Exam 1 Guide

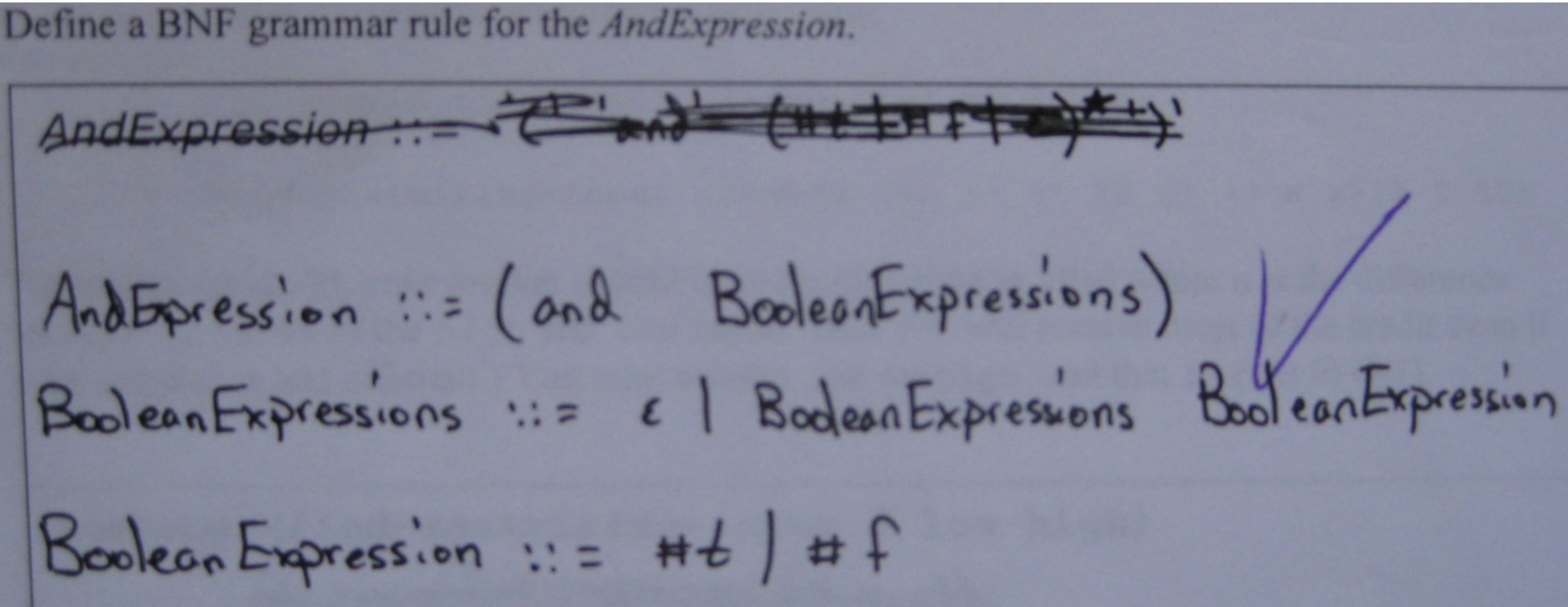

CATENA by Tracy Pierce
Tracy Pierce 01-23-2006
I WILL RULE THE WORLD! MARK MY WORDS...
~~I WILL RULE THE WO~~... ~~MARK MY WORDS~~...
!
... I WOULD RUN.
© 2006 Tracy Pierce

Outline

- Class Average: 83 (59 to 102)
- Grades visible on Automatic Adjudicator
- A curve will be applied later
- Think carefully before asking for a regrade
  - We will look carefully at your answer!
- Not wasting our time: 3/10

Question 1

- Terminals: -1, Epsilon -3, Recursion -3, Correctness -2, Expression to #t/#f only -2



Define a BNF grammar rule for the *AndExpression*.

~~AndExpression ::= '(' 'and' (#t | #f | [illegible])* ')'~~

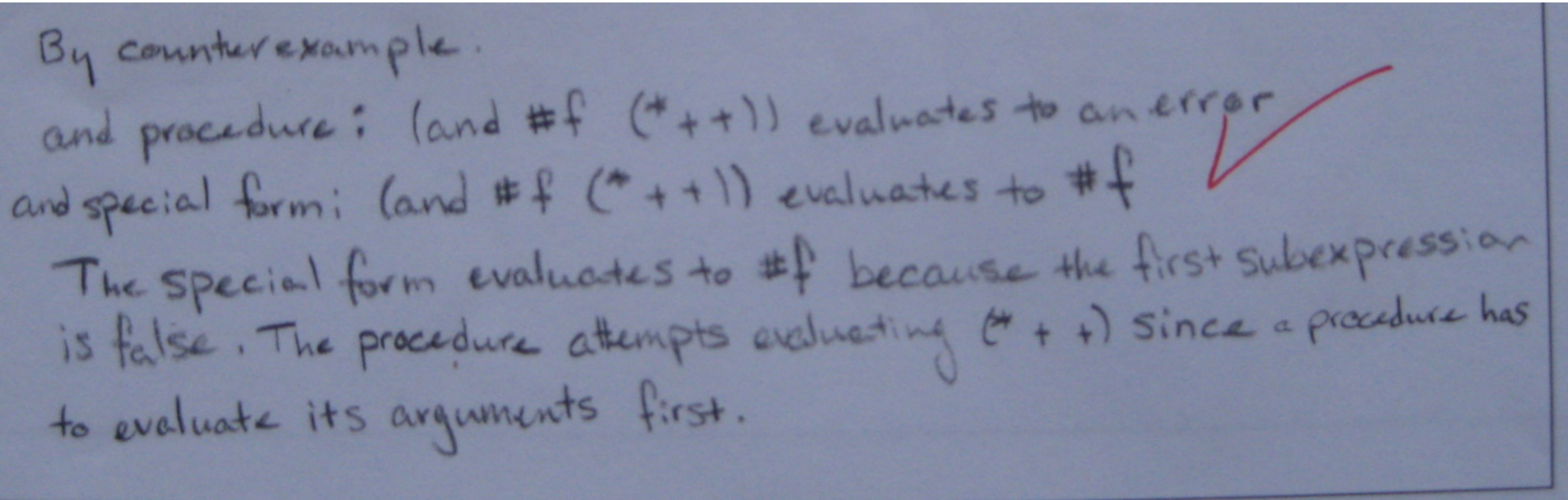
AndExpression ::= (and BooleanExpressions)

BooleanExpressions ::= ε | BooleanExpressions BooleanExpression ✓

BooleanExpression ::= #t | #f

Question 2

- Something about #t/#f vs. all values: -5
- Vague right idea, w/o example: -2
- Reversed: -1



By counterexample.

