a)  
(struct problem (list 0 0 1 1 2 3 3 3 4 4 5 5 7 7 13) 4)

-》

（cons 0 (insert-sorted (list 0 1 1 2 3 3 3 4 4 5 5 7 7 13) 4)）

（cons 0 (cons 0 (insert-sorted (list 1 1 2 3 3 3 4 4 5 5 7 7 13) 4))）

（cons 0 (cons 0 (cons 1 (insert-sorted (list 1 2 3 3 3 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (insert-sorted (list 2 3 3 3 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (insert-sorted (list 3 3 3 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3(insert-sorted (list 3 3 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3 (cons 3(insert-sorted (list 3 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3 (cons 3 (cons 3 (insert-sorted (list 4 4 5 5 7 7 13) 4)))）

（cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3 (cons 3 (cons 3 (cons 4 (list 4 4 5 5 7 7 13)))))

(list 0 0 1 1 2 3 3 3 4 4 4 5 5 7 7 13)

->9

b)

(struct problem (list 0 0 0 0 0 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (insert-sorted (list 0 0 0 0 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (insert-sorted (list 0 0 0 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0(insert-sorted (list 0 0 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0(insert-sorted (list 0 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0(insert-sorted (list 1 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1(insert-sorted (list 1 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1 (cons 1(insert-sorted (list 2 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1 (cons 1 (cons 2(insert-sorted (list 3 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3(insert-sorted (list 5 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3 (cons 5 (insert-sorted (list 5 7 7 9 13) 7)

(cons 0 (cons 0 (cons 0 (cons 0 (cons 0 (cons 1 (cons 1 (cons 2 (cons 3 (cons 5 (cons 5 (insert-sorted (list 7 7 9 13) 7)

(list 0 0 0 0 0 1 1 2 3 5 5 7 7 7 9 13)

->12

c)

(struct problem (list 0 0 0 1 1 1 3 4 6 6 6 8 8 10 10) 0)

(list 0 0 0 0 1 1 1 3 4 6 6 6 8 8 10 10)

->0

;;; (make-random-list n) -> list?

;;; n: integer?, non-negative

;;; Creates a list of size of n made of random

;;; numbers in the range 0 to n.

(define make-random-list

(lambda (n)

(if (zero? n)

null

(cons (random n) (make-random-list (- n 1))))))

(make-random-list 3)

;;; --->(make-random-list 3)

;;; ---> (if (zero? 3)

; null

; (cons (random 3) (make-random-list (- 3 1)))

;;; ---> (if #f

; null

; (cons 3 (make-random-list 2))

;;; --> (cons 3

; (if (zero? 2)

; null

; (cons (random 2) (make-random-list (- 2 1))))

;;; --> (cons 3

; (if #f

; null

; (cons 0 (make-random-list 1)))

;;; --> (cons 3 (cons 0 (make-random-list 1)))

;;; --> (cons 3 (cons 0

; (if (zero? 1)

; null

; (cons (random 1) (make-random-list (- 1 1))))

;;; --> (cons 3 (cons 0

; (if #f

; null

; (cons 1 (make-random-list 0 )))

;;; --> (cons 3 (cons 0 (cons 1 (make-random-list 0 )))

;;; --> (cons 3 (cons 0 (cons 1

; (if (zero? 0)

; null

; (cons (random n) (make-random-list (- n 1))))

;;; --> (cons 3 (cons 0 (cons 1 null))

;;; --> (list 3 0 1)

;;; (insert-sorted l v) -> list?

;;; l: list?, sorted

;;; v: any

;;; Returns l but with v inserted in its sorted position in l.

(define insert-sorted

(lambda (l v)

(match l

[null (list v)]

[(cons head tail)

(if (<= v head)

(cons v l)

(cons head (insert-sorted tail v)))])))

;;; --> (insert-sorted (list 0 1 2) 9)

;;; --> (match (list 0 1 2)

;;; [null (list 9)]

;;; [(cons head tail)

;;; (if (<= 9 0)

;;; (cons 9 (list 0 1 2)

;;; (cons 0 (insert-sorted (list 1 2) 9))))])

;;; --> (if #f

;;; (cons 9 (list 0 1 2)

;;; (cons 0 (insert-sorted (list 1 2) 9))))])

;;; --> (cons 0 (insert-sorted (list 1 2) 9))))])

;;; --> (cons 0

;;; (match (list 1 2)

;;; [null (list 9)]

;;; [(cons head tail)

;;; (if (<= 9 1)

;;; (cons 9 (list 1 2)

;;; (cons 1 (insert-sorted (list 1 2) 9))))]))

;;; --> (cons 0

;;; (if #f

;;; (cons 9 (list 1 2)

;;; (cons 1 (insert-sorted (list 1 2) 9))))]))

;;; --> (cons 0

;;; (cons 1 (insert-sorted (list 1 2) 9))))]))

;;; -> (cons 0

;;; (cons 1

;;; (if #f

;;; (cons 9 (list 2)

;;; (cons 2 (insert-sorted null 9))))]))

;;; -> (cons 0

;;; (cons 1

;;; (cons 2

;;; (match null

;;; [null (list 9)]

;;; [(cons head tail)

;;; (if (<= 9 null)

;;; (cons 9 (list null)

;;; (cons null (insert-sorted null 9))))]))

;;; -> (cons 0 (cons 1 (cons 2 (list 9))))

;;; --> (list 0 1 2 9)

;;; (insertion-sort-helper unsorted) -> list?

