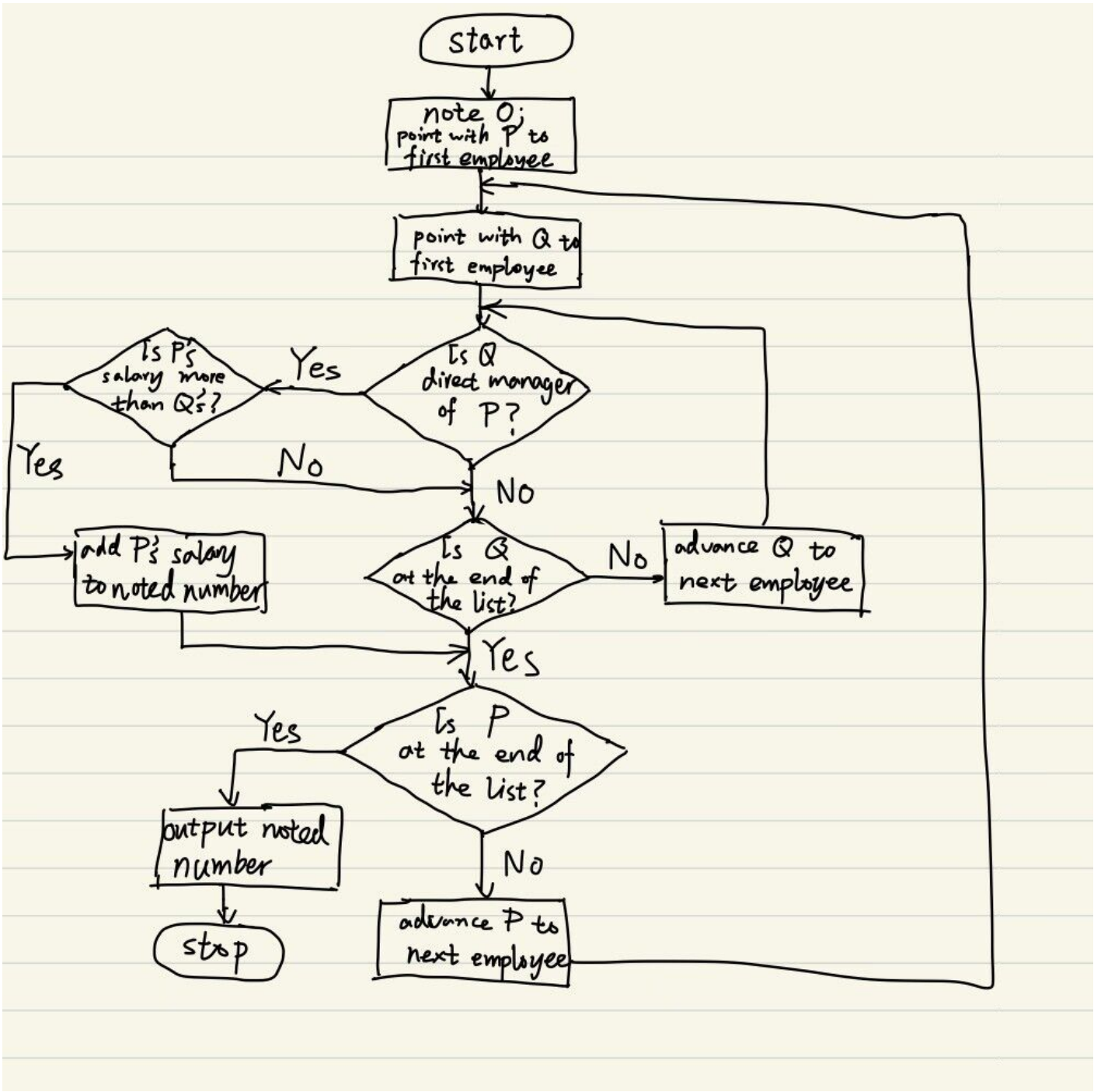
# Homework

## Problems on DH

### 5.4

#### (a)




#### (b)

*Proof.*

**(i)** Firstly, we want to prove its partial correctness by Floyd's Method.

We claim the following invariants and attach them to 4 checkpoints:

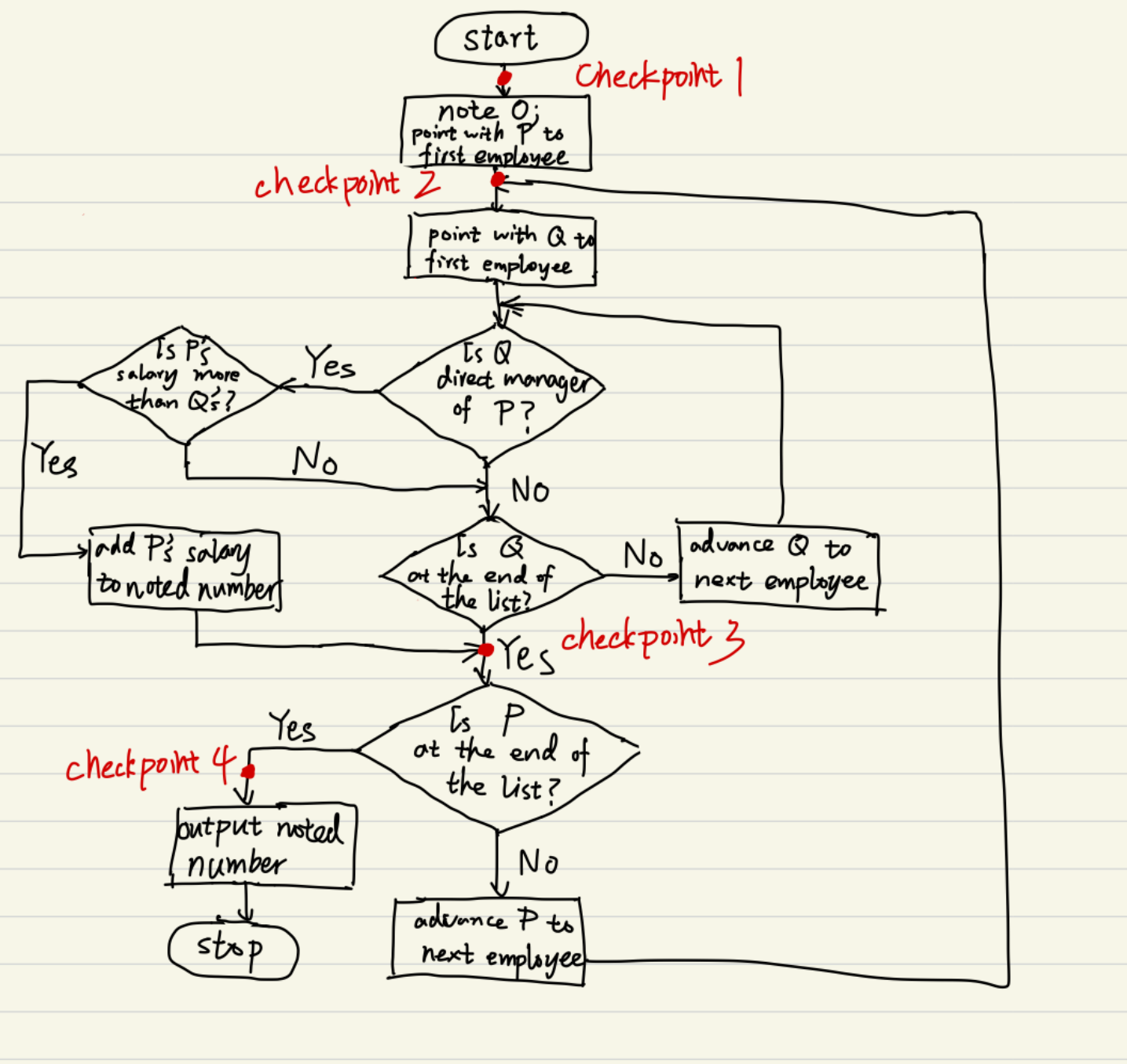
**Assertion 1:** The input is a list of employees with their salaries.

**Assertion 2:** The salaries of employees before P who earn more than at least one of their direct managers have been added to noted number.

**Assertion 3:** The salaries of employees before the employee next to P who earn more than at least one of their direct managers have been

added to noted number.

**Assertion 4:** The output (noted number) is the sum of the salaries of all the employees on the list who earn more than their managers.