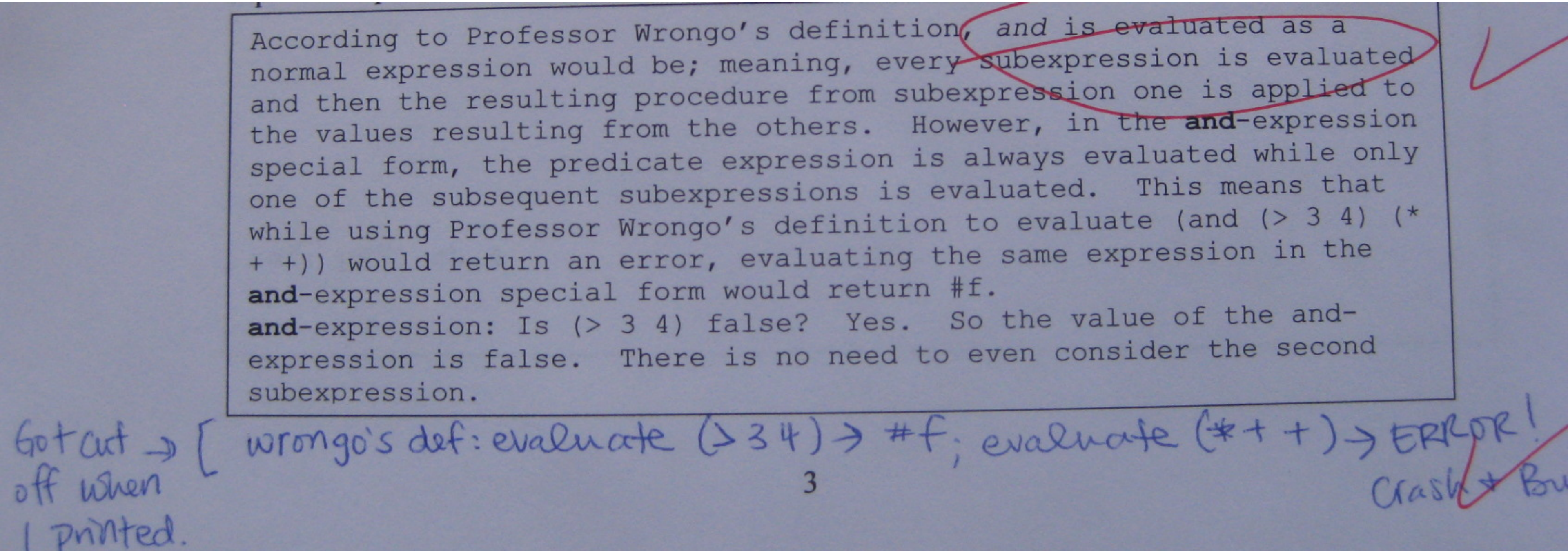
and procedure: (and #f (* + +)) evaluates to an error

and special form: (and #f (* + +)) evaluates to #f

The special form evaluates to #f because the first subexpression is false. The procedure attempts evaluating (* + +) since a procedure has to evaluate its arguments first.

Question 2

- Another phrasing:



According to Professor Wrongo's definition, *and* is evaluated as a normal expression would be; meaning, every subexpression is evaluated and then the resulting procedure from subexpression one is applied to the values resulting from the others. However, in the **and**-expression special form, the predicate expression is always evaluated while only one of the subsequent subexpressions is evaluated. This means that while using Professor Wrongo's definition to evaluate (and (> 3 4) (* + +)) would return an error, evaluating the same expression in the **and**-expression special form would return #f.

and-expression: Is (> 3 4) false? Yes. So the value of the and-expression is false. There is no need to even consider the second subexpression.

