Problem. 4.1: Read the assigned readings and practice source codes under directories CODE/9-12.

Problem. 4.2: Prove the following statements rigorously.
(a) $O(c \cdot g(n)) = O(g(n))$ for a finite constant $c > 0$.
(b) $O(f(n)g(n)) = O(f(n))O(g(n))$.
(c) $O(f(n)) + O(g(n)) = O(g(n))$ if $f(n) = O(g(n))$.
(d) $O(n \log n) = O(n^{3/2})$.
(e) $\Theta$ is transitive.
To prove (a)-(d), pick an arbitrary representation function from the set in the left hand side, and show that it belongs to the class on the right hand side.

Problem. 4.3: Given two vectors $u, v \in \mathbb{R}^n$, their inner product $u^Tv = \sum_{i=1}^{n} u_i v_i$ takes $\Theta(n)$ operations to compute. However, when $u$ and/or $v$ are sparse (which means that many of the elements of $u$ and $v$ are simply 0), time complexity can be reduced by using linked lists.
(a) Create a structure in C such that each node of the linked list contains three variables: `int index`, `double value` and `link next`, where `link` is a pointer that points to the node type. The variable `index` is used to store the corresponding index of the variable `value` in the vector that is not zero.
(b) Create C functions `void addNode(link head, int index, double value)`, `void printList(link p)`, `void freeList(link p)` for the linked list. The function `addNode` will create a new node with initial values `(index, value)` and insert it right before the first node pointed by the pointer `head`.
(c) Create a C function `double innerProduct(link p, link q)` that returns the inner product between the two lists. What is the time complexity for this operation if on average there are only $m \ll n$ non-zero elements in each vector?
(d) Create a C program that creates the two linked list $u$ and $v$ based on data provided from the files "hw4u.txt" and "hw4v.txt". Test your program to verify whether it returns the correct inner product $u^Tv$. You may use your own function `printList` to debug your code.

Note: Each line of the file `hw4u.txt` or `hw4v.txt` contains two values: an integer `index` followed by a real number `value`. The `index` starts from 1 instead of 0 and the lines are sorted based on indices already.