Leçon 8 : Algorithmes de tri. Exemples, Complexité, Applications.

Niveau : Première ; MP2I ; MPI

Prérequis : Tas

I - Introduction et Motivation

Def 1 : Un algorithme de tri prend en entrée une liste d'éléments E muni d'un ordre total $\leq$ (on peut se passer de l'antisymétrie) et renvoie une liste S tq :
(i) S contient exactement les elements de E
(ii) les éléments de S apparaissent dans l'ordre croissant

Motivation 2 : Pourquoi trier ? Pour simplifier certaines opérations usuelles sur les listes (min, max, recherche, ...)

Activité 3 : Chercher (à la main) la présence d'un mot dans une liste non triée, puis triée dans l'ordre alphabétique.

Def 4 : Propriétés sur les tris :
* En place : utilise $O(1)$ espace en plus de l'entrée
* Stable : $\forall i<j$, $E[i] \leq E[j]$ et $E[j] \leq E[i] \Rightarrow E[i]$ avant $E[j]$ dans S.
* Online : On peut trier les données même si elles arrivent au fur et à mesure.

II - Tris quadratiques

II.1 - Tri par sélection

Algorithme 5 :

```
Tri par sélection (E):
    n = len(E)
    S = liste vide
    pour i allant de 0 à n-1:
        m = extraire min de E
        Ajouter m à la fin de S
    renvoyer S
```

Exemple 6 : E = [4;3;6;1]

i	E	m	S
/	[4;3;6;1]	/	[]
0	[4;3;6]	1	[1]
1	[4;6]	3	[1;3]
2	[6]	4	[1;3;4]
3	[]	6	[1;3;4;6]

S = [1;3;4;6]

Terminaison 7 : l'algorithme n'utilise que des boucles bornées.

Correction 8 : Invariant de boucle : "S est trié et contient les i premiers elements de E"

Complexité 9 : environ $len(E)^2$ ($O(|E|^2)$)

Propriétés 10 : Le tri est stable (si on extrait toujours le premier min).

Exercice : le réécrire pour qu'il soit en place (on perd la stabilité).

II.2 - Tri par insertion

Algo 11 :

```
Tri par insertion (E):
    n = len(E)
    pour i allant de 0 à n-1:
        v = E[i]
        j = i-1
        Tant que j>0 et E[j] > E[j+1]:
            échanger E[j] et E[j+1]
            j = j-1
    retourner E
```

Propriétés 12 : l'algo 11 est en place, stable et online.

Complexité 13 : dans le pire des cas : $O(n^2)$
dans le meilleur des cas : $O(n)$

II.3 - Application : la recherche dichotomique

Terminaison 14 : variant de boucle : "b-a"

Correction 15 : Invariant de boucle : "Si x est dans E alors son indice est entre a et b"

Complexité 16 : 1) $C(n) = C(\frac{n}{2}) + c$ 2) Si $n = 2^k$: $C(2^k) = kc$
3) C est croissante.
4) $C(n) \leq C(2^{\log(n)+1}) = O(\log_2(n))$

Ne rien écrire hors du cadre.

Algo17 :

```
Recherche_dichotomique (E, x):
    Si len(E) == 0 :
        renvoyer faux
    Sinon :
        a = 0, b = len(E)-1 , m = (a+b)//2
        Tant que (E[m] != x et (b-a) > 0) :
            Si (E[m] < x) : a = m+1
            Si (E[m] > x) : b = m-1
            m = (a+b)//2
        Si E[m] == x : renvoyer true
        Sinon : renvoyer false
```

TP18 : Implémentation Python. Comparaison des temps d'execution avec la recherche linéaire.

III - Tri efficace

III.1 - Tri fusion

Def19 : l'algorithme de tri fusion repose sur deux opérations :
- partitionner (L) : coupe L en deux listes L_1 et L_2 de même taille (à 1 près)
- fusionner (L_1, L_2) : L_1 et L_2 sont deux listes triées, renvoie L_3 la liste triée contenant exactement les éléments de L_1 et L_2.

