

EA4 – Éléments d’algorithmique

TD n° 4 : dichotomie et tris

Exercice 1 : occurrences et dichotomie

- Écrire une fonction `occurrencesNaif(T, x)` qui renvoie le nombre d'*occurrences* (apparitions) d'un élément `x` dans un tableau `T`. Quelle est sa complexité dans le pire cas pour un tableau de taille n ?

On suppose maintenant que le paramètre `T` est un *tableau trié*.

- Que peut-on dire des occurrences d'un élément `x` dans `T` ?
- Écrire une fonction `trouvePremier(T, x, begin=0, end=None)`, *la plus efficace possible*, qui renvoie `None` si `x` n'apparaît pas dans `T` et sa première position dans le tableau sinon. Quelle est la complexité dans le pire cas pour un tableau de taille n ?
- En déduire une fonction `occurrencesDicho(T, x)` qui renvoie le nombre d'occurrences de `x` dans `T` *le plus efficacement possible*. Quelle est la complexité de `occurrencesDicho(T, x)` dans le pire cas, pour un tableau de longueur n ?

Exercice 2 : quelques variantes du tri par insertion

On considère les deux descriptions suivantes du tri par insertion dans un tableau, où l'insertion est réalisée par échanges successifs :

```
def triInsertionParLaGauche(T) :  
    for i in range(1, len(T)) :  
        for j in range(i+1) :  
            if T[i] < T[j] : break  
        for k in range(j, i) :  
            T[i], T[k] = T[k], T[i]  
  
def triInsertionParLaDroite(T) :  
    for i in range(1, len(T)) :  
        for j in range(i, 0, -1) :      # pour j de i à 1 par pas de -1  
            if T[j-1] <= T[j] : break  
            T[j-1], T[j] = T[j], T[j-1]
```

- Combien chacun de ces algorithmes fait-il de comparaisons dans le pire cas ? dans le meilleur cas ? Même question pour les échanges, et pour le cumul de ces deux types d'opérations. Que peut-on en déduire sur la complexité comparée des deux algorithmes ?
- Montrer l'invariant suivant pour la boucle principale de l'une ou l'autre de ces deux versions :
« *À la fin du tour de boucle d'indice `i`, le sous-tableau `T[:i+1]` contient les mêmes éléments qu'initialement, triés en ordre croissant.* »
- Comment tirer parti de cet invariant pour diminuer le nombre de comparaisons effectuées dans le pire cas ?
- Combien d'affectations un échange nécessite-t-il ? Si l'insertion de `T[i]` se fait en position k , combien d'affectations sont donc effectuées par les versions ci-dessus ?
- Comment diminuer ce nombre d'affectations ?
- Quel est l'effet de ces améliorations sur la complexité de l'algorithme ?

Exercice 3 : montagnes (partiel 2022)

On dit qu'un tableau T de n entiers est une *montagne* s'il est constitué d'une première partie strictement croissante, suivie d'une deuxième strictement décroissante, chacune pouvant éventuellement être vide ; autrement dit, T est une montagne s'il est strictement croissant ou décroissant, ou s'il existe un certain indice $m \in [1, n - 2]$ tel que :

$$T[0] < T[1] < \dots < T[m] \quad \text{et} \quad T[m] > T[m+1] > \dots > T[n-1].$$

- Proposer un algorithme `est_une_montagne(T)` le plus efficace possible qui teste si T est une montagne. Justifier rapidement sa correction et sa complexité.

On suppose maintenant que T est une montagne.

- Proposer un algorithme `pied(T)` le plus efficace possible qui renvoie le plus petit élément de T . Justifier sa correction et sa complexité.
- Étant donné un indice i , comment tester *en temps constant* si $i < m$, où m est l'indice (inconnu *a priori*) du maximum de T ?
- En déduire un algorithme `sommet(T)` le plus efficace possible qui renvoie le plus grand élément de T . Justifier rapidement sa correction et sa complexité.
- Proposer un algorithme `nivelle(T)` le plus efficace possible qui renvoie un tableau trié contenant les mêmes éléments que T . Justifier rapidement sa correction et sa complexité.
- Justifier l'optimalité des algorithmes proposés.

Exercice 4 : tri à bulles

On considère une nouvelle méthode de tri, appelée *tri à bulles* :

```
def triBulles(T) :
    for i in range(len(T)-1, 0, -1) :
        for j in range(i) :
            if T[j] > T[j+1] : T[j], T[j+1] = T[j+1], T[j]
    return T
```

- Appliquer cette méthode de tri sur le tableau suivant : $T = [5, 1, 4, 2, -5, 7, 6]$.
- Expliquer le fonctionnement du tri à bulles. Exhiber l'invariant qui montre que l'algorithme renvoie bien une version triée de T .
- Combien d'opérations élémentaires sont effectués pour trier un tableau de taille n avec cet algorithme ?
- Montrer que si aucun échange n'est fait à l'étape i , l'algorithme peut être arrêté.
- Montrer que si, à l'étape i , aucun échange n'est fait après le rang j alors on peut réduire la borne de la boucle principale (sur i).
- Écrire une fonction `triBullesOptimise` qui implémente ces deux optimisations. Quelle est sa complexité ?
- Les grands éléments qui se trouvent au début du tableau remontent rapidement (on dit que ce sont des *lièvres*) alors que les petits éléments à la fin ne descendent que d'une case à chaque étape (on dit que ce sont des *tortues*). Proposer une version modifiée de `triBulles` appelée `triShaker` où les tortues avancent aussi vite que les lièvres.
- Prouver sa correction et calculer sa complexité.

