1. [10 marks] Consider writing a positive integer \( n \) as a sum of numbers of the form \( t^3 \), where each \( t \) is a (possibly negative) integer. For example, \( 46 = 4^3 + (-3)^3 + 2^3 + 1^3 \).

Here is a greedy algorithm for doing that:

``` pseudocode 
gex(n)
 t := \lfloor \sqrt[3]{n} \rfloor;
 if \( n = t^3 \) then
 return(\( t \))
 else if \( (n-t^3) \leq (t+1)^3 - n \) then
 return(\( t, gex(n-t^3) \))
 else
 return(\( (t+1), -gex((t+1)^3 - n) \));
```

Here the comma denotes concatenation, and the minus sign applied to a list negates all the elements of the list. So \( gex(46) \) returns the list \([4, -3, 2, 1]\).

(a) Prove that, in the unit cost model, \( gex(n) \) produces a representation for \( n \) consisting of \( O(\log \log n) \) terms, in \( O(\log \log n) \) time. Note that here we assume \( \lfloor \sqrt[3]{n} \rfloor \) can be computed in unit time.

(b) Does \( gex \) always return the smallest possible number of terms to represent \( n \) as a sum of cubes? If so, prove it. If not, find a counterexample.
2. [10 marks] Joe works for a bakery and needs to make \(n\) cakes for customers to pick up. Each cake \(i\) must be completed by a deadline \(d_i\), and requires a length \(L\) (independent of \(i\)) that is the amount of time Joe needs to make it continuously before finishing it. The work of a cake cannot be interrupted in the middle, and Joe cannot work on more than one cake simultaneously. Moreover, to ensure the cake is fresh when the customer picks it up, Joe can only start working on a cake no earlier than \(d_i - L'\) for some \(L' > L\) (again, \(L'\) is independent of \(i\)). During busy seasons Joe cannot satisfy all of the customer orders, and will have to select a subset of the orders to work on. Design a greedy algorithm to maximize the number of cakes Joe can finish before the deadlines.

The input consists of the two positive integers \(L' > L\), and the list of deadlines \(d_i\) for \(1 \leq i \leq n\). The output of the algorithm is the actual start time \(t_i > 0\) for each cake \(i\) that Joe will finish on time. All \(d_i\) and \(t_i\) are positive integers. Each cake \(i\) in the output should satisfy \(d_i - L' \leq t_i \leq d_i - L\). No two intervals \([t_i, t_i + L]\) and \([t_j, t_j + L]\) in the output can overlap. The goal is to maximize the number of cakes in the output.

3. [10 marks] (a) The bakery got too busy and hired another baker Jean to help take more customer orders. Two bakers can work on two different cakes independently in parallel. However, each cake still requires a single baker to continuously work for \(L\) time. All other conditions remain the same as question 2. Design a greedy algorithm to maximize the total number of cakes the two bakers can finish on time.

(b) Suppose Jean is slower than Joe, and can only finish a cake in \(L'\) time. Here \(L'\) is defined as in (a), and every other condition remains the same as (a). Construct a counterexample to show the following algorithm may not produce the optimal solution: Whenever a baker becomes available, assign him/her a cake with the earliest deadline that he/she can finish on time, and let him/her start working on it as early as possible. If both bakers become available at the same time, assign a cake to Joe first.

4. [10 bonus marks] Design a polynomial-time dynamic programming algorithm to find the optimal solution for question 3(b). You only need to compute the maximum number of cakes. No need to compute the actual scheduling.

To ease your work, in this question we only require your algorithm to run in polynomial time (with respect to the input size). No need to strive for the most efficient algorithm. In your answer, you need to provide the definition of the subproblems, the construction of the recurrence relation, the way to compute the final output, the justification that the recurrence relation is correct (which can be combined with the recurrence relation construction), and a brief justification that the algorithm runs in polynomial time. The pseudocode and discussion of base case are not required.

(Hint: Notice each cake \(i\) made by Jean must start precisely at time \(d_i - L'\) and finish at \(d_i\).)