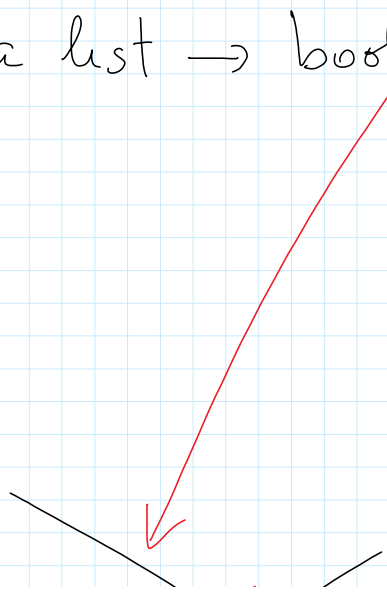
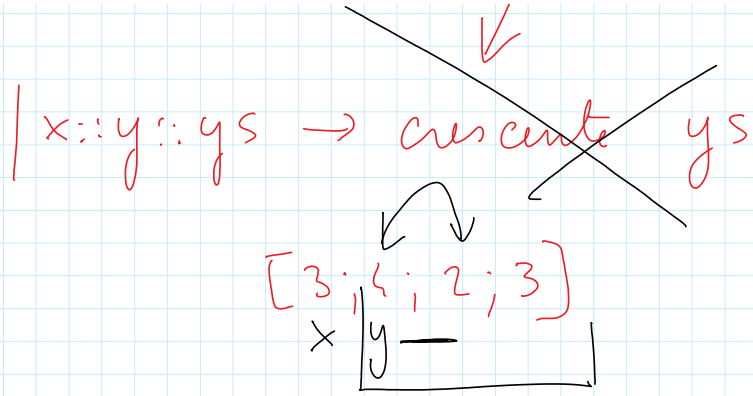
Una lista ordinata

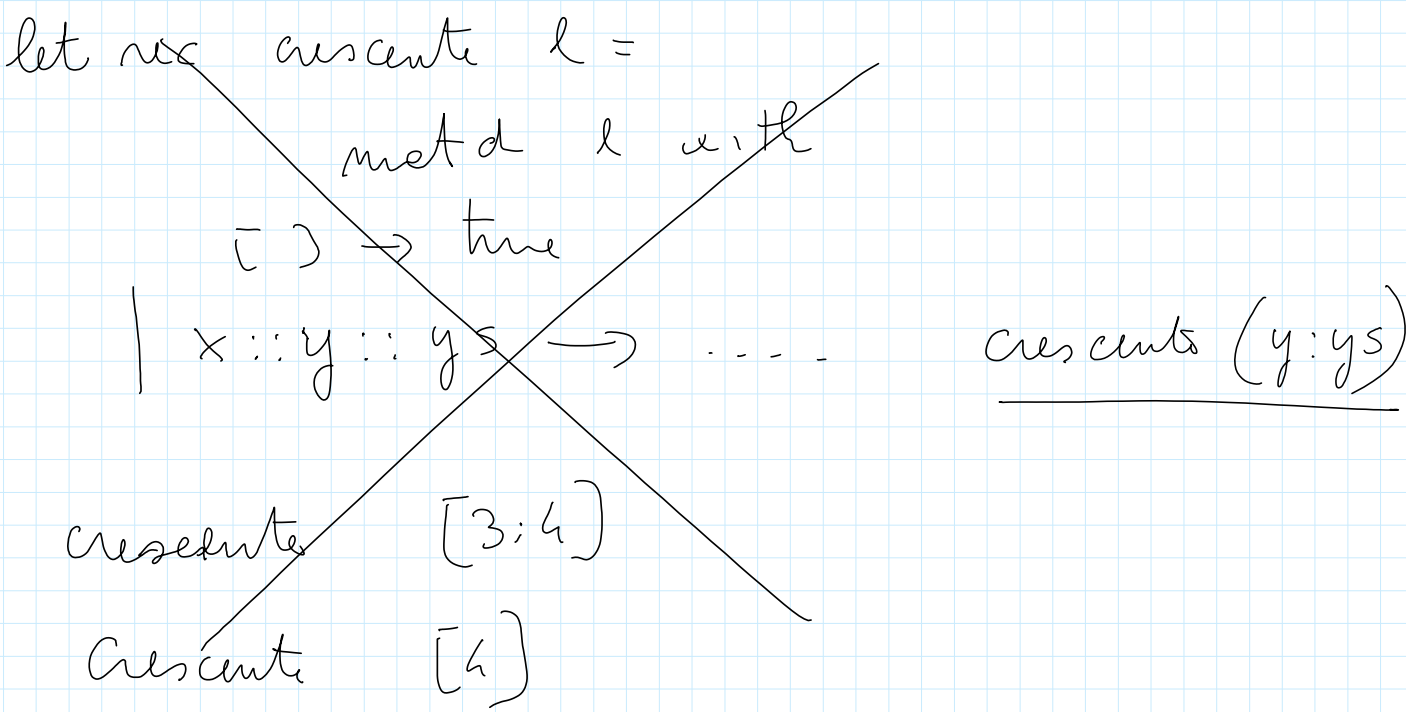
- crescente [1;3;5;10]
  ogni elemento è minore del successivo
- non decrescente [1;3;3;5;10]
  ogni elemento è minore o uguale al successivo

