Name(s)_____ _____

Algorithms, Assignment 2: Master Method, Recursion Trees, and Quicksort

```
def mergesort3(a):
    # I am using m here as a reminder that this
    # changes at each recursive call
    m = len(a)
    if m <= 1:
        return a

    mid1 = 1 * m // 3
    mid2 = 2 * m // 3

    left   = mergesort3(a[    0:mid1 ])
    middle = mergesort3(a[ mid1:mid2 ])
    right  = mergesort3(a[ mid2:m    ])

    # merge3: T(m) <= 12m + 8
    return merge3(left, middle, right)
```

1. The easy way: use the Master Method to get the asymptotic running time of `mergesort3`.

   (a) What are the values for a, b, and d?

   a = _____ b = _____ d = _____

   (b) Write a recurrence equation for `mergesort3`:

   $T(n) \leq$ _____

   (c) What is the Master Method case (circle the answer)?

   Case 1 Case 2 Case 3

   (d) What is the asymptotic upper running time in Big-O notation?

   $T(n) =$ _____

2. The hard way: perform a recursion tree analysis to get the asymptotic running time of `mergesort3`.

   (a) Draw a picture of the recursion tree for n=27 (sorting 27 items with `mergesort3`).

   (b) As a function of L, how many sub-problems are there at any given level? Here, each sub-problem refers to a call to the `merge3` function.
   Note1: the root is **Level 0**, the second level is **Level 1**, and leaves are at **Level $\log_3(n)$**.

   (c) As a function of L, how many elements are there for a given sub-problem (call to `merge3`) found in level `L`?

   (d) As a function of L, how much work is performed at a given level `L`? This equation will be based on your answers to parts (b) and (c).

(e) What is the **total running time** of `mergesort3`?

(f) What is the **asymptotic upper running time** of `mergesort3`?

(g) Referring to the previous question, prove that your answer to part (e) is upper bounded by your answer to part (f). This is a Big-O proof.

3. What is the asymptotic upper running time of a function with the following recurrence?

$$T(n) = T(n-1) + n$$

Although the master method will not work for this recurrence, you can still perform a recursion tree analysis!

4. Consider the following problem: *You want to compute the longest consecutive number of days in which the stock's value did not decrease.*

For example:

Day	1	2	3	4	5	6	7	8
Value	42	40	45	45	44	43	50	49

In this table, the longest consecutive non-decreasing run is 3. From day 2 through day 4.

A naïve solution to this problem is to look at the longest run starting at each day. This would require a $O(n^2)$ algorithm, as we have n days each with a potential $O(n)$ run.

Although there are iterative algorithms for solving this problem, **your task is to design and describe a divide-and-conquer algorithm**. (Write answers on the next page.)

(a) Provide high-level pseudocode for your algorithm. Only provide pseudocode for the recursive function and not for the "combine/merge" operation (similar to how I did not provide an implementation for `merge3` in problem 1).

(b) Write a recurrence for your algorithm:

$T(n) \leq$ _____

(c) What is the asymptotic upper running time in Big-O notation?

$T(n) =$ _____

5. Consider the following array:

61	11	37	47	59	23	73	83	101	97

(a) For this question only: assume that the array above was returned by a call to partition. Circles all numbers in the **already partitioned** array that could have been the pivot.

(b) What is the probability that the numbers "59" and "23" are compared in a call to quicksort that selects pivots at random?

(c) Is the probability from 3(b) greater than, less than, or equal to the probability that the numbers "61" and "11" are compared?

Greater Than Less Than Equal To

(d) How many comparisons would you expect to be performed by Quicksort if we are lucky and always pick the median element as the pivot? You only need to consider the case when n = 10? You can draw a recursion tree to help you count.

(e) How many comparisons would you expect to be performed by Quicksort if we are unlucky and we always pick the minimum or maximum element as the pivot. Consider the case when n = 10. You can draw a recursion tree to help you count.

6. Consider the following code, where the `roll_single_die` function has an equal chance of returning 1, 2, 3, 4, 5, or 6.

```
total = 0
for _ in range(10):
    die_value = roll_single_die()
    if die_value == 2 or die_value == 3:
        total += 1
return total
```

   a. What is the expected value added to `total` in a **single iteration** of the loop?

   b. What is the final expected value for `total`?

7. After implementing and running your code for Quicksort, fill in the table below.
   i. The first column denotes a test file.
   ii. The remaining columns denote which variant of Quicksort you should use.

   For example, for the cell in the fourth row, third column you would use the **median3** variant of Quicksort and the **ordered-10000.txt** file. You should use the first row to test your code. Keep in mind that the values for the random columns should vary on each run.

   If your code encounters a stack-overflow (`RuntimeError: maximum recursion depth exceeded`), you can try to fix the problem by changing the stack limit, or you can write "SO" in the corresponding cell.

File	first	median3	random	random	random
ordered-10.txt	45	19	26	28	27
ordered-100.txt					
ordered-10000.txt					
randomized-10.txt					
randomized-100.txt					
randomized-10000.txt					

   Write down two observations you can make from the table above.