CMPS 6610 Lab 02

Name (Team Member 1):_____
Name (Team Member 2):_____

You may work on this lab with a partner. We will investigate recurrences for work and span of algorithms.

Tree method (2 pts)

Solve the following recurrences by analyzing the recursion tree as shown in Module 2.1.

a) $W(n) = 3T(n/4) + n^2$.

.
.
.
.
.
.
.
.

b) $W(n) = 2T(n/2) + n/\log n$.

.
.
.
.
.
.
.
.

Brick method (2pts)

Solve the following recurrences using the brick method. First determine whether they are root-dominated, leaf-dominated, or balanced and give adequate explanation. Then, state the resulting asymptotic bound for $W(n)$.

a) $W(n) = 2W(0.49n) + 1.01n$.

.
.
.
.
.
.
.
.

b) $W(n) = W(n/2) + W(n/4) + 0.999n$.

.
.
.
.
.
.

.
.

c) $W(n) = \sqrt{n}W(\sqrt{n}) + \sqrt{n}$.

.
.
.
.
.
.
.
.

Analyzing a recursive, parallel algorithm

You were recently hired by Netflix to work on their movie recommendation algorithm. A key part of the algorithm works by comparing two users' movie ratings to determine how similar the users are. For example, to find users similar to Mary, we start by having Mary rank all her movies. Then, for another user Joe, we look at Joe's rankings and count how often his pairwise rankings disagree with Mary's:

	Beetlejuice	Batman	Jackie Brown	Mr. Mom	Multiplicity
Mary	1	2	3	4	5
Joe	1	**3**	**4**	**2**	5

Here, Joe (somehow) liked *Mr. Mom* more than *Batman* and *Jackie Brown*, so the number of disagreements is 2: (3 <-> 2, 4 <-> 2). More formally, a disagreement occurs for indices (i,j) when (j > i) and (value[j] < value[i]).

When you arrived at Netflix, you were astounded to see that they were using the following algorithm to solve the problem:

```
def num_disagreements_slow(ranks):
    """
    Params:
      ranks...list of ints for a user's move rankings (e.g., Joe in the example above)
    Returns:
      number of pairwise disagreements
    """
    count = 0
    for i, vi in enumerate(ranks):
        for j, vj in enumerate(ranks[i:]):
            if vj < vi:
                count += 1
    return count
```

```
>>> num_disagreements_slow([1,3,4,2,5])
2
```

a) What is the work/span of this method?

.
.
.
.
.

You really don't really like what you're seeing, especially after completing Module 2. Armed with your recently acquired knowledge, you quickly threw together this recursive algorithm that you claim is both more efficient and easier to run on the giant parallel processing cluster Netflix has.

```
def num_disagreements_fast(ranks):
    # base cases
    if len(ranks) <= 1:
        return (0, ranks)
    elif len(ranks) == 2:
        if ranks[1] < ranks[0]:
            return (1, [ranks[1], ranks[0]])  # found a disagreement
        else:
            return (0, ranks)
    # recursion
    else:
        left_disagreements, left_ranks = num_disagreements_fast(ranks[:len(ranks)//2])
        right_disagreements, right_ranks = num_disagreements_fast(ranks[len(ranks)//2:])

        combined_disagreements, combined_ranks = combine(left_ranks, right_ranks)

        return (left_disagreements + right_disagreements + combined_disagreements,
                combined_ranks)

def combine(left_ranks, right_ranks):
    i = j = 0
    result = []
    n_disagreements = 0
    while i < len(left_ranks) and j < len(right_ranks):
        if right_ranks[j] < left_ranks[i]:
            n_disagreements += len(left_ranks[i:])   # found some disagreements
            result.append(right_ranks[j])
            j += 1
        else:
            result.append(left_ranks[i])
            i += 1

    result.extend(left_ranks[i:])
    result.extend(right_ranks[j:])
    print('combine: input=(%s, %s) returns=(%s, %s)' %
          (left_ranks, right_ranks, n_disagreements, result))
    return n_disagreements, result
```

```
>>> num_disagreements_fast([1,3,4,2,5])
combine: input=([4], [2, 5]) returns=(1, [2, 4, 5])
combine: input=([1, 3], [2, 4, 5]) returns=(1, [1, 2, 3, 4, 5])
(2, [1, 2, 3, 4, 5])
```

As so often happens, your boss demands theoretical proof that this will be faster than their existing algorithm. To do so, complete the following:

b) Describe, in your own words, what the `combine` method is doing and what it returns.

.
.
.
.

c) Write the work recurrence for `num_disagreements_fast`.

d) Solve this recurrence using any method you like.

e) Assuming that your recursive calls to `num_disagreements_fast` are done in parallel, write the span recurrence for your algorithm.

f) Solve this recurrence using any method you like.

.
.

g) Now argue how your approach relates to the method used by Netflix.