Funzione CAML che ci dice se una lista è ordinata in senso crescente

```
# let rec crescente l =
    match l with
    [] -> true
  | [x] -> true
  | x::y::ys ->
        if x >= y then false
        else crescente (y::ys) ;;
crescente : 'a list -> bool = <fun>
```




| x::y::ys → crescente ys
[3;4;2;3]
x
y


let rec crescente l =
match l with
[] → true
| x::y::ys → ....
crescente (y:ys)
crescente [3;4]
Crescente [4]

$[3;2;-2;10] \longrightarrow [-2;2;3;10]$

Ordinamento di una lista (sorting)

Insertion sort

Se la lista ha almeno un elemento $(x :: xs)$ si ordina ricorsivamente xs e alla lista risultante si aggiunge x in posizione corretta

$[3;-2;1]$

$3 \quad [-2;1] \longrightarrow [-2;1;3]$

$-2 \quad [1] \longrightarrow [-2;1]$

$1 \quad [] \longrightarrow [1]$

giovedì 28 novembre 2019 16:35

```
#let rec insord el l =
    match l with
      [] -> [el]
    | x::xs ->
        if el <= x then el::x::xs
        else x::insord el xs;;
insord : 'a -> 'a list -> 'a list = <fun>
```

Si suppone che l sia ordinata in senso non decrescente.

[-2; 3; 10; 3] ordino in senso non decrescente ottengo

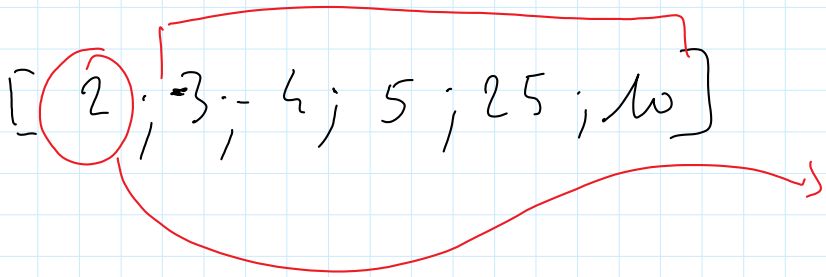
[-2; 3; 3; 10]

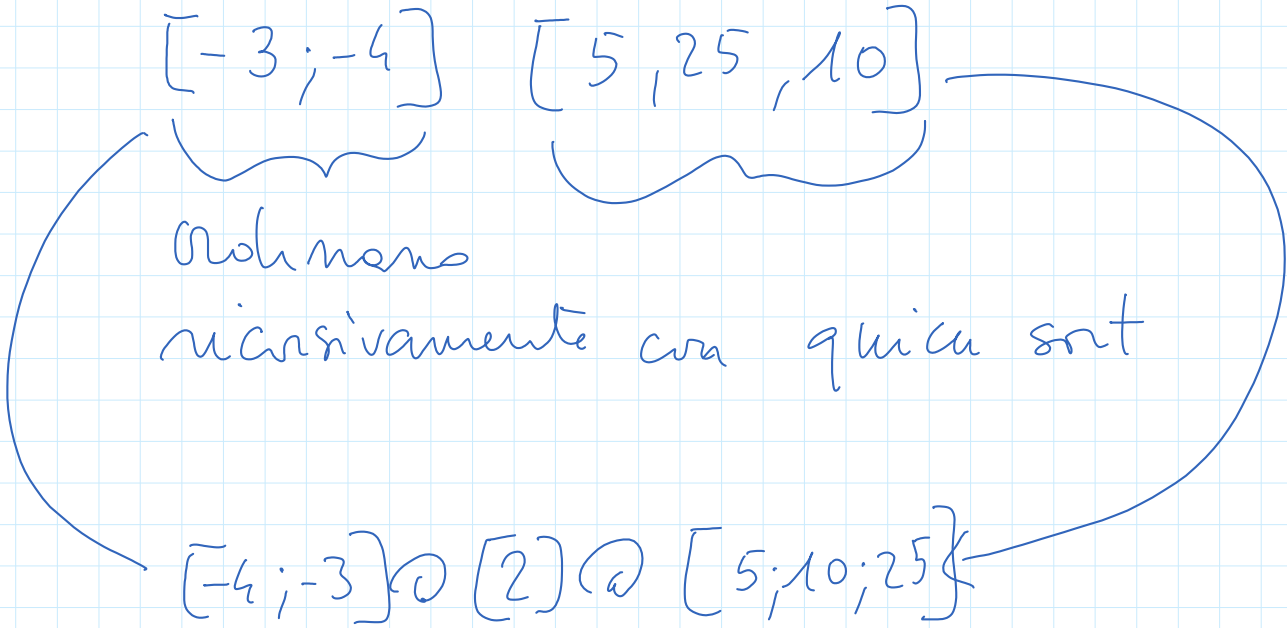
Non è possibile ordinarla in senso crescente.

