**PA5 Recursion Practice**

**Problem 1-3: any language (Python or Lisp)**

1. Write the recursive function AL1 which takes two lists of integers and returns a list of integers in which each element is the sum of the corresponding elements of the incoming lists. For example,

(AL1 ’(3 1 4) ’(8 2 5)) -> (11 3 9)

The lists need not be equal in length; the shorter list should be treated as though it were padded out with zeros at the end. For example,

(AL1 ’(3 1 4) ’(2 2)) -> (5 3 4)

(AL1 ’(3 1 4) nil)-> (3 1 4)

1. Write the recursive function INTERSECT which takes two sets of integers and returns the intersection of the two sets. By set, I mean a list which contains no duplicates. For example,

(intersect ’(4 1 2) ’(2 5 3 1 7)) -> (1 2)

(intersect ’(4 1 2) nil) -> nil

1. We have seen that an iteration can be written as a tail recursion (actually every iteration can be written as a tail recursion – believe/agree?) Use tail recursion to implement a function NONE (that’s N-O-N-E) which takes a list of integers and returns a 2-element list: the first element is the number of odd elements and the second element is the number of even elements in the argument. For example,

(none nil) -> (0 0)

(none ’(3)) -> (1 0)

(none ’(2)) -> (0 1)

(none ’(3 1 4 1 5 6)) -> (4 2)

1. Write a Lisp function op-some which uses functional arguments to implement the op-some schema. The form of a call to op-some is this:

(op-some condition operation arg).

For example, we might use op-some to implement the function sq-odd, which takes a list of integers and returns another list in which only the odd numbers have been squared:

(op-some #’oddp #’sq nil) -> nil

(op-some #’oddp #’sq ’(2)) -> (2)

(op-some #’oddp #’sq ’(3)) -> (9)

(op-some #’oddp #’sq ’(3 6 4 5 2)) -> (9 6 4 25 2)

Of course, we will have to write the function sq which takes a number and returns its square.

1. Use Python to solve perform the same task as specified in 4.