Now we have to prove:

**(1→2): If the input is a list of employees with their salaries, then after carrying out the instruction note 0 and point with P to first employee, then the salaries of employees before P who earn more than at least one of their direct managers have been added to noted number.**

Since P is at the present pointed to the first employee, then there is no employees before P, thus no employees who earn more than their direct managers before P, so nothing has to be added to noted number, so it's 0. Q.E.D

**(2→3): If the salaries of employees before P who earn more than at least one of their direct managers have been added to noted number, then after carrying out the instruction, then the salaries of employees before the employee next to P who earn more than at least one of their direct managers have been added to noted number.**

The inner loop guarantees that P's salary will be compared with all his/her direct managers till it is discovered that P earns more than one of them or less than all of them. And if P earns more than one of his/her direct manager(namely there exists situation where P earns more than Q), then after carrying out the instruction add P's salary to noted number, P's salary has been added to noted number, so from checkpoint 2 to checkpoint 3, after carrying out the instructions, if P earns more than at least one of his/her direct manager, then P's salary is added to noted number. Since Assertion 1 is true, namely the salaries of employees before P who earn more than at least one of their direct managers have been added to noted number, then all these together imply that the salaries of employees before the employee next to P who earn more than at least one of their direct managers have been added to noted number. Q.E.D

**(3→4): The salaries of employees before the next employee of P who earn more than at least one of their direct managers have been added to noted number, then if P is at end of the list, the output (noted number) is the sum of the salaries of all the employees on the list who earn more than their managers.**
Since P is at end of the list, so "the employees before the employee next to P" actually means all the employees on the list here. So if P is at end of the list, "the salaries of employees before the employee next to P who earn more than at least one of their direct managers have been added to noted number" in fact means the salaries of all the employees on the list who earn more than at least one of their direct managers have been added to noted number. So the output (noted number) is the sum of the salaries of all the employees on the list who earn more than their managers. Q.E.D

**(2→2): If the salaries of employees before P who earn more than at least one of their direct managers have been added to noted number, then if P isn't at end of the list, after carrying out the instructions (advance P to next employee P'), the salaries of employees before P' who earn more than at least one of their direct managers have been added to noted number.**
Since we have proved **(2→3)** is true, we just have to show that **(3→2)** is true. If the salaries of employees before the employee next to P who earn more than at least one of their direct managers have been added to noted number and P isn't at end of the list, then after advance P to next employee (P'), we can see that P' is the employee next to P, so since the salaries of employees before the employee next to P who earn more than at least one of their direct managers have been added to noted number, then after advance P to P', we can say the salaries of employees before P' who earn more than at least one of their direct managers have been added to noted number, which implies that **(3→2)** is true.

In conclusion, we just showed that CA is partial correct by Floyd's Method.
**(ii)** Now we want to show that CA terminates.
Consider the outer loop **(2→2)**, the algorithm will escape the loop when P is at end of the list. Since the given input of list is infinite and P moves to one employee at one time, then P will eventually reached the end of the list, then the algorithm escapes the loop. Namely we can conclude that the distance between P and the end of the list is the convergent and it will converge to 0, when the outer loop is escaped. In the same way we can prove that the distance between Q and end of the list is convergent of the inner loop and it won't decrease forever. So both the inner and the outer loops are finite loop, so the algorithm will terminate.

By proving the partial correctness and termination of CA, we just have shown that CA is totally correct. Q.E.D

### (c)

Yes, it is.

### (d)

No, it isn't.

## 5.6

### (a)

For all the algorithms with only one loop, three (well-placed) invariants are sufficient for the proof.

### (b)

The flowchart of an algorithm without any loops.

### (c)

Four.

### (d)

Two. **(i)** If the two loops are independent respectively and neither of them connect to the start or stop points, then four invariants are necessary. **(ii)** If one loop is nested in another, and the outer loop dosen't connect to the start or stop points, or if the two loops are independent respectively, but one loop connects to the start or stop point, then three invariants are necessaary. **(iii)** If one loop is nested in another, and the outer loop connects to the start point, then two invariants are necessary.

