Q1 (A) #t

(cdde (list (list 2 3 4) 5))

First cdr : (list 5)

Second cdr : ()

½ for #t ; ½ for explanation
(out ½ for variations of #t)

(B) 147

Outer lambda applied to 3, 5

∴ x : 3
y : 5

Inner (returned) lambda applied to 2, 7

∴ z : 2
w : 7

(* (+ 2 5) (* 7 21)) = (* 3 7)) = 147

½ + ½

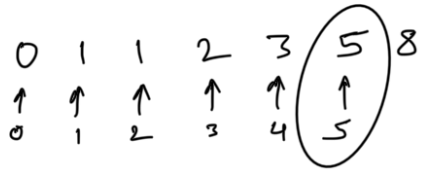
Any variation of this explanation is fine

Q2 (A)

```
(define (fib n)
  (define (fib-iter a b count)
    (if (= count 0)
        b
        (fib-iter (+ a b) a (- count 1))))
  (fib-iter 1 0 n))
```

1 mark
(scoping / naming variations are fine)

(fib 5)
(fi 1 0 5)
(fi 1 1 4)
(fi 2 1 3)
(fi 3 2 2)
(fi 5 3 1)
(fi 8 5 0)



1 mark
(wrong index : -½)

(B) Tail-call optimization allows reusing the stack frame when recursive calls to a procedure occur at the tail position; i.e., without any computation left to be performed after unwinding the recursion. 1 mark for a reasonable explanation (above process not discussed: -½)
This happens with the above process too. Thus, with TCO, the six calls to "fi" can share the same stack frame.

0.5 marks for when TCO happens (function calls itself/another function with no added computation, etc)
0.5 marks for how the optimization works (best would be showing the stack frames and noting that there is only 1 stack frame at each moment, earlier stack frames are reused) and hence space complexity reduces

Q3 (A) car (cons v w)

$=_\beta$ car $((\lambda f.\lambda s.\lambda b.\ b\ f\ s)\ v\ w)$

$=_\beta$ car $((\lambda s.\lambda b.\ b\ v\ s)\ w)$

$=_\beta$ car $(\lambda b.\ b\ v\ w)$

$=$ $(\lambda p.\ p\ \text{true})\ (\lambda b.\ b\ v\ w)$

$=_\beta$ $(\lambda b.\ b\ v\ w)\ \text{true}$

$=_\beta$ true v w

$=$ v

2 marks (partial if jumped steps)

(B) cdr $= \lambda p.\ p$ false , where false x y $=$ y

1 mark if both cdr & false are defined

0.5 marks if false not defined explicitly. (Implicitly assuming definition of false in derivation does not count)

Q4

```
(define (reverse l)
  (if (null? l)
      nil                ; l also works
      (append (reverse (cdr l)) (list (car l)))))
```

2 marks if completely right

-½ without this

1 for missing base case. -0.5 for minor misplaced parentheses -0.5 for not having the arguments of append as lists

Bonus

```
(define (flip l)
  (cons b a))
(define (reverse l)
  (foldl flip nil l))
```

Put a ☆ in the front page if done right.

Only fully correct solutions for a PC. Allowed cases where foldl was redefined to have different order or arguments for f.