Quiz 3

1. a)
```
def has_sum(total, n, m):
    if total == 0:
        return True
    elif total < 0:
        return False

    else:
        return has_sum(total-n, n, m) or has_sum(total-m, n, m)
```

if total == 0 / return True } base case: if they only need 0 handouts, then of course they can make 0 copies regardless of what n or m is

elif total < 0 / return False } base case: regardless of n, m, it's impossible to print negative copies!

has_sum(total-n, n, m): represents making n copies first

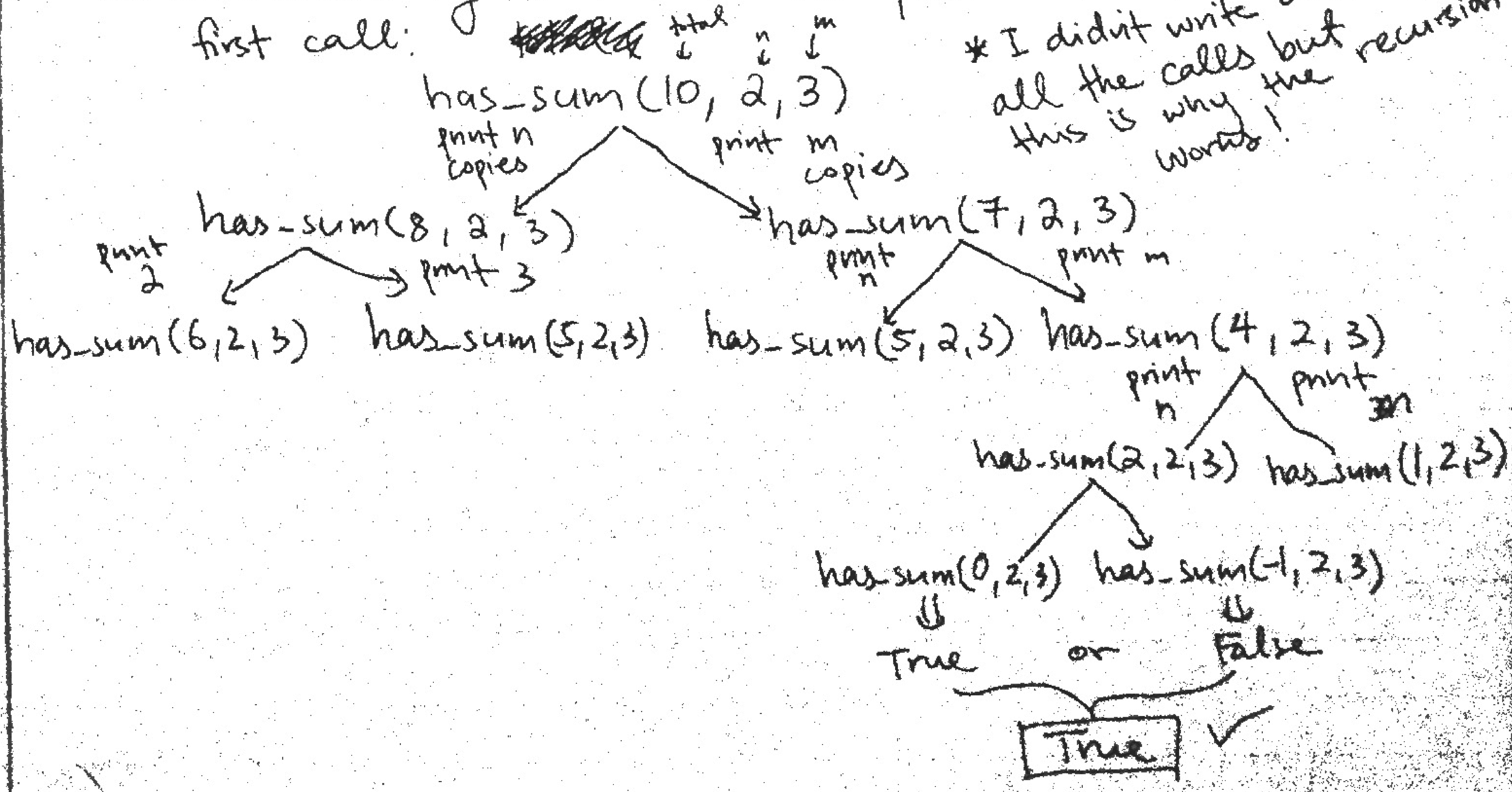
has_sum(total-m, n, m): represents making m copies first

* this problem is conceptually <u>very</u> similar to count_partitions or count_change. it's tweaked so instead of counting the # ways to make total # copies using n and m, we just need to return if it's possible or not

- each recursive call represents a branch of possibilities, as exhibited by this small example:

first call: ~~[illegible]~~

* I didn't write out all the calls but this is why the recursion works!



```
b) def sum_range(lower, upper):
       def copies(pmin, pmax):
           if lower <= pmin and pmax <= upper:
               return True
           elif upper < pmin:
               return False
           else: return copies(pmin+50, pmax+60) or
                        copies(pmin+130, pmax+140)
       return copies(0, 0)
```

we don't want to modify lower & upper, since those shouldn't change throughout the recursive calls

↳ suggests we need a helper function so we can have parameters that we can change

the total if we always printed the min

total if we always print max

if these conditions are met, you have enough copies and also not too many copies!

if the min you can print is larger than the upper bound, you can't satisfy the conditions at all

represents using printer 1

represents using printer 2

we start w/ 0 copies for pmin & pmax since we haven't yet printed anything