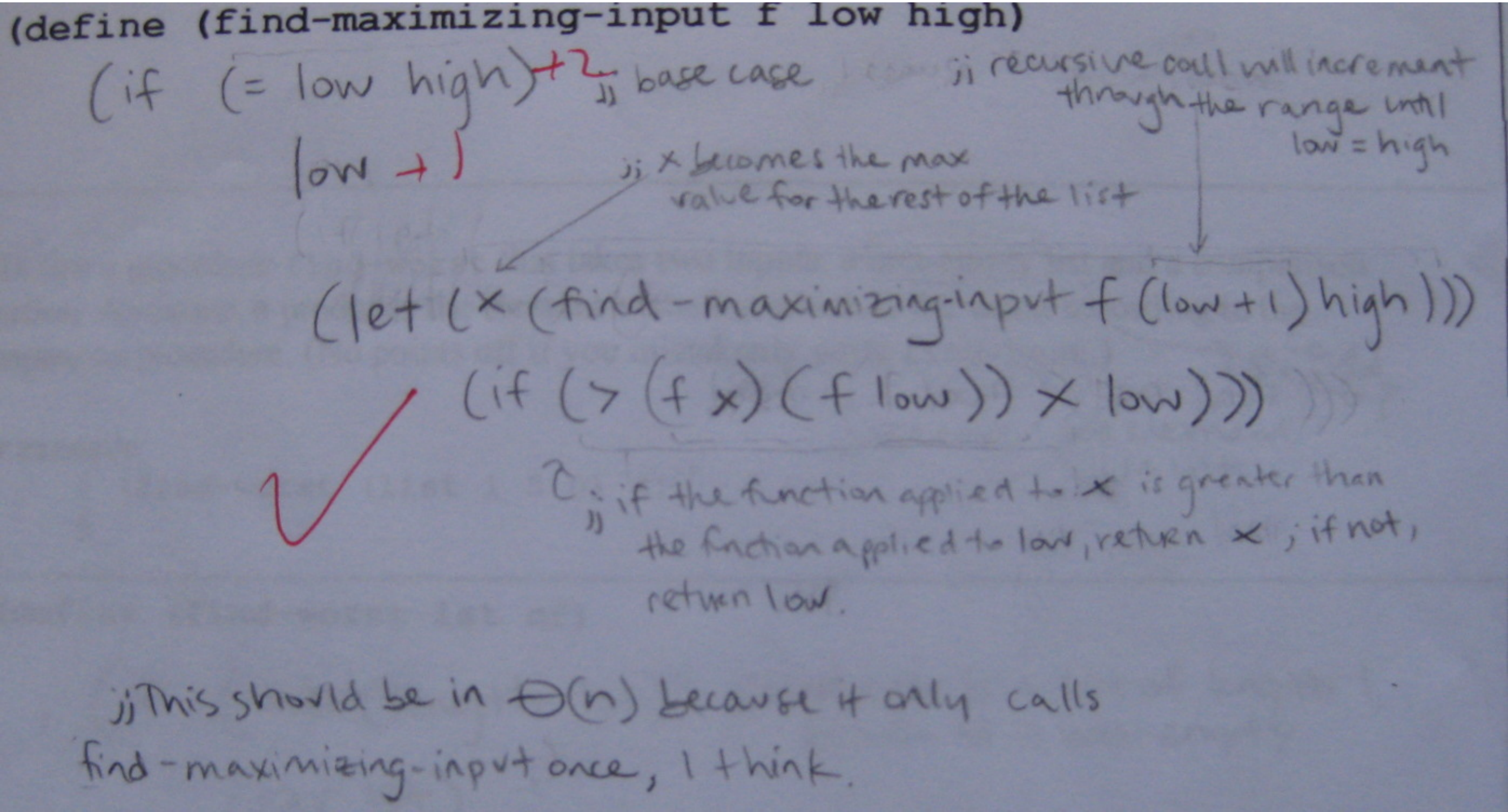
Got cut off when I printed. → [wrongo's def: evaluate (> 3 4) → #f; evaluate (* + +) → ERROR!

Crash + Bu

3

Question 3

- **(= low high)** : -2, **low** as base result : -1, **if** in recursive case : -2, recursive call : -3, infinite loop: -3, non-linear time: -2, explanation: -1



```
(define (find-maximizing-input f low high)
(if (= low high) +2 ;; base case        ;; recursive call will increment
                                         through the range until
  low +1                                 low = high
                  ;; x becomes the max
                     value for the rest of the list

  (let (x (find-maximizing-input f (low +1) high)))
    (if (> (f x) (f low)) x low)))

             ? ;; if the function applied to x is greater than
               ;; the function applied to low, return x ; if not,
                  return low.

;;This should be in Θ(n) because it only calls
find-maximizing-input once, I think.
```

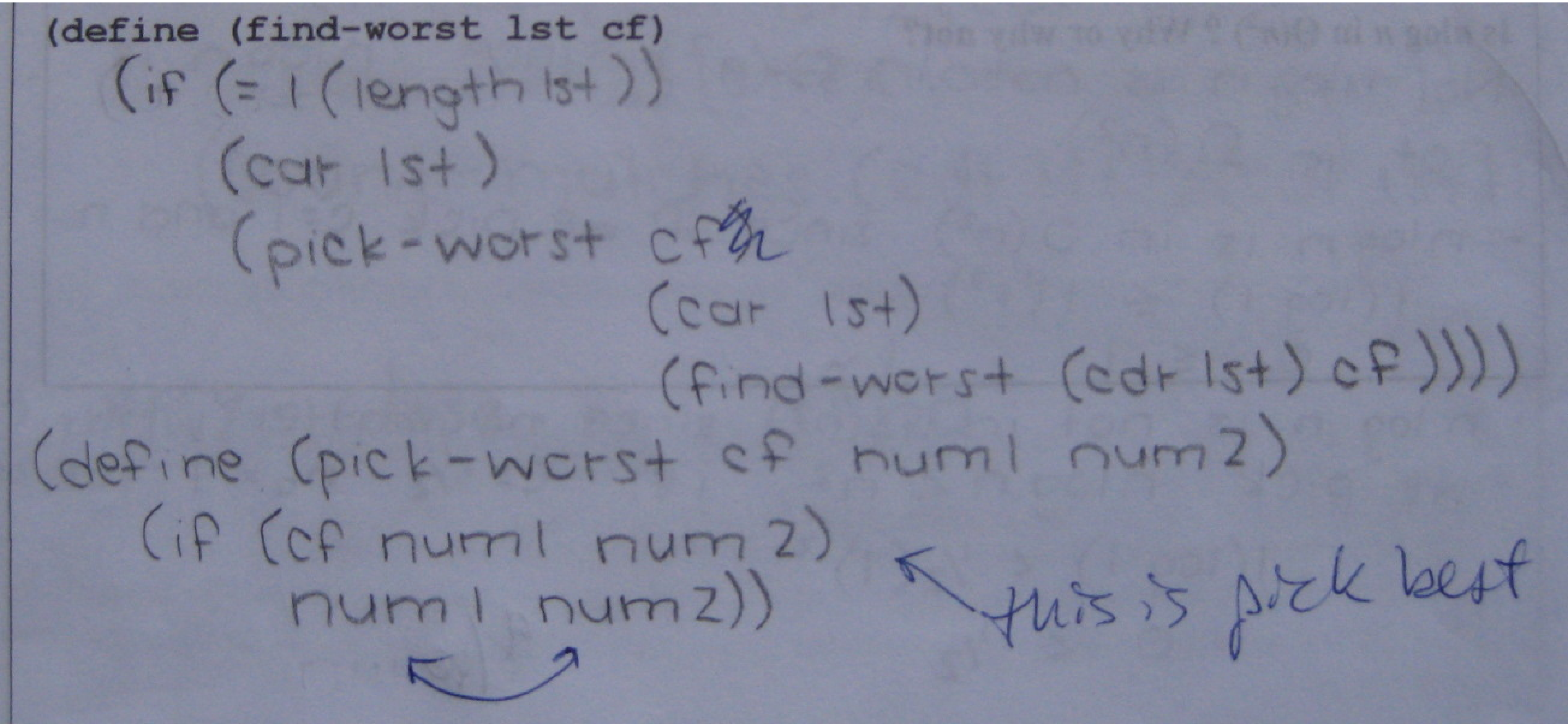
Question 4

- **lambda** : -5, **(x)** : -2, **(+ x n)** : -3

```
(define (make-incrementer n)
  (lambda (x) (+ x n))
```

