;;LONGEST INCREASING SUBSEQUENCE

(define longer ;restituisce ila lista più lunga

(lambda (a b )

(if (< (length a) (length b)) b

(if (> (length a) (length b)) a b ))))

(define llis (lambda (lista) (llis-rec lista 0 0) ))

(define llis-rec

(lambda (lista i t) ;i posizone in cui cercare < LUNGH t: numero precedente da confrontare

(let ( (lungh (length lista)) )

(cond ( (= i lungh) 0 )

( (<= (list-ref lista i) t) (llis-rec lista (+ 1 i) t) ) ;continua a cercare

(else (max ;Confronta le 2 lunghezze ;LIS --> (list-ref lista i) < (list-ref lista i+1)

(+ 1 ;viene incrementata l lunghezza della LIS

(llis-rec lista

(+ i 1) ;posizione successiva

(list-ref lista i) )) ;salva il numero appena letto

;NON viene incrementata la lunghezza

(llis-rec lista

(+ i 1) t )) ))))) ;mantiene il dato precedente

(define lis ;restituisce una lista LIS

(lambda (lista) (lis-rec lista 0 0) ))

(define lis-rec

(lambda (lista i t) ;i posizone in cui cercare < LUNGH t: numero precedente da confrontare

(let ( (lungh (length lista)) )

(cond ( (= i lungh) null )

( (<= (list-ref lista i) t) (lis-rec lista (+ 1 i) t) ) ;continua a cercare

(else (longer ;Confronta le 2 liste

;LIS --> (list-ref lista i) < (list-ref lista i+1)

(append ;viene incrementata la lista

(list (list-ref lista i))

(lis-rec lista

(+ i 1) ;posizione successiva

(list-ref lista i) )) ;salva il numero appena letto

;LIS --> (list-ref lista i) >= (list-ref lista i+1)

;NON viene incrementata la lunghezza

(lis-rec lista

(+ i 1) t )) ))))) ;mantiene il dato precedente

;ESERCIZIO 1 03-09-2019 (lis '(27 90 7 29 49 8 53 1 28 6)) → ((7 29 49 53) (27 29 49 53))

;LISTA DI TUTTE LE LIS della lista passata come argomento

(define lis ; val : lista di liste (di interi)

(lambda (s) (lis-rec s 0 null) )) ; s : lista di interi

(define best ;retituise la lista più lunga

(lambda (a b) ;a, b: LISTE

(if (< (length a) (length b)) a

(if (= (length a) (length b))

(list a b)

b ))))

(define lis-rec

(lambda (s t p)

(cond ( (null? s) (list (reverse p)) )

( (<= (car s) t) (lis-rec (cdr s) t p) )

(else (best (lis-rec (cdr s) t p)

(lis-rec (cdr s) (car s) (cons (car s) p)) )))))