.00 / 10.00	

Question 3

Consider the following Python function

```
def multiply_up(n):
    if n == 0:
        return 1
    else:
        return n * multiply_up(n - 1)
```

If n is non-negative integer, enter a math expression in n for the value returned by

`multiply_up(n)`.

You entered:

n!

Preview   Help

Your Answer	Score	Explanation
n!	✔ 10.00	Correct. This is a recursive implementation of factorial.
Total	10.00 / 10.00	

Question 4

Consider this recursive Python function that computes the Fibonacci numbers below:

```
def fib(num):
    if num == 0:
        return 0
    elif num == 1:
        return 1
    else:
        return fib(num - 1) + fib(num - 2)
```

Let $f(n)$ be the total number of calls to the function `fib` that are computed during the recursive evaluation of `fib(n)`. Which recurrence reflects the number of times that `fib` is called during this evaluation of `fib(n)`?

You may want to add a global counter to the body of `fib` that records the number of calls for small values of n.

Your Answer	Score	Explanation
⊙ $f(0) = 1$ $f(1) = 1$ $f(n) = f(n-1) + f(n-2) + 1$	✔ 10.00	Correct. The number of calls to `fib(n)` is one plus the number of recursive calls to `fib(n - 1)` and `fib(n - 2)`.
○ $f(0) = 1$ $f(1) = 1$ $f(n) = 2f(n-1) + 1$		
○ $f(0) = 0$ $f(1) = 1$ $f(n) = f(n-1) + f(n-2)$		
○ $f(0) = 1$ $f(1) = 1$ $f(n) = f(n-1) + 1$		
Total	10.00 / 10.00	

Question 5

The number of recursive calls to `fib` in the previous problem grows quite quickly. The issue is that `fib` fails to "remember" the values computed during previous recursive calls. One technique

for avoiding this issue is *memoization*, a technique in which the values computed by calls to `fib` are stored in an auxiliary dictionary for later use.

The Python function below uses memoization to compute the Fibonacci numbers efficiently.

```
def memoized_fib(num, memo_dict):
    if num in memo_dict:
        return memo_dict[num]
    else:
        sum1 = memoized_fib(num - 1, memo_dict)
        sum2 = memoized_fib(num - 2, memo_dict)
        memo_dict[num] = sum1 + sum2
        return sum1 + sum2
```

If $n > 0$, how many call to `memoized_fib` are computed during the evaluation of the expression `memoized_fib(n, {0 : 0, 1 : 1})` ? Enter the answer as a math expression in n below.

You may want to add a global counter to the body of `fib` keeps track of the number of calls so that you can track the number of recursive calls.

You entered:

2*n-1

[Preview]   Help

Your Answer		Score	Explanation
2*n-1	✔	10.00	Correct.
Total		10.00 / 10.00	

Question Explanation

Add a global counter to `memoized_fib` to count the number of calls. A simple pattern in n will emerge.

Question 6

In a previous homework, we implemented an iterative method for generating permutations of a set of outcomes. Permutations can also be generated recursively.

Given a list `outcomes` of length n, we can perform the following recursive computation to generate the set of all permutations of length n:

- Compute the set of permutations `rest_permutations` for the list `outcomes[1 :]` of length $n - 1$,
- For each permutation `perm` in `rest_permutations`, insert `outcome[0]` at each possible position of `perm` to create permutations of length n,
- Collect all of these permutations of length n into a set and return that set.

If $p(n)$ is the number of permutations returned by this method, what recurrence below captures the behavior of $p(n)$?

Your Answer		Score	Explanation
⦿ $p(0) = 1$ $p(n) = n\, p(n - 1)$	✔	10.00	Correct.
◯ $p(0) = 1$ $p(n) = 2\, p(n - 1)$			
◯ $p(0) = 0$ $p(n) = p(n - 1) + n$			
◯ $p(0) = 1$ $p(n) = 2\, p(\frac{n}{2}) + n$			
Total		10.00 / 10.00	

Question 7

Using the math notes for recurrences, look up the solution to the recurrence from problem #6.

Enter the solution to this recurrence (as given in the notes) as a math expression in n below.

You entered:

```
n!
```

Preview   Help

Your Answer		Score	Explanation
n!	✔	10.00	Correct. There are $n!$ permutations of length n from a set of n outcomes.
Total		10.00 / 10.00	

Question Explanation

If you miss this problem, be sure to double check your recurrence from problem #6.

Question 8

As part of this week's mini-project, you will implement a function `merge(list1, list2)` that takes two ordered lists and merges the lists into a single ordered list that contains all of the elements in either `list1` and `list2`.

The body of `merge` consists of a `while` loop that iterates until one of the lists `list1` and `list2` is empty. Each iteration of the loop compares the first element in each list, pops the smaller element from its containing list and appends this element to the answer. Once one list is exhausted, the entries in the remaining list are appended to the answer.

If `list1` and `list2` are both of length n, which expression below grows at the same rate as the running time (i.e; the number of Python statements executed) of `merge`?

Your Answer		Score	Explanation
○ n			
○ $n \log(n)$			
○ n^2			
◉ $\log(n)$	✖	0.00	Incorrect. This answer grows too slowly.
Total		0.00 / 10.00	

Question 9

Another part of this week's mini-project will be implementing a recursive sorting algorithm known as `merge_sort`. The basic idea behind `merge_sort` is to split the unordered input list of size n into two unordered sub-lists of approximately size $\frac{n}{2}$, recursively call `merge_sort` to sort each of these sublists and, finally, use `merge` to merge these two sorted sublists.

If $t(n)$ corresponds to the running time of `merge_sort` as a function of the input list size n, which recurrence below captures the behavior of $t(n)$ most accurately?

Your Answer	Score	Explanation
$\bigcirc$ $t(1) = 1$ $t(n) = t(\frac{n}{2}) + n$		
$\odot$ $t(1) = 1$ $t(n) = 2\,t(\frac{n}{2}) + n$	✔ 10.00	Correct.
$\bigcirc$ $t(1) = 1$ $t(n) = t(\frac{n}{2}) + 1$		
$\bigcirc$ $t(1) = 1$ $t(n) = 2\,t(\frac{n}{2}) + 1$		
Total	10.00 / 10.00	

Question 10

Use the math notes for recurrences and look up the approximate solution to the recurrence for `merge_sort` from problem #9.

Select the solution to this recurrence (as given in the notes) as an expression in n below.

Your Answer	Score	Explanation
$\odot$ $n\log(n)$	✔ 10.00	Correct.
$\bigcirc$ n		
$\bigcirc$ n^2		
$\bigcirc$ $2n$		
Total	10.00 / 10.00	