Question 5

- loosely -2 per wrong element

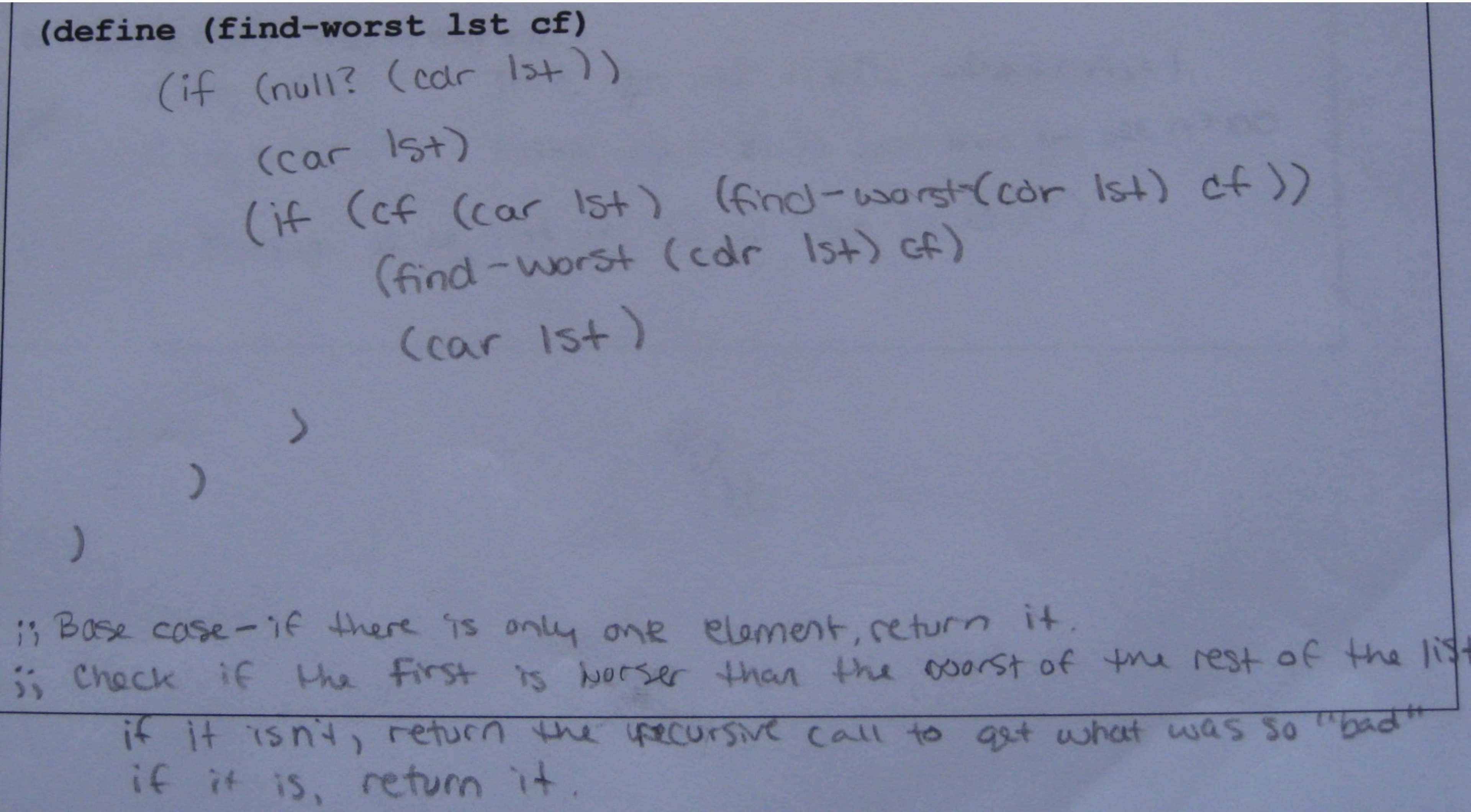


```
(define (find-worst lst cf)
  (if (= 1 (length lst))
      (car lst)
      (pick-worst cf
                  (car lst)
                  (find-worst (cdr lst) cf))))
(define (pick-worst cf num1 num2)
   (if (cf num1 num2)
       num1 num2))
```

