

1) Fibonacci P est un nombre premier

let rec fib n p = match n with

$0 \rightarrow 0$

$| 1 \rightarrow 1$

$| n \rightarrow fib(n-1)P + fib(n-2)P ;;$

$$u_0 = 0, u_1 = 1, \quad u_n = u_{n-1} + u_{n-2} \pmod{P}$$

$P \backslash n$	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
2	0	1	1	0	1	1	0	1	1
3	0	1	1	2	0	2	2	1	0
5	0	1	1	2	3	0	3	3	1

transformer en algo itératif.

let fibo n p =

let $u = \text{ref } 0$ and $uv = \text{ref } 1$ in

for $k=2$ to n do $uv := (!u + !uv; u := !uv - !u)$
done;

$!uv$;
mod p)

la complexité temporelle est $O(n)$. : raisonnable.

en sachant $u_0 = a, u_1 = b$
revient à u_{n-1} sachant
 $u_0 = b, u_1 = a+b$

transformer en algo récursif linéaire

let fibo n p =

let rec fibo-aux n a b = match n with

* while uv sachant que $u_0 = a, u_1 = b$ r
| $n \rightarrow fibo_aux (n-1) b ((a+b) \xrightarrow{0} a \xrightarrow{1} b \xrightarrow{n \bmod p})$
in $fibo_aux n 0 1$;;

(la ft récursive auxiliaire

nécessite 2 variables aux.
et b en plus de n)

\Rightarrow comp. $O(n)$

Algorithmes de tri

2 classes de complexité principales:

- algos en $O(n^2)$

- algos en $O(n \log n)$: les meilleurs.

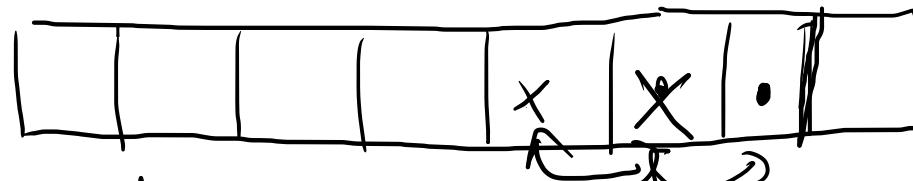
On peut raffiner selon qu'on parle de complexité dans le pire des cas ou en moyenne.

Auj. on étudie ceux en $O(n^2)$.

Le tri bulle

(Bubble sort)

Principe de l'algo :



1^{re} étape: on parcourt le tableau et on échange le contenu des cases i et $i+1$ si $t[i] > t[i+1]$. A la fin de cette 1^{re} étape, le maximum est relégué en dernière position

étapes suivantes : on recommence en ignorant la dernière case, puis les 2 dernières cases et ainsi de suite jusqu'à travailler uniquement sur les 2 premières cases.

ex : [5 8 1 3 2 7] \rightarrow 5 1 3 2 7 8

5 1 3 2 7 8
 → 1 3 2 5 7 8 phase 2

→ 1 2 3 5 7 8 phase 3 → Pas de changement phase 4

On peut remarquer que s'il n'y a pas eu de changement lors d'une phase alors c'est que le tableau est trié donc on peut s'arrêter.

code pour un tableau

```

let un passage (v, b) =
  let n = Array.length v in
  b := true;
  for h = 0 to n-2 do
    if v.(h) > v.(h+1) then (let aux = v.(h) in
      v.(h) ← v.(h+1); v.(h+1) ← aux; b := false)
    done;
  
```

à array * bool ref → unit

let tri_bubble_tableau $v = \left(\text{type: } 'a \text{ array} \rightarrow 'a \text{ array} \right)$
let $b = \text{ref false}$ in
while $\boxed{!b = \text{false}}$ do $\text{un_passage}(v, b)$ done;
 $v_{ij} = \text{not}(!b)$

Codage pour une liste

let rec passage l bo = match l with
 [] \rightarrow [], bo

| [a] \rightarrow [a], bo

| a :: b :: q \rightarrow if $a <= b$ then let (l_1 , rep) = passage (b :: q) bo
 in (a :: l_1 , rep)

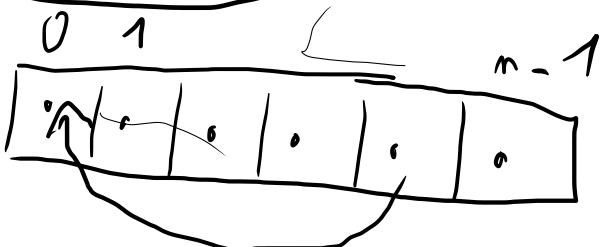
else let (l_1 , rep) = passage (a :: q) bo
 in (b :: l_1 , false) ;;

let tri_bulle l =

let (s, verdict) = passage l true in
if verdict then s else tri_bulle s ;;

Tri sélection

Principe:



chercher le minimum, le placer en tête,
recommencer avec la zone du tableau
entre 1 et $n-1$

Complexité:

- dans tableau de longueur k , la recherche du min nécessite $k-1$ comparaisons
- on effectue ça pour le variant de m à 1
au final le n^{e} de comparaisons est $\sum_{k=1}^m (k-1) = \frac{m(m-1)}{2}$

