Correction TD1

Le but de ce TD est surtout que le raisonnement algorithmique se mette en place.

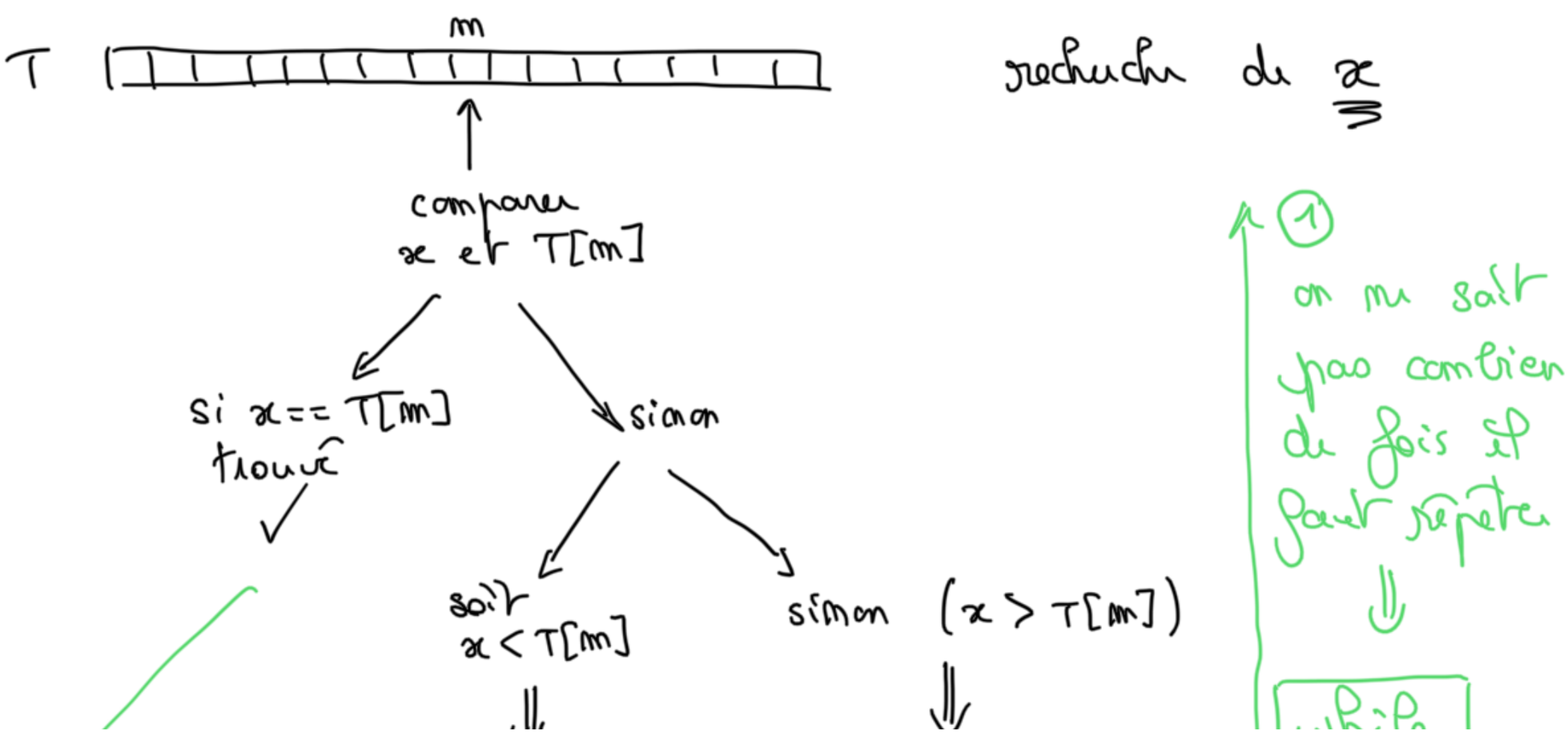
Donc :

- bien montrer comment on construit progressivement l'algo, le stockage, les variables ...

⚠ Un for arrêté par un break ou return ne sera pas toléré.

Ex. 1

Pour la recherche dichotomique ds 1 tableau trié :



chercher à gauche

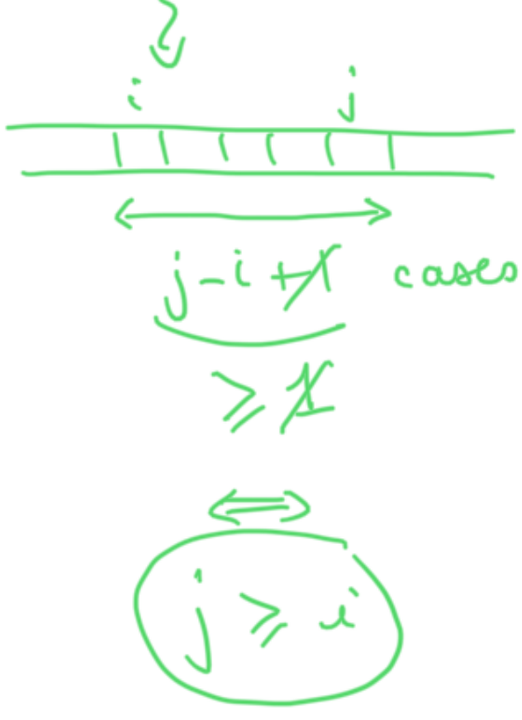
chercher à droite

il faut pouvoir localiser des sous-tableaux de T

⇓

désignés par les indices de début / fin de la portion (inclus)

