A sequence T with k elements is called subadditive if the elements satisfy the following property: $\frac{T[1]}{1} \geq \frac{T[2]}{2} \geq \frac{T[3]}{3} \geq \cdots \geq \frac{T[k]}{k}$. In this problem, you are given a sequence S of length n over positive integers and want to find the length of the longest possible subsequence of S that is subadditive.

For example, if S is $(1, 2, 4, 8, 10)$, then its longest subadditive subsequence is $(4, 8, 10)$ because $\frac{4}{1} \geq \frac{8}{2} \geq \frac{10}{3}$ and our answer should be the length of this subsequence, 3.

1. Let $L(i)$ denote the length of the longest subadditive subsequence of the suffix $S[i \cdots n]$ of S that starts with the element $S[i]$. Consider the following recursive equations for computing $L(i)$. Here we assume $S[n+1]$ is 0 by default.

   Base Case: $L(n+1) = 0$ (corresponding to the suffix of S of length 0).

   Recursive equation: $L(i) = \max_{j>i:\, \frac{S[i]}{1} \geq \frac{S[j]}{2}} (1 + L(j))$

   The intention is that if the first element of the optimal subsequence of $S[i \cdots n]$ is $S[i]$ and the second element is $S[j]$, then we must ensure $\frac{S[i]}{1} \geq \frac{S[j]}{2}$. The above expression takes the maximum over all j that satisfy this property.

   Unfortunately, this recursive equation is not correct. Find a counterexample where the above equation returns an incorrect answer for $L(i)$.

Question1: CounterExample: S = { 10, 8, 15, 14}
Using the recursive equation in the example algorithm:
(S[1]/1) = 10 >= (S[2]/2) = 4;
(S[2] /1) = 8 >= (S[3]/2) =7.5;
(S[3] /1) = 15 >= (S[4] /2) = 7;

HOWEVER, the proposition of S[1]/1 >= S[2]/2 >= S[3]/3 >= S[4]/4 is false since S[2]/2 <= S[3]/3. Therefore the example algorithm is problematic.

Question2:

Assume we have an array S[ ] which contains a series of numbers.

Let SubA[ ] denote an subarray that is subadditive, and let L(i) denote the length of the longest subadditive sequence in the SubA[ ] array.

If there exists a number x such that x belongs to the S[ ] and doesn't belong to SubA[ ], and if x satisfies the below condition:

If SubA[last_index] / SubA.size() >= x / SubA.size() +1, then SubA[ ].append x. Recurse on the updated SubA[ ] array with a new size of (SubA.size + 1). Let curSize be the current size of the SubA[ ] subarray, and (i + curSize) is the last index of the subarray. Then in this manner we can have a recursive equation that:

L(i) = $MAX_{j>i+1} \frac{S[i+curSize]}{curSize}$ >= $\frac{S[j]}{curSize+1}$ (1 + L(i))

Question3:

Algo design:

Initiate S[ ] // the given array with a series of numbers
Initiate SubA[ ] // the subarray with longest subadditive sequence

```
SubAFinder(S[ ], SubA[ ]) {

For (i =0; i< SubA.size; i++) {
//initialize all the elements in SubA to 1 so we can update the number when we find a
//subadditive sequence
SubA[ i ] = 1;
}
For ( i=0; i< S[ ].size -1; i++) {
  For (j=0; j< i -1; j++) {
    If S[ j ] / SubA[ j ]  >= S[ i ] / SubA[ j ]+1 AND SubA[ j ] + 1 >= SubA [ i ] {
        SubA[ i ] = SubA [ i ] + 1;
          Find the max element in the SubA array → max
Return max;
}
}
```

Running Time analysis:

Initializing SubA[ ] elements to 1 → O(n)
2 for loops → O(n^2)
Finding the max elements in the SubA[ ] array → O(n)

T(n) = O(n) + O(n^2) + O(n) → O(n^2) → poly(n)

Proof:
Base case:
When The S[ ] array only contains a single element → SubA[ ] = { 1 } → this is true since no comparison is needed.

Inductive hypothesis:
Assume that when the S[ ] array contains n = k elements, the algorithm would correctly find the longest subadditive array.

Inductive step:
When n = k + 1, let a = the k+1th element in the array S[ ]

Let SubA[i] be the length of the subadditive sequence at index i, and according to the inductive hypothesis, we already have the length of subadditive sequences at each index of array for a size n = k.
If SubA[length - 1] / SubA.length >= S[k+1] / SubA[i] +1, then we can increment the SubA[i] by 1, updating the corresponding index value, in that manner, we will find the max element of the SubA array with a length of n = k + 1 eventually.