this is pick best

Question 5

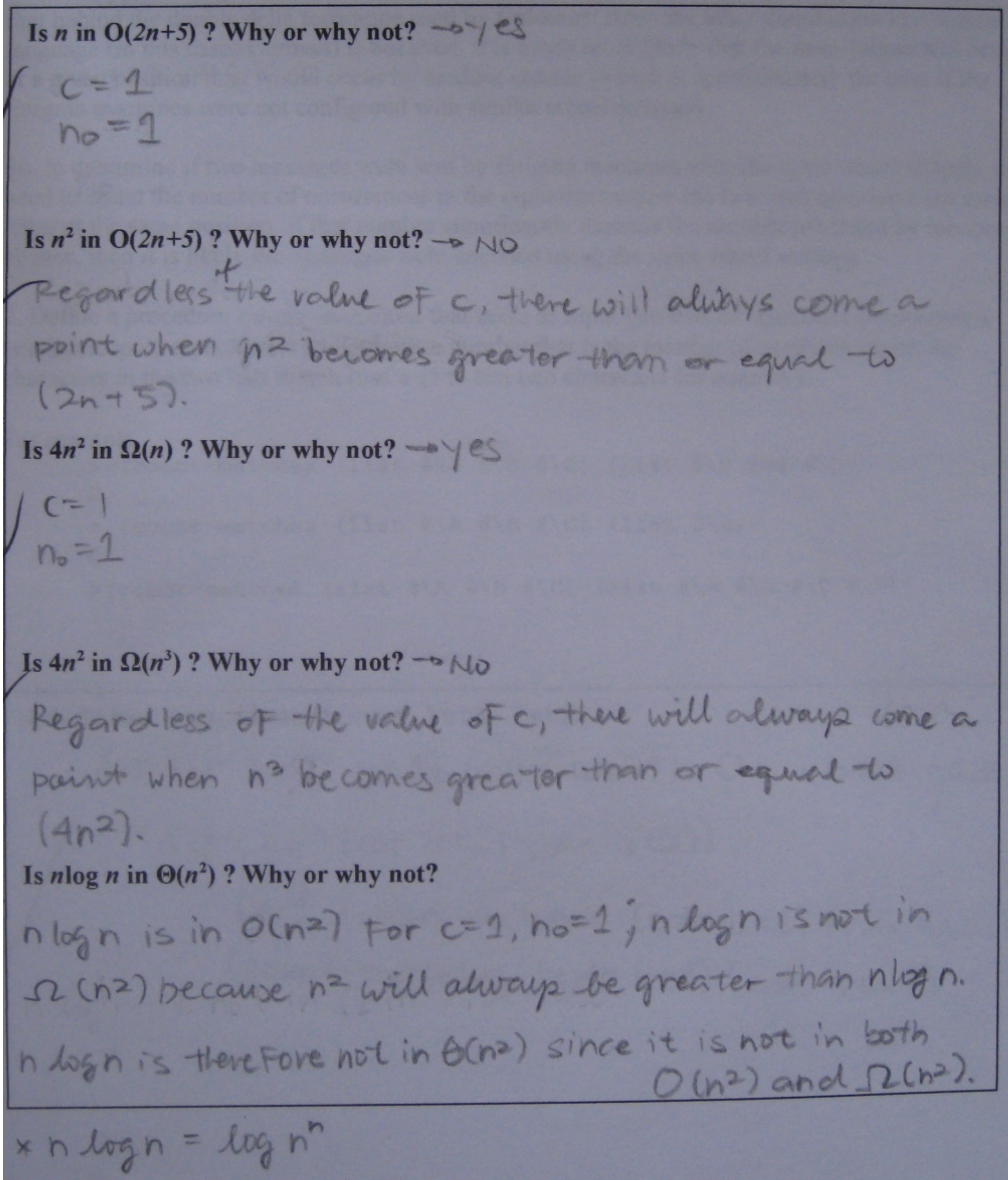
- This one is not Theta(n), but is still full credit. Also: sort and take the car.



```
(define (find-worst lst cf)
      (if (null? (cdr lst))
          (car lst)
          (if (cf (car lst) (find-worst (cdr lst) cf))
               (find-worst (cdr lst) cf)
                (car lst)

          )

      )

 )

;; Base case - if there is only one element, return it.
;; Check if the first is worser than the worst of the rest of the lis
       if it isn't, return the recursive call to get what was so "bad"
       if it is, return it.
```

Question 6

- 1 point per correct yes/no
- 1 point per correct explanation
- weak overall: -1 or -2
- Gotcha: n0 >= 1 for Part 3



Is n in $O(2n+5)$? Why or why not? → yes

$c = 1$
$n_0 = 1$

Is n^2 in $O(2n+5)$? Why or why not? → NO

Regardless the value of c, there will always come a point when n^2 becomes greater than or equal to $(2n+5)$.

Is $4n^2$ in $\Omega(n)$? Why or why not? → yes

$c = 1$
$n_0 = 1$

Is $4n^2$ in $\Omega(n^3)$? Why or why not? → No

Regardless of the value of c, there will always come a point when n^3 becomes greater than or equal to $(4n^2)$.

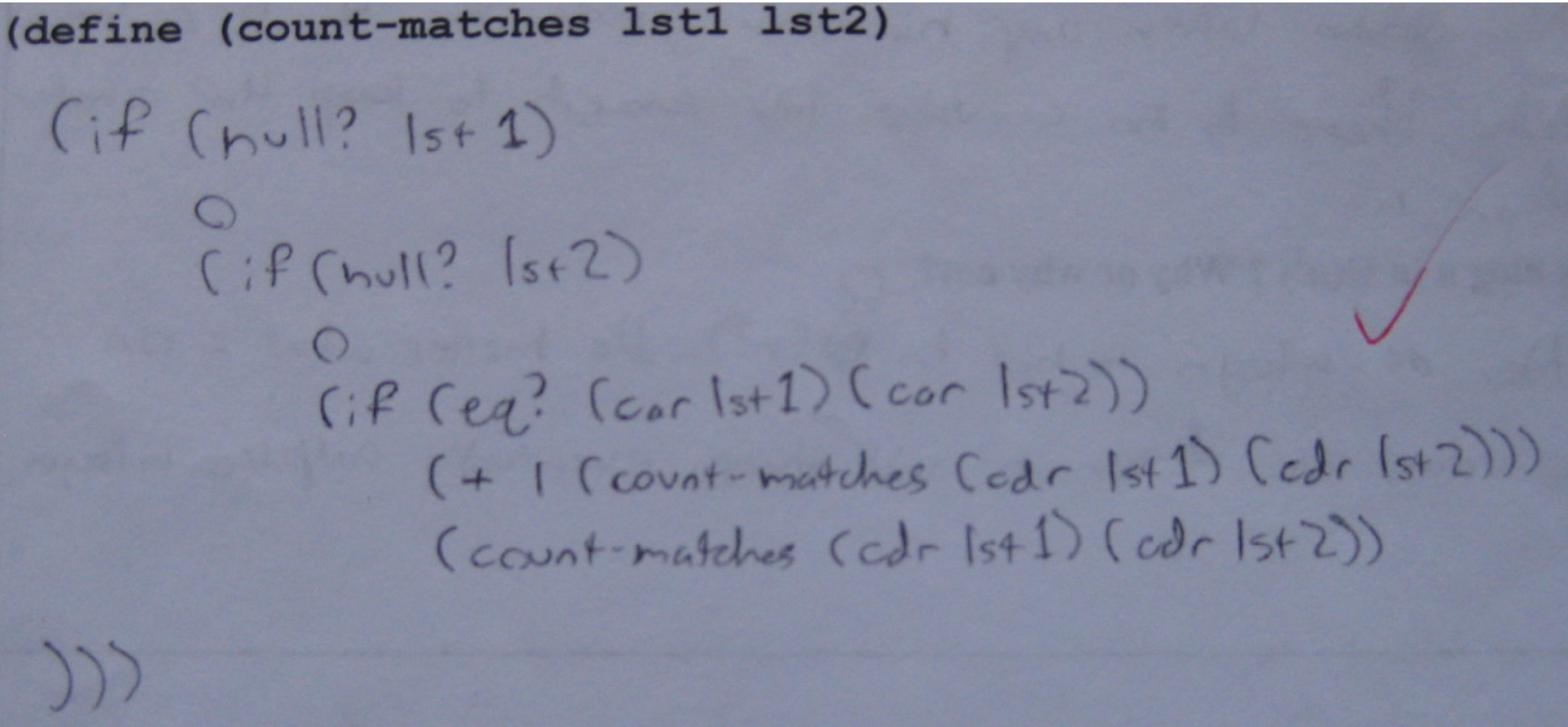
Is $n\log n$ in $\Theta(n^2)$? Why or why not?