Algorithmes sur les ensembles :

$$S = a_1, \dots, a_n$$

But : répondre efficacement à
- $x \in S$

- combien de fois $x \in S$
- $\{y \geq x, y \in S\}$

Rés

$$T = [a_1, \dots, a_n]$$

def occurrences(T, x)

```
y = 0
for i in T:
    if (i == x):
        y += 1
return y
```

Complexité:

$\Theta(n)$ comparaisons

$$S = 2, 9, \text{F}, 5, \text{F}, 1, 2, \text{F}$$

$$T = [1, 2, 2, 5, \text{F}, \text{F}, \text{F}, 9]$$



def $f(T, x, \underline{\text{begin}} = 0, \underline{\text{end}} = \text{None})$

if ($\text{end} = \text{None}$):

$\text{end} = \text{len}(T) - 1$

if ($\text{end} \leq \text{begin} + 1$)

return begin if $T[\text{begin}] == x$

$m = (\text{end} - \text{begin}) // 2$

$v = T[m]$

if $v > x$:

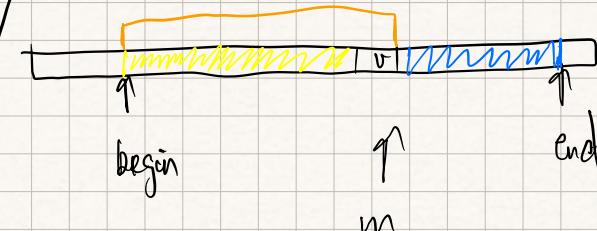
return $f(T, x, \text{begin}, m)$

if $v < x$:

return $f(T, x, m+1, \text{end})$

if $v == x$:

return $f(T, x, \text{begin}, m+1)$



Si $v > x$

Si $v < x$

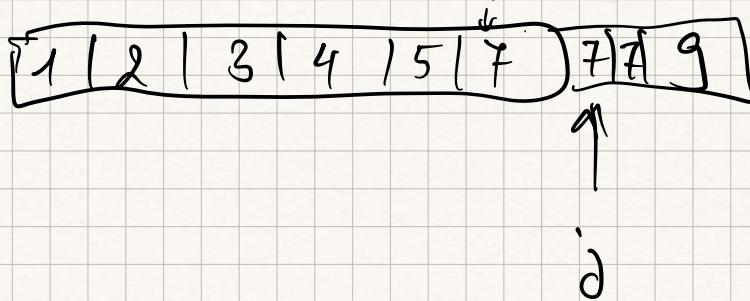
Si $v == x$

Complexité $\Theta(\log n)$

1^{ère} élé de x : i → OCC de x ?

dernière occ : j

$$x \stackrel{j-i+1}{=} 2, i=1, j=2$$



$$j - i = 1$$

Ex² Algorithmes de Tri

```
def triInsertionGauche(T)
    for i in range(n):
        for j in range(i+1):
            if T[i] < T[j]
                break
            for k in range(j,i)
                T[i], T[k] = T[k], T[i]
```

Meilleur cas : Tableau inversément trié

$\Theta(n)$ comparaisons

$\Theta(n^2)$ affectations

Pire cas : $\Theta(n^2)$ pour les 2

Insertion Droite

Meilleur cas : Tableau trié

$\Theta(n)$ comparaisons et affectations

Pire cas : Inversement trié,

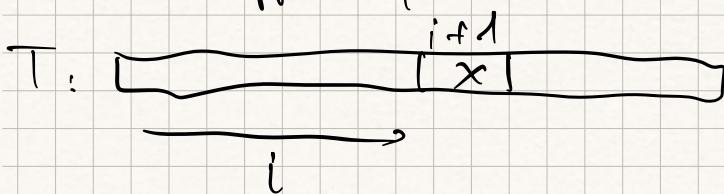
$\Theta(n^2)$ pour les deux

2x Provaient :

Après le i^{e} tour de boucle, $T[i:i+1]$ est trié

$i = 0$: évident (1 seule élément)

On suppose que c'est vrai pour



1x Après la première boucle, j est l'indice où insérer x

$$\rightarrow \underline{T[j-1]} \leq \underline{x} < \underline{T[j]}$$

Condition de sortie du break

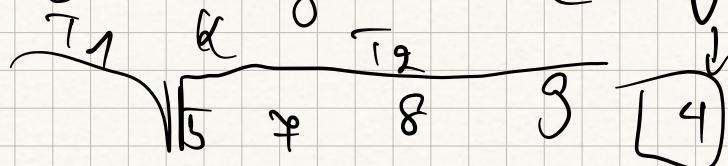
On commence à $j=0$

Comme au break à j , on a $T[i] \geq T[j-1]$

2x La 2^e boucle insère $T[i:]$ à la position j

sans changer l'ordre

v



T_1 est trié

T_2 est trié

$$T_2 \geq 0 \geq T_1$$

