1. Given two sorted lists, $L_1$ and $L_2$, write a procedure (algorithm) to compute $L_1 \cap L_2$ using only the basic list operations.

2. Given two sorted lists, $L_1$ and $L_2$, write a procedure to compute $L_1 \cup L_2$ using only the basic list operations.

3. Polynomials are important for many areas of mathematics and many of the programs such as Octave, Mathematica and Maple that allow you to do polynomial arithmetic have to be carefully designed.

   One way of representing a polynomial would be to store the coefficient of each term in an array, where the *order* of that term would be used as index. So the polynomial $-3 \cdot x^2 + 3 \cdot x - 17$ would be stored in an array of size 3 with

   | | |
   |---|---|
   | 0 | -17 |
   | 1 | 3 |
   | 2 | -3 |

   The trouble with this approach is that an innocuous looking polynomial like $2 \cdot x^{10000} + 1$ requires a massive amount of space. A linked list approach is much more space efficient: with this approach we keep track of the terms of the polynomial that have non-zero coefficient with a pair of numbers that represent the term's coefficient and the order. So the polynomial above would be stored by a two-element linked list where the first element would be $(2, 10000)$ and the second would be $(1, 0)$.

   (a) Write a function to *add* two polynomials, of term count $m$ and $n$. Do not destroy the input. Use a linked list implementation. If the polynomials have $m$ and $n$ terms, respectively, what is the time complexity of your program? (You may assume that the two input linked lists are ordered according to the *exponent* of the terms, from largest to smallest, as in the above example.)

   (b) Write a function to *multiply* two polynomials using a linked list implementation. Make sure that the output polynomial is sorted by exponent and has at most one term for any exponent.

      i. Give an algorithm to solve this problem in $O(m^2n^2)$