Lect 07

- finish reverse
- sorting

```
fun append(l1, l2) =
    case l1 of
      [] => l2
    | x::xs => x :: append(xs, l2)
```

k constant

independent of n

$W_{append}(n)$ = recurrence

$= \cancel{2} + W_{append}(n-1)$

$\cancel{b_0}\ 1\ \cancel{b_1}\ 1$

$W_{append}(n) = \cancel{2}n + \cancel{2}^{\cancel{b_2}\ 1}$ (closed form)

length of l_1 is $O(n)$

```
fun reverse(l: int list): int list =
    case l of
        []  => []
      | x::xs => append(reverse xs, [x])
```

code

recursion

$W_{rev}(n) = k_1 + W_{rev}(n-1) + W_{append}(n-1)$

n: length of l

length xs

length of reverse xs

$W_{rev}(0) = k_0$ 1

$W_{rev}(n) = k_1 + W_{rev}(n-1) + W_{append}(n-1)$

length xs

recursion

length of l

length of reverse xs

$W_{rev}(0) = k_0 \; 1$

$W_{rev}(n) = 1 + W_{rev}(n-1) + kn$

$= n + W_{rev}(n-1)$

↳ cleaned up recurrence

$$W_{rec}(n) = n + W_{rec}(n-1)$$

recurrence

$$= n + (n-1) + W_{rec}(n-2)$$

$$= n + (n-1) + (n-2) + W_{rec}(n-3)$$

$$\vdots$$

closed form

$$= n + (n-1) + (n-2) + (n-3) + (n-4) + \cdots + 3 + 2 + 1 + (1)$$

$$= \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} + (1)$$

$n^2 + n \leq 2n^2$ for some n

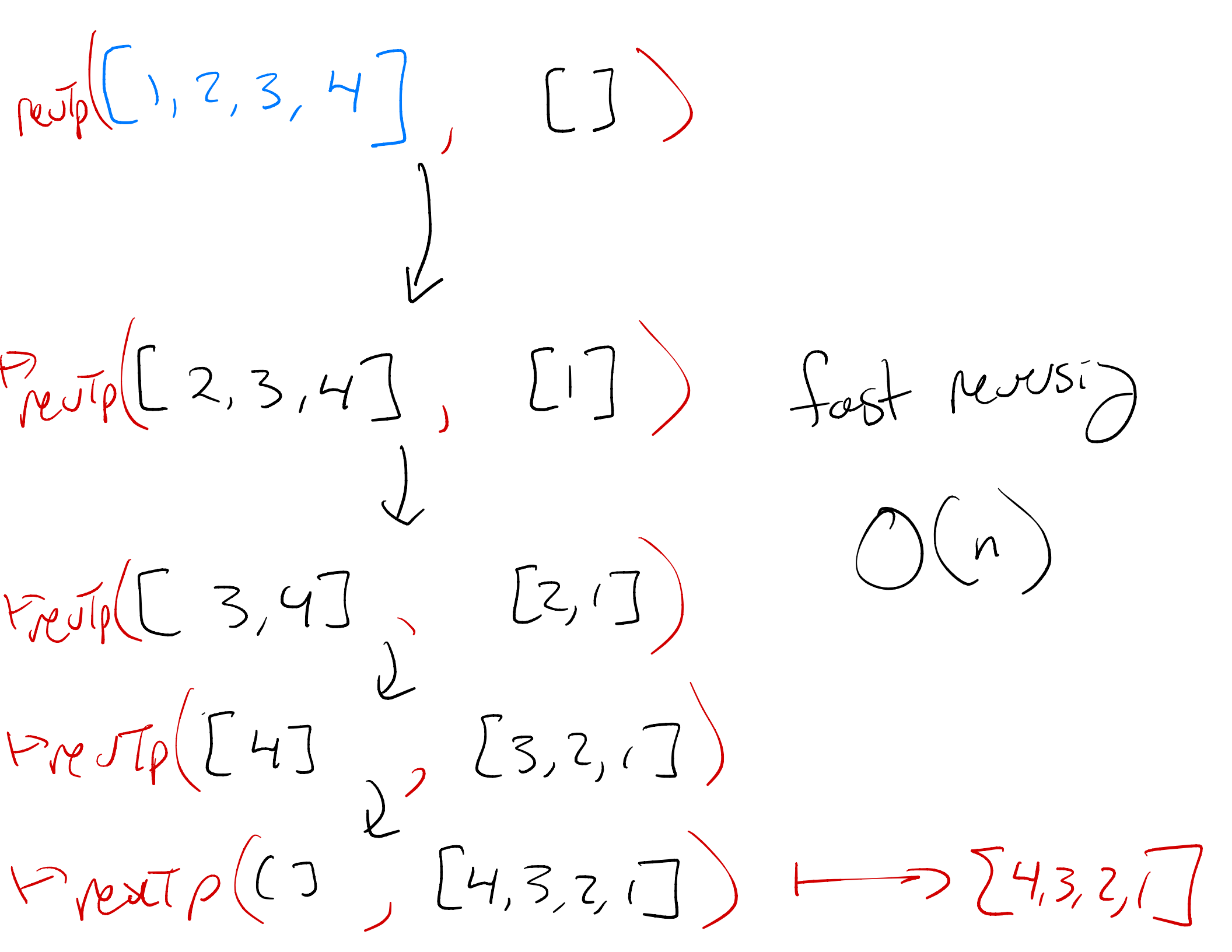
