HW 3 Derek Lin

1. a. Vector of vectors.
   Outer vector index represents the subject number.
   Inner vector index represents the experiments the subject has been part of.
   b. Map with key = grade value, and then value = vector of strings of student names of those students that earned that grade. (Or a student identifier string)
   OR a simple vector-implemented list with just the names of the students that got an A. Depends on the complexity of the use case.
   c. Map with key = student name, and then a Student struct containing student info like grades, seat number, academic standing.
   d. Same as b.

2. The function will reverse the LinkedList. funcA will go through the linked list and get to the last node. funcB will take the linked list in its given order, and recursively call itself, each time passing in the next node. When it takes in the last node, it returns the last node as `in`. One step before the recursive call, the previous node's `next` is set to `in`, which is now the last node and the one most recently returned. This is performed until the second node's `next` is set to the first node. In funcA, the value of the `first -> next` is set to null. Finally, funcA starts to callback, which the pointer to the last node of the original LL, and the head of the new, reversed LL.

3. a.
   Θ(n^3). There are 3 loops. It is important to note that the if statement becomes inconsequential for large values of i, j, k, and simply becomes some value n. Thus, we will just chalk it up to n. Because the outermost loop runs an inner loop n times, and that inner loop runs an innermost loop n times, and that innermost loop runs an assignment operation(considered the most significant operation at large values of n) n times, the runtime for this code is Θ(n^3) → Θ(n^2)

   b.
   The function func(int x) is a recursive function. The line `if (x <= 1) return;`, is our base case and has runtime of Θ(1).
   And the recursive call has runtime of f(n - 1). Because it's reducing the n by one for each call, it's f(n-1). So, our runtime for func() is f(n) = Θ(1) + f(n - 1). Because the function is going to be running n - 1 times, the runtime of f(n) is Θ(n). The runtime for the for loop can be expressed as summation of 2^k from k = 0 to log (n - 1). Since i is incrementing by `i *= 2`, the loop is going to loop through log (n - 1) times instead of regular n - 1 if it's

incrementing by ++i. The runtime for this loop is Θ(n). So the total runtime is the runtime of loop + runtime of function, Θ(n) + Θ(n) = 2 * Θ(n) => Θ(n)

c.

The while loop is executed n/i times. So the j loop is executed n times. And the summation of n^2 from I=0 to I = n. Thus, it is theta of n squared log n.

d.

The outer for loop of i < n with a[i] = sqrt(i) has a time complexity of Θ(n). The code consists of two summations: summation of i = 1 to i = log3(n) of constant c, summation of i = 1 to i = log3(n) of 3^i. This equals to Θ(log(n)) and Θ(n). thus the overall runtime is Θ(n) + Θ(log(n)) + Θ(n) => Θ(n).

In other words, you're adding the geometric series from i = 1 to i = log base 3 n.