**Piazza questions and answers from previous terms**

 list? vs. pair?

Recall that a pair is simply a container for two values; the simplest way to make one is to apply cons.

A list is a linked list of pairs.  Each pair except the last one is a reference to the next pair in the list; the cdr of the last pair must be null, otherwise the list is improper.

pair? is a constant-time procedure that simply asks, "is this value a reference to a pair?"

list? is a linear-time operation that asks, "is this value either null or a reference to the first pair of a proper list?

So efficiency is one basis to choose between the two tests.

I hope that the following transcript will help you better understand these procedures.

> (list? '())

#t

> (pair? '())

#f

> (list? '(a b c))

#t

> (pair? '(a b c))

#t

> (list? '(a b . c))

#f

> (pair? '(a b .c))

#t

> (pair? 'a)

#f

> (list? 'a)

#f

# duplication in lists

Can you ever do better than O(n^2) for determining the duplication of numbers in a list?

A novel approach might be for each index compare this value to all other index values.

Another way is to pre-process with a sorting algorithm with Big O lower than O(n^2). Then compare each value next to each other. This is O(n) + complexity of sort.

However sorting isn't an easy option in this case since we can't mutate anything yet.

So are we left to O(n^2) efficiency for now until we learn more about manipulation in scheme?

**the instructors' answer,**

*where instructors collectively construct a single answer*

[Actions](https://piazza.com/class/jl863803n0a6tl?cid=14)

If you don't use any auxiliary data structure, N^2 is the best you can do.  If you copy the numbers into a binary heap or balanced tree you can do O(N log N) worst case.  With a hash table you can have O(N) expected value, but O(N^2) worst case.

Finally, I don’t think Scheme’s sort procedure mutates anything.

# Operations with lists

I had no problem using - and + between a number and a list, but I do if I use it with two lists, even if there is only one element on them. Are there special operators for adding and subtracting the elements in a list? Or is it not possible to operate with them being in a list?

Thank you

[hw1](https://piazza.com/class/jl863803n0a6tl?cid=19)

This private post is only visible to Instructors and [2 others](https://piazza.com/class/jl863803n0a6tl?cid=19)

[**edit**](https://piazza.com/class/jl863803n0a6tl?cid=19)·[good question](https://piazza.com/class/jl863803n0a6tl?cid=19)0

Updated 2 months ago by

Ana Huerta

**the instructors' answer,**

*where instructors collectively construct a single answer*

[Actions](https://piazza.com/class/jl863803n0a6tl?cid=19)

Depending on what you are trying to do, **apply** may be what you need. Or (for things like A2-A5) you can use a loop or (for A1 where all lists have fixed length) you can just write two or three - or + expressions.

# Comparing numeric quantities

If x and y are known to be numbers, use (= x y) instead of (equal? x y) to compare them.

# Assignment 2

This is an individual assignment. You can talk to anyone and get as much help as you need, but you should type in the code and do the debugging process, as well as the submission process. You should never give or receive code for the individual assignments

*\*Restriction on Mutation continues.\** One of the main goals of the first few assignments is to introduce you to the functional style of programming, in which the values of variables are never modified. Until further notice, you may not use set! or any other built-in procedure whose name ends in an exclamation point. It will be best to not use any exclamation points at all in your code. You will receive zero credit for a problem if any procedure that you write for that problem uses mutation or calls a procedure that mutates something.

Assume valid inputs. As in assignment 1, you do not have to check for illegal arguments to your procedures. Note that in the set? problem, any Scheme list is a valid input.

## Q1 (5 points)

(a) Write the procedure (fact n) which takes a non-negative integer n and returns n factorial. You can just copy this procedure from Assignment 0, and call it from your choose procedure from part (b).

fact: NonNegativeInteger -> Integer

Examples:

(fact 0) -> 1

(fact 1) -> 1

(fact 5) -> 120

(b) (5) Write the procedure (choose n k) which returns the number of different subsets of k items that can be chosen from a set of n distinct items. This is also known as the binomial coefficient. If you’ve forgotten the formula for this, a Google search for “Binomial Coefficient” should be helpful.

choose: NonNegativeInteger x NonNegativeInteger -> NonNegativeInteger (examples on next page)

Examples:

(choose 0 0) -> 1

(choose 5 1) -> 5

(choose 10 5) -> 252

## Q2 (5 points)

Write a procedure (sum-of-squares lon) that takes a (single-level) list of numbers, lon, and returns the sum of the squares of the numbers in lon.

sum-of-squares: Listof(Number) -> Number

Examples:

(sum-of-squares '(1 3 5 7)) -> 84

(sum-of-squares '()) -> 0

## Q3 (8 points)

Write the procedure (range m n) that returns the ordered list of integers starting at the integer m and increasing by one until just before the integer n is reached (do not include n in the resulting list). This is similar to Python's range function. If n is less than or equal to m, range returns the empty list.

range: Integer x Integer -> Listof(Integer)

Examples:

(range 5 10) -> (5 6 7 8 9)

(range 5 6) -> (5)

(range 5 5) -> ( )

(range 5 3) -> ( )

## Q4 (10 points)

In mathematics, we informally define a set to be a collection of items with no duplicates. In Scheme, we could represent a set by a (single-level) list. We say that a list is a set if and only if it contains no duplicates. We say that two objects o1 and o2 are duplicates if (equal? o1 o2). Write the predicate (set? list), that takes any list as an argument and determines whether it is a set.

set? : list -> Boolean

Examples:

(set? '()) -> #t ; empty set

(set? '(1 (2 3) (3 2) 5)) -> #t ; (2 3) and (3 2) are not equal?

(set? '(r o s e - h u l m a n)) -> #t

(set? '(c o m p u t e r s c i e n c e)) -> #f ; duplicates

## Q5 (5 points)

The union of two sets is the set of all items that occur in either or both sets (the order of the items in the list does not matter).

union: Set x Set -> Set

Examples:

(union '(a f e h t b) '(g c e a b)) -> (a f e h t b g c) ; (or some permutation of it)

(union '(2 3 4) '(1 a b 2)) -> (2 3 4 1 a b) ; (or some permutation of it)

## Q6 (5 points)

Write a procedure (more-positives? lon) that return true if the list has more positive numbers than non-positive numbers.

hint: you might need to write a recursive helper function with a

different return type or different parameters. This one is

straightforward to solve with tail recursion, but we won't require

that.

more-positives: Listof(Integer) -> Boolean

Examples:

(more-positives '(1 2 -1)) -> #t

(more-positives '(10 -2 -1)) -> #f

(more-positives '(10 0)) -> #f

## Q7 (10 points)

Write a procedure (nearest-pair lon) that returns a scheme Pair of the two integers from the list that are numerically closet to each other, in sorted order. You can assume that that there is one unique closest pair (i.e. dont have to consider when there are two pairs with the same smallest distance). You can assume the list has 2 elements at least.

hint: it might be handy to search racket's built in help for "sort" as part of solving this problem

nearest-pair: ListOf(Integer) -> Pair(Integer)

Examples:

(nearest-pair '(10 20 22 30 40)) -> (20 . 22)

(nearest-pair '(7 101 20 100 0)) -> (100 . 101)