1: Probability of k Heads

2: Count Number of Parsings

Collaboration with Maks Cegielski-Johnson.

Algorithm: The following is an algorithm for counting the number of ways a string can be parsed, given a language of valid words.

Algorithm 1 Count number of parsings

```
Global Variables
    cache = dictionary of parsings counts for a given word
    language = collection of words in the language
    L = length of the language
end Global Variables

procedure PARSING_COUNT(s)
    if len(s) == 0 then
        return 1
    end if

    if s in cache then
        return cache[s]
    end if

    for index in range(min(N , L) + 1) do
        substring = the first index characters of s
        if substring in language then
            remaining = s with substring removed from the front of s
            count = count + parsing_count(remaining)
        end if
    end for

    cache[w] = count
    return count
end procedure
```

The recursive algorithm will try every possible parse of s, using the words in the language. At each call of the recursive algorithm, s is scanned left to right for matches with words in the language. If a match is found, a substring is formed (removing the match from the beginning of s) and passed to *parsing_count* and the process is repeated on the substring. The scan of s is continued to find additional parses. Scanning continues the minimum of N and L. A parse is successful if the string passed to *parsing_count* is empty. The total count of parses is returned once all possibilities are tested.

Correctness: The algorithm will terminate as the input string s gets smaller with each recursive call to *parsing_count* (moving towards the base case) or no parsing is found and no recursive call is made.

We can observe that we will find every possible parsing as we exhaustively try every possible parsing of s given the language. Thus the algorithm is always correct.

Running time: There are 2^{N-1} possible parsings in the worst-case. This worst-case is when s is a string consisting of all the same letter (ex. aaaaa) and the language is a growing sequence of length N (ex. a, aa, aaa, aaaa, aaaaa).

We combat this exponential runtime by caching the parse counts of the substrings of s. In the worst-case, there are N entries in the language that need to be parsed and cached. The recursion tree with have N levels, and there will be $N - k$ lookups on each level where k is the recursion level.

$$\sum_{i=1}^{n} n - i = \frac{n(n-1)}{2}$$

This is making the assumption that the language is stored in a hash set structure that provides constant time lookup.

$\boxed{O(n^2) \text{ time complexity}}$

3: The Ill-prepared Burglar

(a) Consider the scenario where the ill-prepared burglar can carry 10 pounds and the house has the following items.

Item	Value	Weight
T.V.	8	9
Laptop	5	5
Fine Art	7	5

His strategy of picking the most valuable items first won't be most optimal here. He will end up with the T.V. in his bag with a value of 9. The most optimal strategy would be taking the laptop and fine art, putting his loot value at 12.

(b) Again, the ill-prepared burglar can only carry 10 pounds, this time picking the items with the best ratio.

Item	Value	Weight	Ratio
Finer Art	9	6	1.5
iPad	5	5	1
jewelry	5	5	1

The burglar's new ratio strategy will lead him to pick the finer art, with no room for anything else and a value of 9. The optimal choice would of been to take the iPad and jewelry for a total value of 10.

Algorithm: The following is an algorithm for finding the max total value of items that can fit in a bank locker.

Algorithm 2 Count number of parsings

```
Global Variables
    V = collection of the values for each item
    W = collection of the weights for each item
    cache = the max value of items for a given size
end Global Variables

procedure MAX_VALUE(weight)
    for cur_weight in range(1, weight + 1) do
        for index in range(len(V)) do
            if cur_weight $\geq$ W[index] then
                if V[index] + cache[cur_weight - W[index]] > cache[cur_weight] then
                    cache[cur_weight] = V[index] + cache[cur_weight - W[index]]
                end if
            end if
        end for
    end for

    return cache[weight]
end procedure
```

TODO: explanation of algorithm

Correctness:

Running time:

(d)

4: Central Node in Trees

5: Faster LIS

(a)

(b)

6: Maximizing Happiness

(a) Consider the following setting of children and gifts.

	gift 1	gift 2
child 1	2	1
child 2	2,999	1

If Santa is in a hurry and assigns gifts greedily, child 1 will receive present 1 and child 2 will receive present 2. This will result in a total value of 3. The best assignment that could take place is giving child 2 gift 1 and giving child 1 gift 2. This will result in a total value of 3,000.

(b)