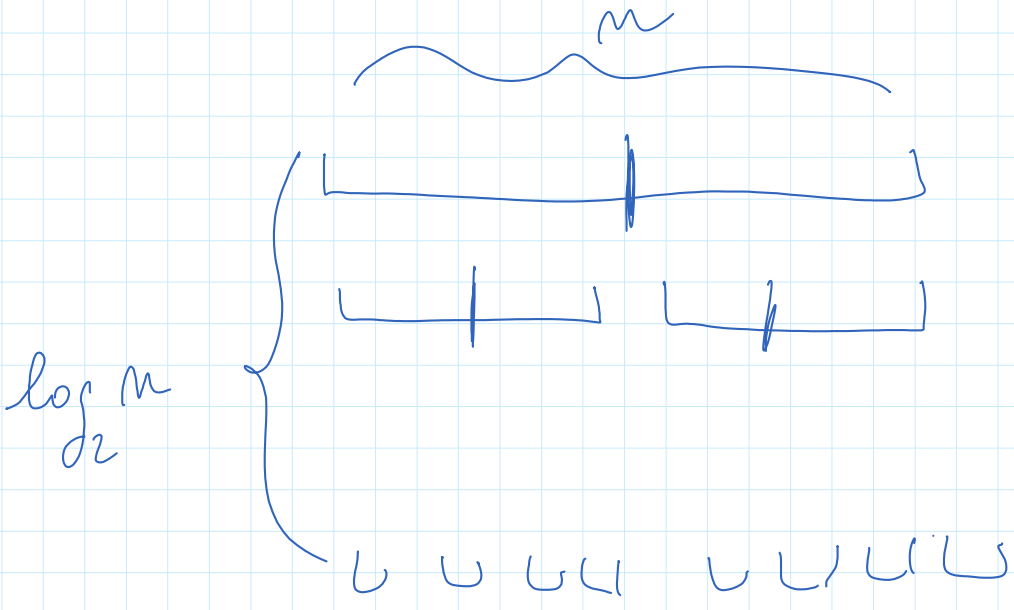
giovedì 28 novembre 2019 16:41

```
# let rec inssort l =
    match l with
      [] -> []
    | x::xs -> insord x (inssort xs) ;;
inssort : 'a list -> 'a list = <fun>
```

QUICK SORT


[2;-3;-4;5;25;10]


[-3;-4]
[5,25,10]
richiamo
ricorsivamente con quick sort
[-4;-3] [2] [5;10;25]


n
$\log_2 n$

giovedì 28 novembre 2019 16:49

```
# let rec split l n =
    match l with
    [] -> ([], [])
  | [x] -> if x <= n then ([x], [])
           else ([], [x])
  | x::y::ys -> let (l1, l2) = split (y::ys) n
                in
                if x <= n then (x::l1, l2)
                else (l1, x::l2) ;;
```

Ok!

```
  | x::y::ys -> let (l1, l2) = split ys
                in
```

Ma vengono tanti casi!