Algo18 :

```
tri_fusion (L):
    Si L est vide ou contient 1 element :
        renvoyer L
    L1, L2 = partitionner(L)
    renvoyer fusion ( tri_fusion(L1), tri_fusion(L2))
```

```
fusion (L1, L2):
    L3 = []
    i, j = 0
    Tant que i ≤ |L1| et j ≤ |L2| :
        Si L1[i] ≤ L2[j] :
            L3.ajouter (L1[i])
            i++
        Sinon :
            L3.ajouter (L2[j])
            j++
    Ajouter la fin de L1 et de L2 à L3
    retourner L3.
```

Dev 1 : Correction et Terminaison du tri fusion

Propriétés 19 : tri stable mais pas en place. Pas en ligne.

Complexité 20 : $C(n) = 2C(n/2) + O(1) \Rightarrow C(n) = O(n \log n)$

III.2 - Tri rapide (Quick-Sort)

Algo21 :

```
tri_rapide (L, début, fin):
    Si debut ≥ fin-1 : retourner L
    pivot = L[fin-1]
    i = début
    Pour j allant de début à fin-2 :
        si L[j] ≤ pivot :
            permuter L[j] et L[i]
            i++
    permuter L[i] et L[fin-1]
    tri_rapide (L, début, i-1)
    tri_rapide (L, i+1, fin)
    retourner L
tri_rapide (L):
    renvoie tri_rapide (L, 0, |L|)
```

Complexité 21 : pire des cas : $O(n^2)$; meilleur : $O(n\log n)$. Choix du pivot aléatoire, en moyenne $O(n\log n)$

Propriété 22 : tri en place, non stable, non online.

III.3 - Tri par tas

Algo 24 : tri tas (L) :
insérer les elts de L dans un tas vide T
Extraire successivement le minimum

Complexité 25 : $O(n \log n)$

Propriété 26 : Possible en place ; non stable ; non online.

III.4 - Minoration de la complexité du tri

Propriété 27 : On ne peut pas trier n éléments avec une complexité inférieure à n-1.

Propriété 28 : Un algorithme de tri par comparaison aura une complexité au pire des cas au mieux $O(n\log n)$.

Remarque 29 : Si on ne trie pas par comparaison, on peut avoir des complexités plus faibles, par exemple si E contient des entiers $\leqslant 10$, on peut compter les elements de chaque valeur.

Dev 2 : Amelioration du tri par comptage avec des dictionnaires

IV - Applications

IV.1 - Algorithmes gloutons.

Def 30 : Un algorithme glouton est un algorithme qui résout un pb d'entrée E en :

1) Faire un précalcul sur E (ex : trier E)

2) Construire une solution en parcourant E sans jamais revenir en arrière.

Exemple 31 : On a n évènements sportifs respectivement entre les dates $\{di\}_{i\leqslant n}$. $\{fi\}_{i\leqslant n}$. Combien faut-il de gymnase au minimum et comment leur allouer les évènements ?

Glouton : 1) Trier les di par ordre croissant.
2) Mettre chaque evenement dans le premier gymnase disponible, quitte à en ouvrir un nouveau.

Propriété 32 : Ce glouton est optimal.

Exemple 33 : Instance : n pièces de valeurs $\{p_1, \dots, p_n\}$, $S \in \mathbb{N}$
Pb : Combien de pièces rendre au minimum pour faire S ?
Glouton : 1) Trier les pièces par valeur décroissante.
2) Rendre la plus grosse possible à chaque fois.

Def 34 : Un arbre courant de poids minimal est un sous-graphe connexe acyclique maximal de poids minimal.

Algorithme 34 (Kruskal)

```
Kruskal(L) : // L est la liste des arêtes
    trier L par poids croissant
    A = []
    Pour a dans L :
        Si a U A n'a pas de cycle :
            Ajouter a à A
    renvoyer A.
```

IV.2 - Somme de flottants

On veut sommer n flottants. A chaque somme, on peut avoir une erreur d'arrondi. Dans quel ordre faut-il sommer pour minimiser cette erreur ?