$n \log n$ is in $O(n^2)$ for $c=1$, $n_0=1$; $n \log n$ is not in $\Omega(n^2)$ because n^2 will always be greater than $n \log n$. $n \log n$ is therefore not in $\Theta(n^2)$ since it is not in both $O(n^2)$ and $\Omega(n^2)$.

× $n \log n = \log n^n$

Question 7

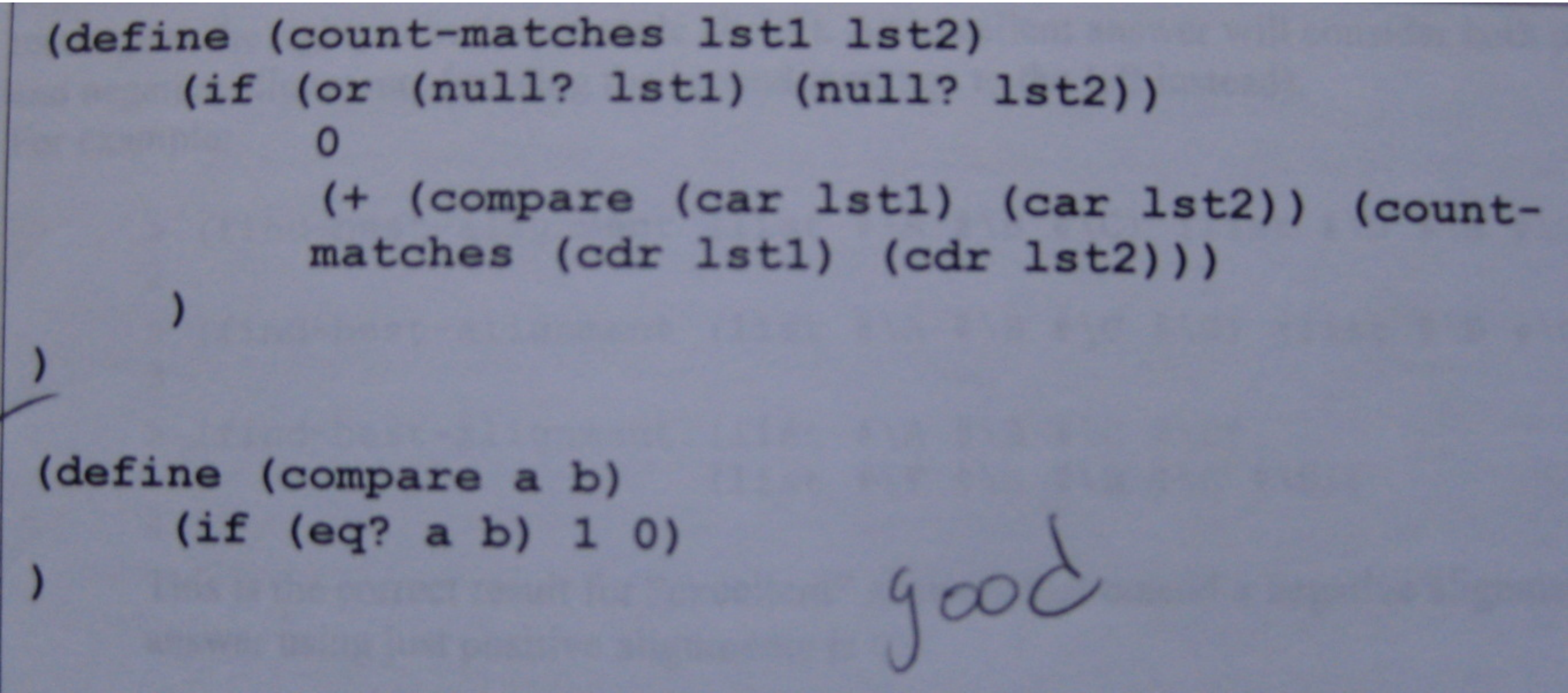
- base case conditional: -2, base case result: -1, plus to combine results in recursive step: -2, use of eq? : -1, recursive call: -3, correct use of car/cdr: -1, other errors: -1



```
(define (count-matches lst1 lst2)
  (if (null? lst1)
      0
      (if (null? lst2)
          0
          (if (eq? (car lst1) (car lst2))
              (+ 1 (count-matches (cdr lst1) (cdr lst2)))
              (count-matches (cdr lst1) (cdr lst2))
)))
```

Question 7

- Another writeup:

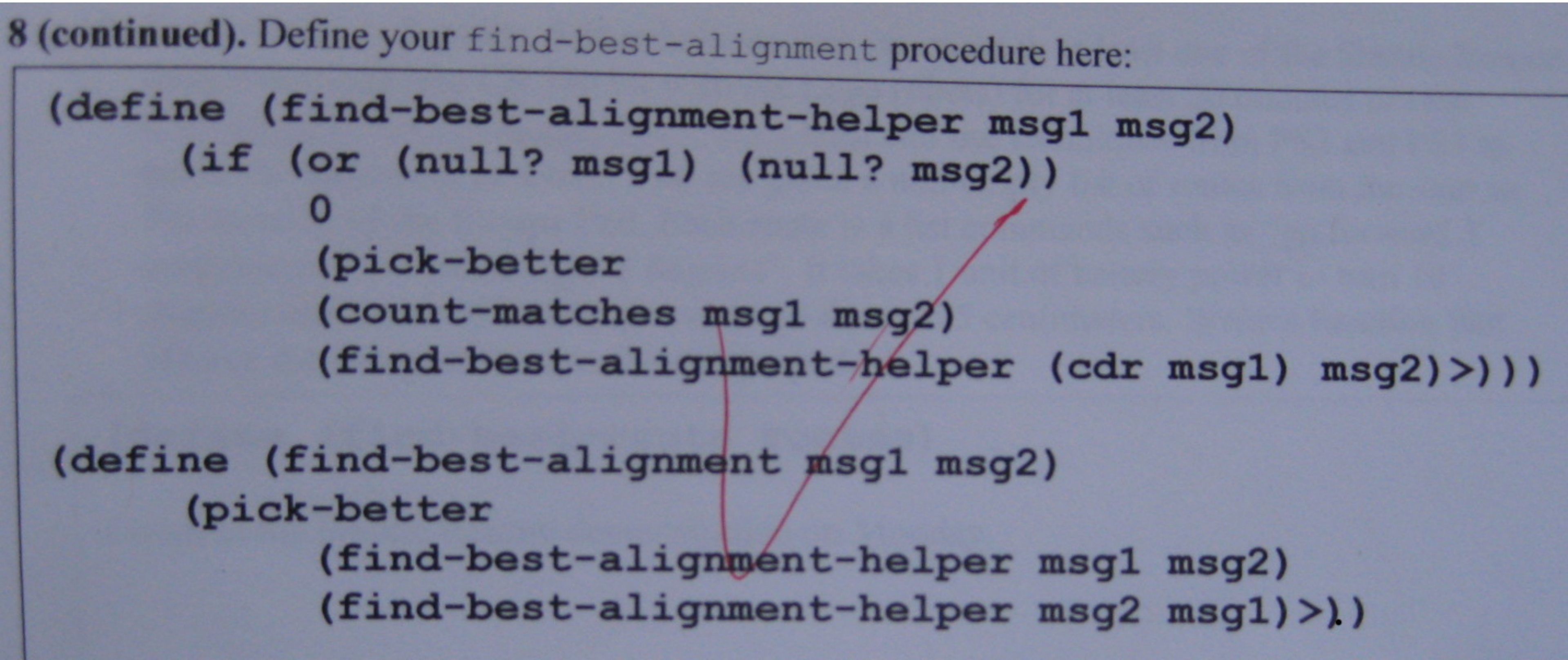


```
(define (count-matches lst1 lst2)
     (if (or (null? lst1) (null? lst2))
          0
          (+ (compare (car lst1) (car lst2)) (count-
          matches (cdr lst1) (cdr lst2)))
     )
)

(define (compare a b)
     (if (eq? a b) 1 0)
)
```

good

Question 8

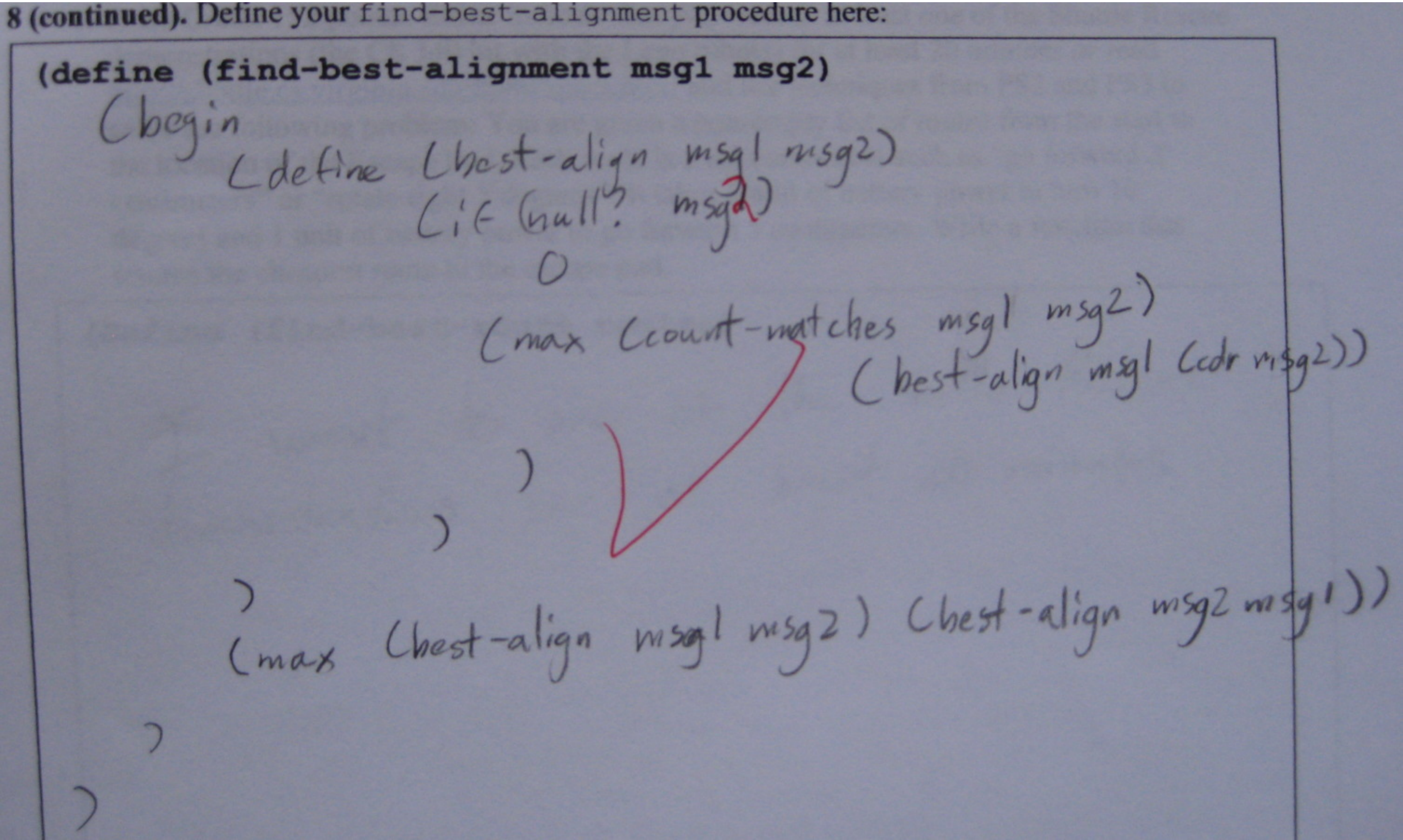
- base case conditional: -2, base case result: -1, use of pick-better or similar: -1, count-matches: -2, recursive call: -2, describe code without writing it: up to -3, did not “go both ways”: -1