① on va exécuter cela tant que la portion contient au moins une case



D'où le pseudo code :

Variables

T : tableau de n cases
i, j : indices début / fin de la portion
m : indice du milieu
x : élément cherché
res : indice de la case du x (ou -1 si pas trouvé)

Code

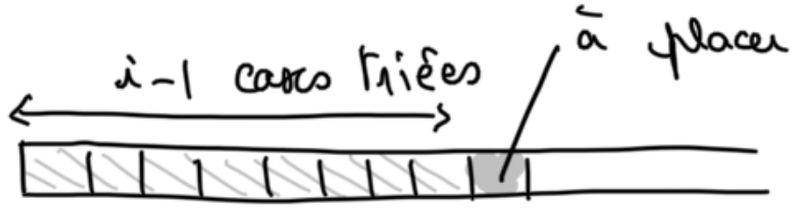
```
i ← 0
j ← n-1
res ← -1
tantque ( (j ≥ i) et (res == -1))
{   m ← ⌊ (i+j) / 2 ⌋

    si (T[m] == x)
       res ← m
    sinon si (x < T[m])
       j ← m-1
    sinon // x > T[m]
       i ← m+1
    fsi
ftq
```

façon d'arrêter la boucle quand on trouve

ii) Dans cette version, n'importe quelle occurrence est renvoyée (ni la 1ère, ni la dernière).

Ex. 2 Tri par insertion

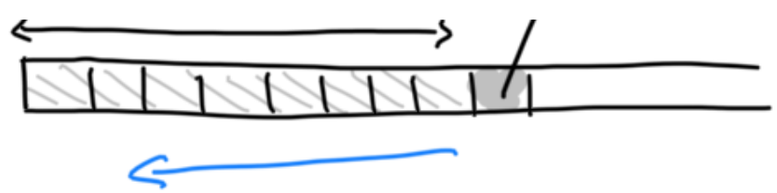


Principe :

- → à l'étape i, les i-1 premières cases ont été mises en ordre
- → la i ème est insérée à la bonne place un peu comme dans un jeu de taquin

tous les éléments doivent trouver leur place ⟹ boucle for

Mise en place de l'algo ... i-1 cases triées /*



pour i de ... à ... (sens ←)

- on parcourt les cases de droite à gauche) → variable j
- si T[j] > x alors il faut le décaler vers la droite pour faire de la place à x à gauche

D'où le pseudo-code

Variables

T : tableau de n cases
i, j : indices
x : élément à insérer

Code

2ème case

pour i de 1 à n-1 (un tableau à 1 seul élément est bien trié)
  x ← T[i]
  j ← i-1
  tant que ((T[j] > x) && (j > 0))

```
    tant que ((i[j] > x) && (j >= 0))
    | T[j+1] ← T[j]
    ftq
    T[j+1] ← x
fpour
```

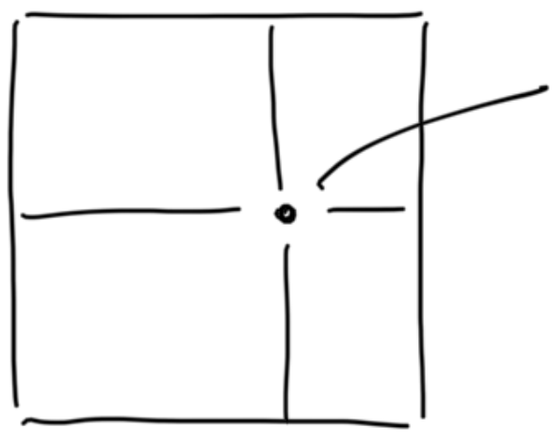
← prendre le temps de mettre en place la condition. Montrer que les ⩾, > et val. s'ajustent par des tests ...