## 5.8

Suppose L represents $\Lambda$, namely empty string.

```
rev(X):
(1) X = S, Y = L;
(2) if eq(last(X), last(L)) is true, then X is empty.
    (2.1) stop;
(3) otherwise, X isn't empty.
(4) while X isn't empty, do the following:
    (4.1) Y = Y·last(X), X = all-but-last(X);
(5) output Y, which is the reverse of S;
(6) stop;
```

*Proof.*
*Partial Correctness:*
**Assertion 1:** The input X is a string.
**Assertion 2:** Every time before loop (4) is conducted, S = X·rev(Y).
**Assertion 3:** The output Y = rev(S).
**(1→2):** Since X is a string and X = S, Y = L, then obviously we have S = X·rev(Y).
**(2→3):** Since we havve S = X·rev(Y), then when the algorithm escapes the loop, namely when X is empty, then we have S = L·rev(Y), so S = rev(Y), namely Y =rev(S).
**(2→2):** Since we have S = X·rev(Y), then if X isn't empty, after carrying the instructions Y' = Y·last(X), X' = all-but-last(X), so X'·rev(Y') = all-but-last(X)·last(X)·rev(Y) = X·rev(Y) =S.
In conclusion, we have proved that the algorithm is partially correct.
*Termination:*
The length of string X is the convergent and every time the loop is conducted, its length will be minused by one, and when it converges to 0 (X is empty), the loop escaped and the algorithm terminates.
Now we have showed the algorithm is partial correct and it terminates, thus it's total correct.

## 5.9

```
equal(X,Y):
(1) while neither X nor Y is emtpy, do the following:
    (1.1) A = head(X), B = head(Y);
    (1.2) X = tail(X), Y = tail(Y);
    (1.3) if eq(A, B) is false, then return false;
(2) if both X and Y are empty, then return true;
(3) otherwise return false;
```

*Proof.*
*Partial Correctness:*
**Assertion 1:** Both X and Y are symbol strings.
**Assertion 2:** The original strings aren't equal.
**Assertion 3:** The original X and Y are equal.
**Assertion 4:** The original X and Y aren't equal.
**(1→2):** Since X and Y are symbol strings, and A = head(X), B = head(Y), and then because A and B aren't equal, namely head(X) and head(Y) are't equal, then obviously X and Y aren't equal.
**(1→3):** Since X and Y are symbol strings, and X and Y are both empty, then obviously X and Y are equal.
**(1→4):** Since X and Y are symbol strings, but at this point one of them is empty while the other isn't, then apparently they aren't equal.
**(1→1):** Since X and Y are symbol strings, after carrying out the instructions, X' = tail(X), Y' = tail(Y), obviously X' and Y' are still symbol strings.
*Termination:*
The length of the shorter one of X and Y is the convergent. Every time the loop is conducted, by instruction X = tail(X), Y = tail(Y), both their lengths (including the length of the shorter) would be minused by one, and finally reach zero (converge to zero). Once zero is hit, the algorithm escapes the loop, so the algorithm will terminate with a convergent.
Since we have proved the partial correctness and termination of the algorithm, we have actually proved its total correctness.

## 5.10

### (a)

*Proof.*
*Partial Correctness:*
**Assertion 1:** S is a symbol string.
**Assertion 2:** equal(S, Y) is true, if and only if S is a palindrome.
**(1→2):** Because S is a symbol string, then from 5.8 we can know that Y is the reverse of S. From 5.9 we can know that equal(S, Y) is true if and

only if S and Y are equal, namely same string. So equal(S, Y) is true, if and only if S and it's reverse are exactly same strings, namely coinciding the definition of palindrome, so equal(S, Y) is true, if and only if S is a palindrome.
*Termination:*
From 5.8 and 5.9 we can see that both rev(S) and equal(S, Y) will terminate, and when they all terminate, Pall terminates instantly. So Pall will terminate.
By proving the partial correctness and termination of Pall, we have already proved the total correctness of Pall.

### (b)

Consider an algorithm that consists of two sub-algorithms in sequent order, then if all the two sub-algorithms terminate, then the algorithm terminates, too.

## 5.11

### (a)

ab

### (b)

Three.
(1) A = head(ab) = a.
(2) B = last(ab) = b.
(3) eq(A, B) is false.