8 (continued). Define your `find-best-alignment` procedure here:

```
(define (find-best-alignment-helper msg1 msg2)
     (if (or (null? msg1) (null? msg2))
          0
          (pick-better
          (count-matches msg1 msg2)
          (find-best-alignment-helper (cdr msg1) msg2)>)))

(define (find-best-alignment msg1 msg2)
     (pick-better
          (find-best-alignment-helper msg1 msg2)
          (find-best-alignment-helper msg2 msg1)>))
```

Question 8

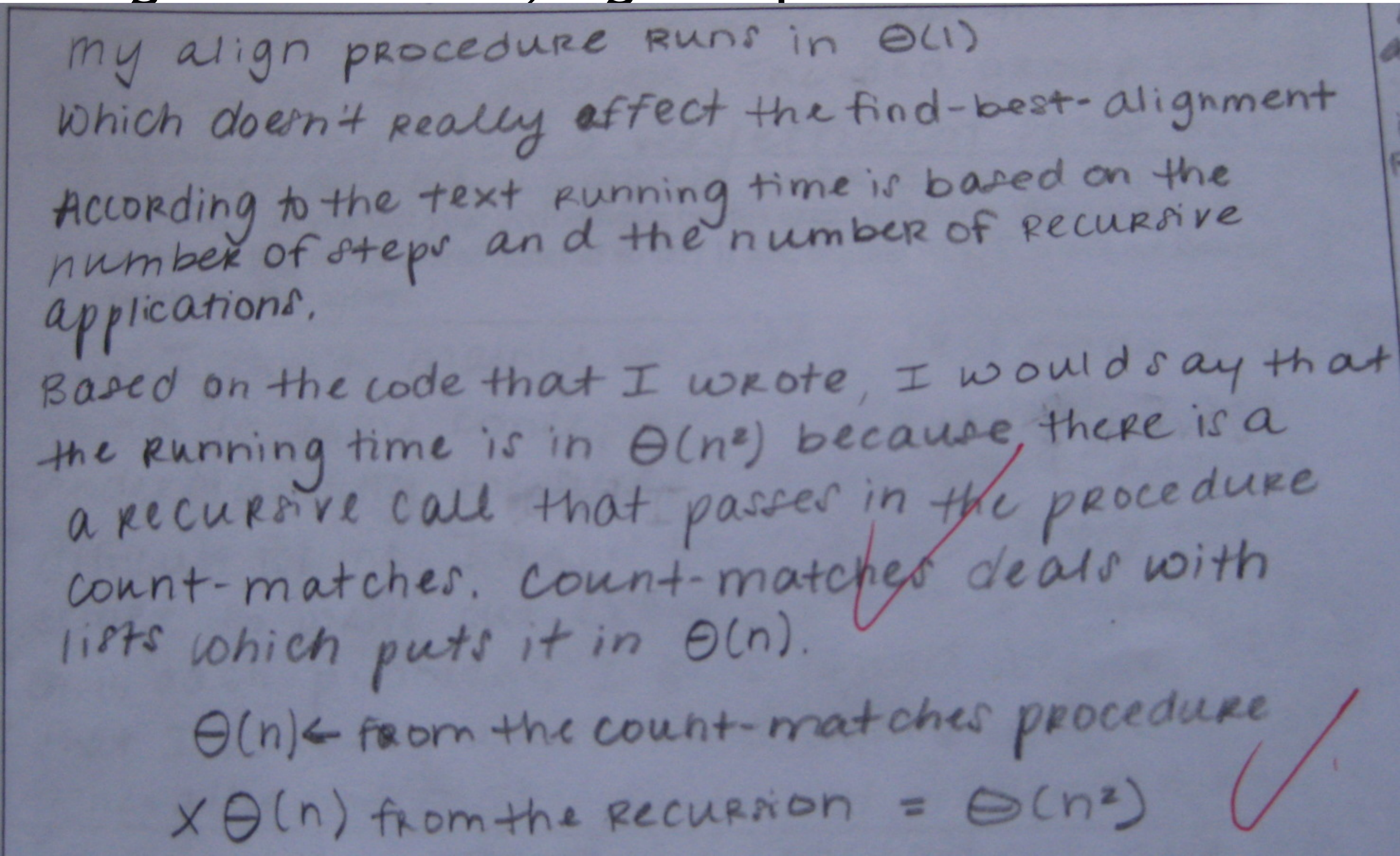


8 (continued). Define your `find-best-alignment` procedure here:

```
(define (find-best-alignment msg1 msg2)
(begin
    (define (best-align msg1 msg2)
            (if (null? msg2)
                  0
                  (max (count-matches msg1 msg2)
                              (best-align msg1 (cdr msg2))
                  )
            )
    )
    (max (best-align msg1 msg2) (best-align msg2 msg1))
  )
)
```

Question 9

- Right answer: -5, right explanation: -5



my align procedure runs in $\Theta(1)$
which doesn't really affect the find-best-alignment

According to the text running time is based on the number of steps and the number of recursive applications.

Based on the code that I wrote, I would say that the running time is in $\Theta(n^2)$ because there is a recursive call that passes in the procedure count-matches. Count-matches deals with lists which puts it in $\Theta(n)$.

$\Theta(n) \leftarrow$ from the count-matches procedure
$\times\, \Theta(n)$ from the recursion $= \Theta(n^2)$