;;; unsorted: list?

;;; sorted: list?

;;; Returns sorted but with the elements of unsorted inserted into

;;; sorted so that the whole list is sorted.

(define insertion-sort-helper

(lambda (unsorted sorted)

(match unsorted

[null sorted]

[(cons head tail)

(insertion-sort-helper tail (insert-sorted sorted head))])))

(insertion-sort-helper (list 1 2 3)(list 4 5))

;;; --> (match (list 1 2 3)

;;; [null (list 4 5)]

;;; [(cons head tail)

;;; (insertion-sort-helper tail (insert-sorted (list 4 5) head))])

;;; --> (insertion-sort-helper (list 2 3) (insert-sorted (list 4 5)1))

;;; --> (insertion-sort-helper (list 2 3)

;;; (match (list 4 5)

;;; [null (list 1)]

;;; [(cons head tail)

;;; (if (<= 1 head)

;;; (cons 1 (list 4 5))

;;; (cons head (insert-sorted tail 1)))]))

;;; --> (insertion-sort-helper (list 2 3)

;;; (if (<= 1 4)

;;; (cons 1 (list 4 5))

;;; (cons 4 (insert-sorted (list 5) 1)))]))

;;; --> (insertion-sort-helper (list 2 3)

;;; (if #t

;;;

;;; (cons 4 (insert-sorted (list 5) 1)))]))

;;; --> (insertion-sort-helper (list 2 3)

;;; (cons 1 (list 4 5)))

;;; --> (insertion-sort-helper (list 2 3)

;;; (list 1 4 5))

;;; --> (match (list 2 3)

;;; [null (list 1 4 5)]

;;; [(cons head tail)

;;; (insertion-sort-helper tail (insert-sorted (list 1 4 5) head ))]

;;; --> (insertion-sort-helper tail (insertpsorted (list 1 4 5) head))]

;;; --> (insertion-sort-helper (list 3) (insert-sorted (list 1 4 5) 2))

;;;--> (insertion-sort-helper (list 3)

;;; (match (list 1 4 5)

;;; [null (list 2)]

;;; (if (<= 2 head)

;;; (cons v (list 1 4 5))

;;; (cons head (insert-sorted tail 2))))

;;; --> (insertion-sort-helper (list 3)

;;; (if (<= 2 1)

;;; (cons 2 (list 1 4 5))

;;; (cons 1 (insert-sorted (list 4 5) 2))))

;;; --> (insertion-sort-helper (list 3)

;;; (if #f

;;; (cons 2 (list 1 4 5))

;;; (cons 1 (insert-sorted (list 4 5) 2))))

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1 (insert-sorted (list 4 5) 2))))

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1

;;; (match (list 4 5)

;;; [null (list 2)]

;;; [(cons head tail)

;;; (if (<= 2 head)

;;; (cons 2 (list 4 5))

;;; (cons head (insert-sorted tail 2)))])

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1

;;; (if (<= 2 4)

;;; (cons 2 (list 4 5))

;;; (cons 4 (insert-sorted (list 5) 2)))])

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1

;;; (if #t

;;; (cons 2 (list 4 5))

;;; (cons 4 (insert-sorted (list 5) 2)))])

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1

;;; (cons 2 (list 4 5))))

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1

;;; (cons 2 (list 4 5))))

;;; --> (insertion-sort-helper (list 3)

;;; (cons 1 (list 2 4 5))))

;;; --> (insertion-sort-helper (list 3) (list 1 2 4 5))))

;;; --> (match (list 3)

;;; [null (list 1 2 4 5)]

;;; [(cons head tail)

;;; (insertion-sort-helper tail (insert-sorted (list 1 2 4 5) head))])

;;; --> (insertion-sort-helper null (insert-sorted (list 1 2 4 5) 3))])

;;; --> (insertion-sort-helper null (insert-sorted (list 1 2 4 5) 3))])

;;; --> (insertion-sort-helper null

;;; (match (list 1 2 4 5)

;;; [null (list 3)]

;;; [(cons head tail)

;;; (if (<= 3 head)

;;; (cons 3 (list 1 2 4 5))

;;; (cons head (insert-sorted tail 3)))])

;;; --> (insertion-sort-helper null

;;; (if (<= 3 1)

;;; (cons 3 (list 1 2 4 5))

;;; (cons 1 (insert-sorted (list 2 4 5) 3)))])

;;; --> (insertion-sort-helper null

;;; (if #f

;;; (cons 3 (list 1 2 4 5))

;;; (cons 1 (insert-sorted (list 2 4 5) 3)))])

;;; --> (insertion-sort-helper null (cons 1 (insert-sorted (list 2 4 5) 3)))])

;;; --> (insertion-sort-helper null (cons 1

;;; (match (list 2 4 5)

;;; [null (list 3)]

;;; [(cons head tail)

;;; (if (<= 3 head)

;;; (cons 3 (list 2 4 5))

;;; (cons head (insert-sorted tail 3)))])

;;; --> (insertion-sort-helper null (cons 1

;;; (if (<= 3 head)

;;; (cons 3 (list 2 4 5))

;;; (cons head (insert-sorted tail 3)))])