Soluzione ma x che y

giovedì 28 novembre 2019 16:57

```
# let rec quicksort l =
    match l with
      [] -> []
    | [x] -> [x]
    | x::y::ys -> let (l1,l2) = split (y::ys) x
```

in

```
quicksort l1 @ [x] @
quicksort l2 ;;
```

≡

```
quicksort l1 @ ( x :: quicksort l2 )
```

```
quicksort [2;3;-1;4;∅]
= ([-1;∅], [3;4]) = split [3;-1;4;∅] 2
```

quicksort [-1;∅] @ [2] @ quicksort [3;4] = [-1;0;2;3;4]

quicksort [-1; ∅)

([], [∅]) = split [∅] -1

[] ⊕ [-1] ⊕ [∅]

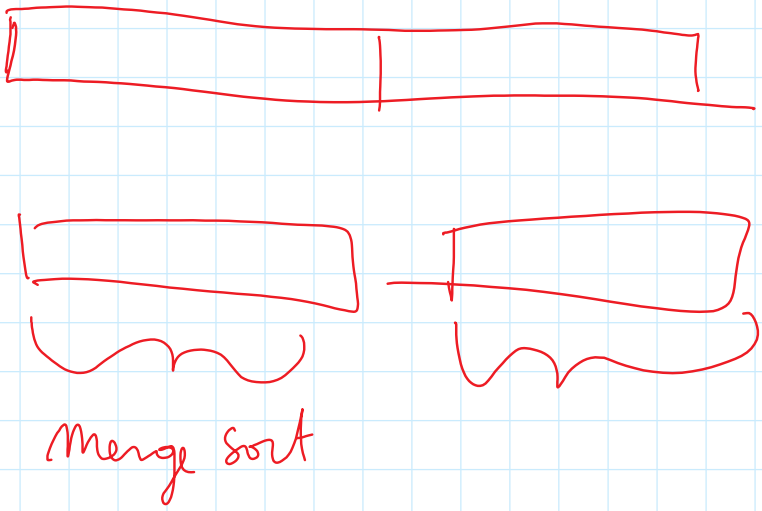
quicksort [3; 4)

([], [4]) = split [4] 3

[] ⊕ [3] ⊕ [4]

MERGE SORT

ordinamento per fusione



[; . . ;]

[. . . . 15)

[0; 1; 2; 3; 3; 5; 7; 10; 15]

```
# let rec split l =
        match l with
    [] -> ([], [])
  | [x] -> ([x], [])
  | x::y::ys -> let (l1, l2) = split ys
                in (x::l1, y::l2) ;;
split : 'a list -> 'a list * 'a list = <fun>
```

```
# let rec merge l1 l2 =
        match (l1, l2) with
    ([], l2) -> l2
  | (l1, []) when l1 <> [] -> l1
  | (x::xs, y::ys) ->
        if x < y then x :: merge xs (y::ys)
          else y :: merge (x::xs) ys ;;
merge : 'a list -> 'a list -> 'a list = <fun>
```

```
# let rec mergesort l =
    match l with
    [] -> []
  | [x] -> [x]
  | x::y::ys -> let (l1,l2) = split (x::y::ys)
                in
                merge (mergesort l1) (mergesort l2) ;;
```

oh!

split l

prec: int list → int → int list * int

che data una lista di interi, l, e un intero, n, restituisce una coppia (l1, m) tale che

l1 contiene tutti i valori di l che precedono la prima occorrenza di n, e m è il numero di valori di l1 che sono maggiori di 0.

ASS. esiste in l almeno una occorrenza di n.

prec [-2; 3; 4; 6; -3; 8] 6 =

([-2; 3; 4], 2)

```
# let rec prec l n =
    match l with
    [] -> ([], 0)
  | x::xs -> if x = n then ([], 0)
             else let (l1, m) = prec xs n
             in if x > 0 then (x::l1, m+1)
                else (x::l1, m) ;;
```

giovedì 28 novembre 2019 17:34

prec : 'a list → 'a list

che data una lista, restituisca la lista degli elementi che sono immediatamente preceduti da un elemento minore

prec [3;4;5;2;-1;7;3;8] = [4;5;7;8]

```
let rec prec l =
        match l with
  [] -> []
| [x] -> []
| x::y::ys ->
      if x<y then y::prec (y::ys)
      else prec (y::ys);;
```

len : int list → bool

controlla che ogni elemento, escluso l'ultimo, sia minore della somma di tutti gli elementi che seguono.

len [1; -2; 2; 3; -10; 14] = true

len [1; 12; 3; 4; -10; 14] = false

```
let rec sum l =
        match l with
          [] -> 0
        | x::xs -> x + sum xs ;;
```

```
let rec len l =
        match l with
          [] -> true
        | [x] -> true
        | x::y::ys ->
            if x < sum (y::ys)
```

```
if x < sum (y::ys)
    then less (y::ys)
    else false;;
```