$\Theta(m^2)$

en moyenne ou dans le pire des cas, c'est pareil

Codage avec un tableau

$i \leq n-1$

on recherche l'indice k compris entre i et $n-1$ tq

$t[k] = \min_{i \leq j \leq n-1} t[j]$, on échange les cases i et k .

et on fait varier i de 0 à $n-2$.

let tri_selection $t =$ let $n = \text{Array.length } t$ in
for $i = 0$ to $n-2$ do
 let $b = \text{ref } i \text{ in}$
 for $j = i+1$ to $n-1$ do
 if $t(j) < t(!b)$ then $b := j$
 done;
 let temp = $t(i)$ in $t(i) \leftarrow t(!b); t(!b) \leftarrow \text{temp}$
 done;
 $i := i + 1$

Codeage avec une liste

fonction auxiliaire : liste \rightarrow (min de la liste, liste privée de ce min)

let rec min_reste = function [] \rightarrow failwith "erreur"

| [a] \rightarrow a, []

| a::l \rightarrow let (b,s) = min_reste l in
if a < b then (a,b::s) else (b,a::s);;

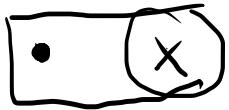
let rec tri_selection = function

[] \rightarrow []

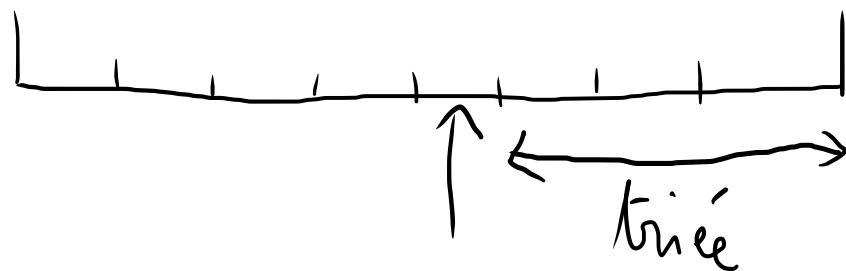
| l \rightarrow let (x,t) = min_reste l in x :: (tri_selection t);;

Complexité = $\Theta(n^2)$

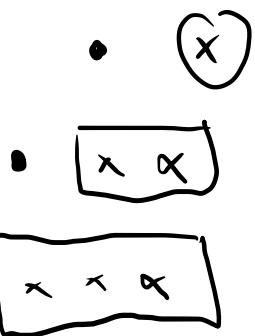
TRI INSERTION



Principe :



Pour i variant de $m-2$ à 0 (on insère l'elt d'indice i dans la zone déjà triée des indices
affinant de $i+1$ à $m-1$.



on commence par le
croisage sur les listes

fonction auxiliaire: inserer : 'a → 'a list → 'a list

inserer x l = renvoie une liste triée formée de & à laquelle on a rajouté x , sachant qu'en

let rec inserer x l = match l with

compléxité $O(\text{len}(l))$

| $b :: q \rightarrow \text{if } x <= b \text{ then } x :: l \text{ else } b :: (\text{inserer } x \text{ } q)$;;

let rec tri_inserion = function

| $a :: l \rightarrow \text{inserer } a (\text{tri_inserion } l)$;;

compléxité en fait de $m = \text{longueur(liste)}$

$$T(m) = T(m-1) + O(m) \quad \text{(on se rejoue en } \underline{T(m) = O(m^2)})$$

Algo en n^2

Codage pour un tableau

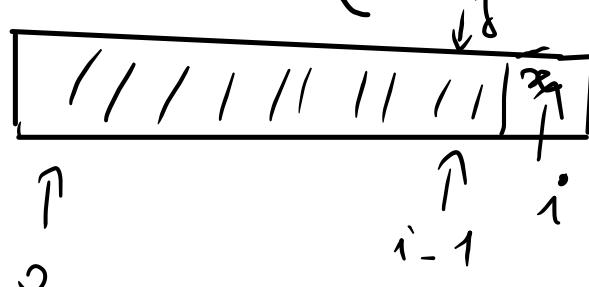
on le fait dans l'autre sens.

$i = 4$, $j = 3$
 $j = 2$
 $j = 1$

10	20	30	40	40
10	20	25	30	40

[on suppose que la zone $t[0] \dots t[i-1]$ est déjà triée. On va insérer $t[i]$ dans la zone entre les indices 0 et i.]

Pour i variant de 1 à $n-1$.
dans la boucle for, la zone entre 0 et i est triée.
let tri_insertion t :



10	20	30	40	40	25
0	1	2	3	4	

Invariant de boucle : après passage de i
la zone entre 0 et i est triée.
let m = Array.length t in
for i = 1 to $m-1$ do
let $x = t[i]$ and $j = \text{ref}(i-1)$ in
while $\text{!}j >= 0$ $\&\&$ $t[\text{!}j] > x$ do
 $t[\text{!}j+1] \leftarrow t[\text{!}j]$; decr j
done;
done;
 $t[\text{!}j+1] \leftarrow x$ $t[i];$