closed form

$$W_{rev}(n) = \frac{n^2}{2} + \frac{n}{2} + 1$$

biggest summand

↓

big-O $W_{rev}(n)$ is $O(n^2)$


revLp([1,2,3,4], [])
revLp([2,3,4], [1])
revLp([3,4], [2,1])
revLp([4], [3,2,1])
revLp([], [4,3,2,1])
fast reverse
O(n)
[4,3,2,1]

```
fun fastRev(l: int list) : int list =
    revTwoPiles(l, [])
```

$W_{fastRev}(n) = 1 + \overbrace{W_{revTwoPiles}(n)}^{n}$ is $O(n)$

```
fun revTwoPiles(l:intlist, r:intlist): int list =
  case l of
      [] => r
    | x::xs => revTwoPiles(xs, x::r)
```

is $O(n)$

$W_{revTwoPiles}(\underbrace{n}_{\text{length of } l}) = 1 + W_{revTwoPiles}(n-1)$

Use extra memory (* in input *) to

Save time

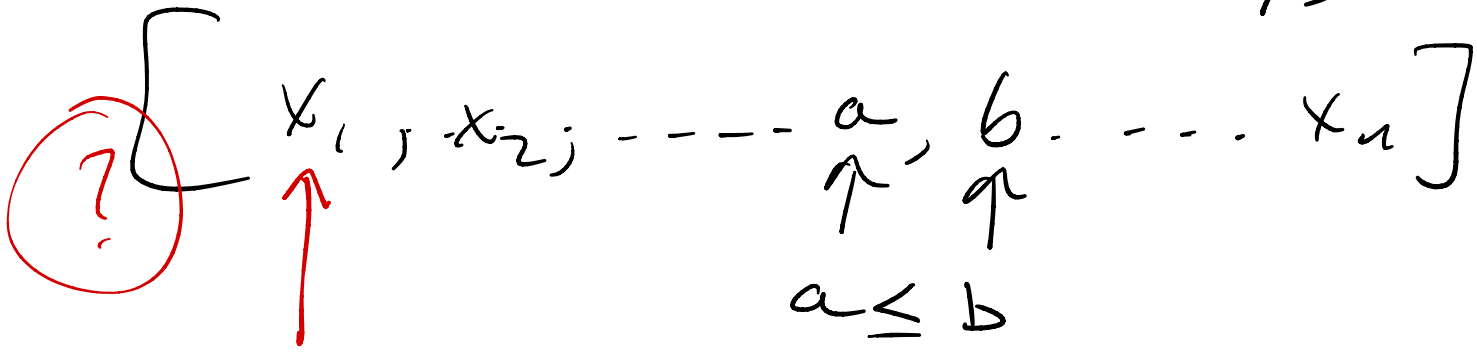
Sorting

[4, 1, 3, 2] $\xrightarrow{\text{sort}}$ [1, 2, 3, 4]

Put numbers in increasing order

A sorted list is:

(1) All numbers in the list are equal to or greater than the number to the left of them (if any)



Equivalent: A list is sorted iff

either (a) it's []

or (b) it's x::xs where

xs sorted

and x ≤ everything in xs

(1,2,3) is sorted
if [2,3] sorted
1 ≤ [2,3] ✓

- [3] sorted
- 2 ≤ [3] ✓

[] sorted
3 ≤ [] ✓

Equivalent: A list is sorted iff

either (a) it's []

(b) it's [x]