Complexité : $O(n^2)$

→ meilleur cas :
tableau trié ⇒ $O(n)$

→ pire cas :
trié à l'envers

Ex. 3 Problème des min et max



trouver les cases qui sont
→ min de leur ligne
→ max de leur colonne

Les gars chercher ...
ils vont forcément tomber sur quelque chose du genre :

→ pour chaque ligne
→ calculer le min
→ puis reparcourir la ligne
→ pour chaque occurence du min
→ vérifier si c'est un max de la colonne

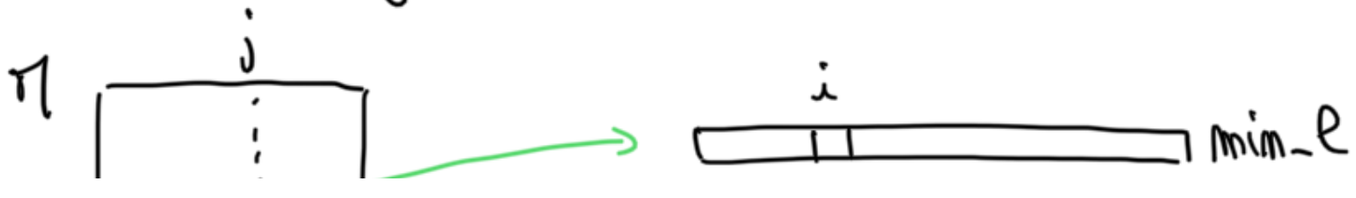
Estimer la complexité : $O(n^3)$!

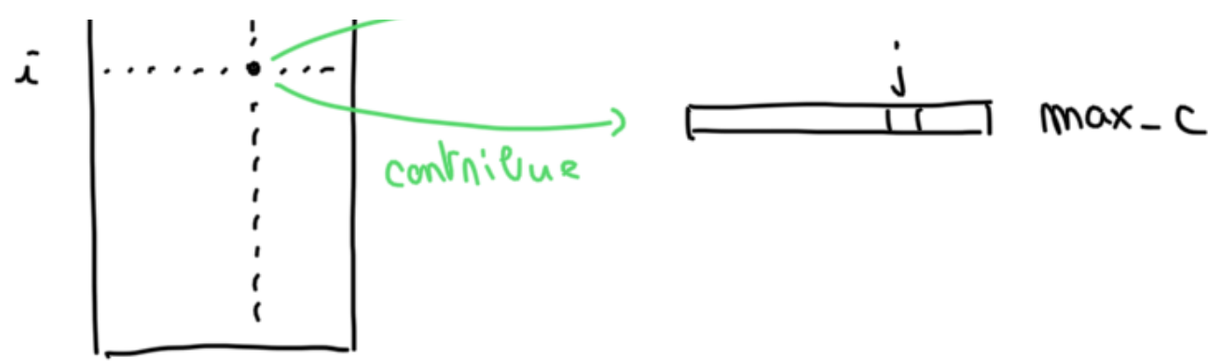
A faire assimiler :

Pour gagner en temps, il faut sacrifier de l'espace mémoire

On va créer 2 tableaux annexes pour calculer et stocker les min des lignes / max des colonnes.

1) ces tableaux peuvent être calculés simultanément, en parcourant une seule fois la matrice :





2) une fois ces tableaux remplis, les min-max pour simplement les cases dont la valeur est égale au min de la ligne et max de la colonne

D'où le pseudo-code :

Variables

M : matrice de taille n x m
min_l,
max_c : tableaux annexes
i, j : variable indices

Code

// Initialisation des tableaux annexes
pour i de 0 à n-1
    min_l[i] ← M[i][0]

```
fpour
pour j de 0 à m-1
  | max_c[j] ← M[0][j]
fpour

// Calcul des tableaux annexes
pour i de 0 à n-1
   pour j de 0 à m-1
      si (M[i][j] < min_l[i])
      | min_l[i] ← M[i][j]
      fsi
      si (M[i][j] > max_c[j])
      | max_c[j] ← M[i][j]
      fsi
   fpour
fpour

// Extraction des min-max
pour i de 0 à n-1
```

```
pour i de 0 à m-1
    si ((M[i][j] == min_l[i]) &&
        (M[i][j] == max_c[j]))
    | afficher (i,j)
    fsi
  fpour
fpour
```

Complexité :
on n'est plus qu'en $O(m^2)$

Ex. 4

La médiane d'un ensemble d'éléments est la valeur telle qu'il y ait autant d'éléments inférieurs que supérieurs.
Pour la calculer, le plus simple est de trier le tableau
$\Rightarrow$ c'est alors l'élément du milieu.

Pour économiser un peu de temps, si les $n/2$ premiers éléments sont triés, alors le $\frac{n}{2}$ème est la médiane.
Pour ce tri partiel, on peut utiliser la sélection / échange :
on ne trie que les $n/2$ premiers éléments _

Variables

T : tableau de n éléments
i, j, m, imin : indices entiers
tmp : variable contenant la valeur du min actuel

Code

$m \leftarrow \left\lceil \frac{i+j}{2} \right\rceil$

pour i de 0 à m
    tmp ← T[i]
    imin ← i
    pour j de i+1 à n-1
        si (T[j] < tmp)
        | imin ← j
        | tmp ← T[imin]
        fsi
    fpour
    T[imin] ⟷ T[i]
fpour
retourner T[m]

Complexité : $O\left(\frac{n^2}{2}\right)$

on ne gagne qu'un facteur 1/2, pas un ordre de grandeur ...