## 5.12

### (a)

*Proof.*
**Assertion 1:** X is a symbol string, X = S.
**Assertion 2:** S is palindrome if and only if X is palindrome and E is true.
**Assertion 3:** S is palindrome if and only if E is true.
**(1→2):** Because X is a symbol string and X = S, then after carrying out the instruction E = true, it is obvious that S is palindrome is equivalent with X is a palindrome and E is true.
**(2→3):** Because S is palindrome if and only if X is palindrome and E is true, then from checkpoint 2 to 3, namely when X is empty, it is obvious that an empty string is palindrome, so at this point X is always palindrome, so X is palindrome and E is true $\leftrightarrow$ E is true, so S is palindrome if and only if E is true.
**(2→2):** Since S is palindrome if and only if X is palindrome and E is true, then **(i)** if head(X) = last(X) and after X = all-but-last(tail(X)), X is palindrome $\leftrightarrow$ head(X) = last(X) and all-but-last(tail(X)) is palindrome, so S is palindrome $\leftrightarrow$ X is palindrome and E is true $\leftrightarrow$ head(X) = last(X) and all-but-last(tail(X)) is palindrome and E is true. **(ii)** if head(X) $\neq$ last(X) and after E = false, then X is apparently not palindrome, neither is S, and E is false, so we still have S is palindrome if and only if X is palindrome and E is true.

### (b)

Disprove by counterexample: If S = ab, then eq(head(X), last(X)) will always be false, while X will always be non-empty, so the algorithm stumbles into infinite loops and never terminates.

## 5.13

### (a)

*Disproof.*
If the length of S is odd and S is palindrome, then finally the length of X will be one, and tail(X) = L, namely Y = L, so last(Y) = L, but head(X) $\neq$ L, so E = false, which is fault.

### (b)

If eq(head(X), last(Y)), then X = all-but-last(Y), so then length of X will be minused by one, or if eq(head(X), last(Y)) is false, then E is false. So with the process of the algorithm, either length of X will be decreased to 0 or E is assigned to be false, and both these situations make the algorithm escape the loop and terminate. So it will terminate.

# 5.14

## (a)

```
Pal4(S):
(1) X = S;
(2) E = true;
(3) while X isn't empty and E is true, do the following:
    (3.1) if eq(head(X), last(X)) is true, then X = all-but-last(tail(X));
    (3.2) otherwise E = false;
(4) return E;
```

## (b)

*Proof.*
*Partial Correctness:*
**Assertion 1:** X is a symbol string, X = S.
**Assertion 2:** S is palindrome if and only if X is palindrome and E is true.
**Assertion 3:** S is palindrome if and only if E is true.
**(1$\rightarrow$2):** Because X is a symbol string and X = S, then after carrying out the instruction E = true, it is obvious that S is palindrome is equivalent with X is a palindrome and E is true.
**(2$\rightarrow$3):** Because S is palindrome if and only if X is palindrome and E is true, then from checkpoint 2 to 3, namely when X is empty, it is obvious that an empty string is palindrome, so at this point X is always palindrome, so X is palindrome and E is true $\leftrightarrow$ E is true, so S is palindrome if and only if E is true.
**(2$\rightarrow$2):** Since S is palindrome if and only if X is palindrome and E is true, then **(i)** if head(X) = last(X) and after X = all-but-last(tail(X)), X is palindrome $\leftrightarrow$ head(X) = last(X) and all-but-last(tail(X)) is palindrome, so S is palindrome $\leftrightarrow$ X is palindrome and E is true $\leftrightarrow$ head(X) = last(X) and all-but-last(tail(X)) is palindrome and E is true. **(ii)** if head(X) $\neq$ last(X) and after E = false, then X is apparently not palindrome, neither is S, and E is false, so we still have S is palindrome if and only if X is palindrome and E is true.
*Termination:*
If eq(head(X), last(X)), then X = all-but-last(tail(X)), so then length of X will be minused by one, or if eq(head(X), last(X)) is false, then E is false. So with the process of the algorithm, either length of X will be decreased to 0 or E is assigned to be false, and both these situations make the algorithm escape the loop and terminate. So it will terminate.
Thus, Pal4 is totally correct.

## (c)

For a string with $n$ symbols, if done with Pal1, needs at most $2n + 5n = 7n$ basic opertions, while if done with Pal4, only needs at most $4 \cdot [\frac{n}{2}]$ basic operations, so it is more efficient than Pal1.