or (c) it's x::y::xs where

y::xs sorted

and $x \leq y$

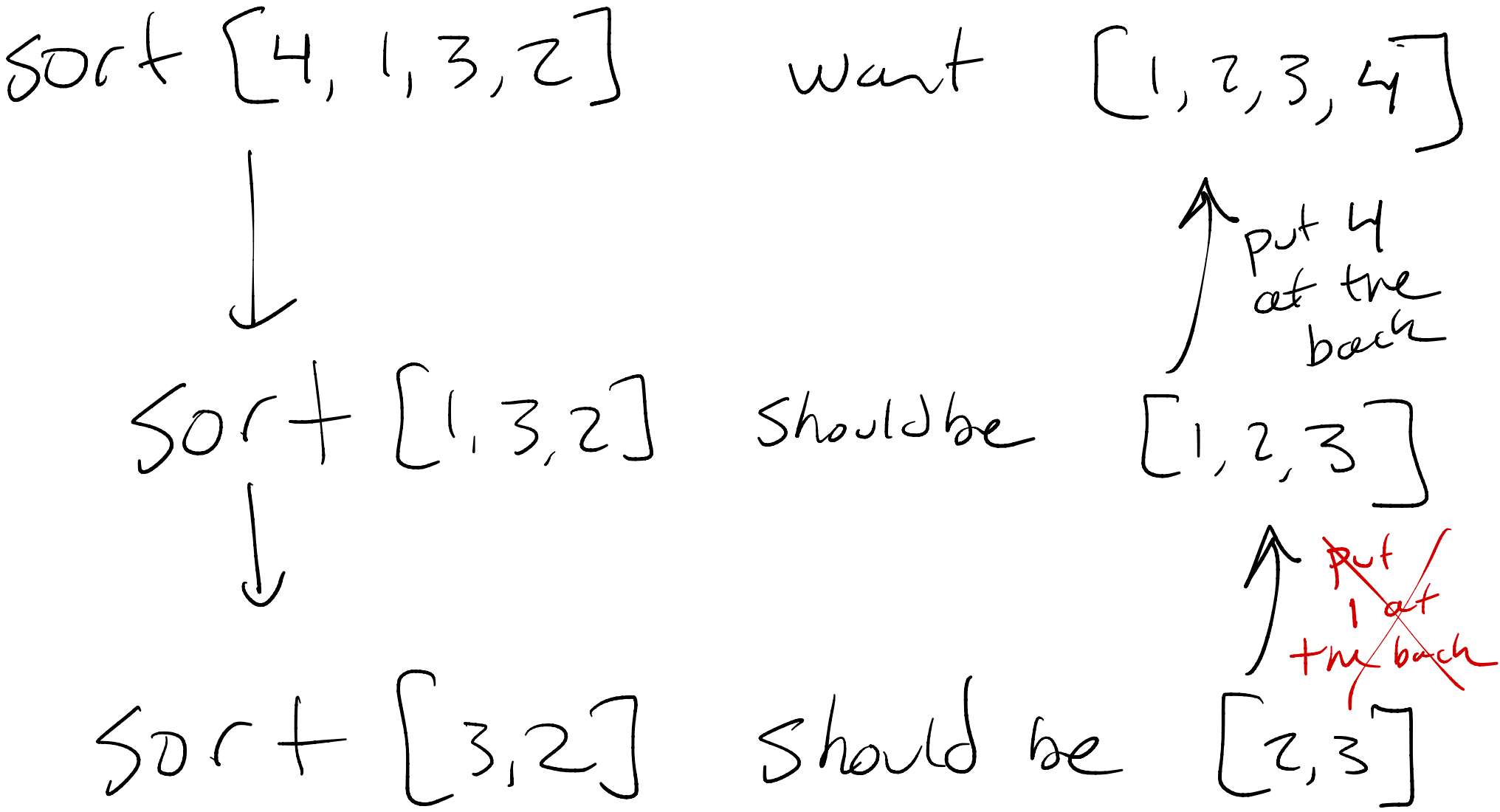
(* Specification: For all lists l,
Purpose: sort(l) is a sorted permutation of l *)

↳ all same numbers, possibly in a different order

```
fun isort (l : int list) : int list =
  case l of
    [] => []
  | x::xs => insert (x, isort(xs))
```

isort(xs) is sorted

Insertion sort


sort [4, 1, 3, 2]
want [1, 2, 3, 4]
put 4 at the back
sort [1, 3, 2]
should be [1, 2, 3]
put 1 at the back
sort [3, 2]
should be [2, 3]

(* Spec: for any num x and (sorted int list l)
insert(x, l) is a sorted permutation of x::l *)

```
fun insert (x:int, l: int list): int list =
    case l of
      [] => [x]
    | y::ys => case x ≤ y of
                 true => x::y::ys
               | false => y::insert(x, ys)
```

Def

```
fun isort (l : int list) : int list =
  case l of
    [] => []
  | x::xs => insert (x, isort(xs))
```

is sorted

$$W_{isort}(n) = 1 + W_{isort}(n-1) + W_{insert}(n-1)$$

with of l

$$\leq 1 + W_{isort}(n-1) + O(n)$$

is $O(n^2)$

```
fun insert (x:int, l:int list):int list =
  case l of
    [] => [x]
  | y::ys => case x <= y of
               true => x::y::ys
             | false => y::insert(x, ys)
```

Def

$W_{insert}(n) = \begin{cases} 4 \\ \text{or} \\ 4 + W_{insert}(n-1) \end{cases}$

$n = \text{length of } l$

$\text{is } O(n)$

worst case

$\leq 4 + W_{insert}(n-1)$

- insert(1, [2,3,4]) constant
- insert(5, [2,3,4]) time proportional to length of list

Worst case is $O(n)$

isort([1, 2, 3, 4]) fast $O(n)$

isort([4, 3, 2, 1]) slow $O(n^2)$