DS 1

le 04/10/2024
Durée : 2h

Le devoir est constitué d'un exercice (sur 6 points) et d'un problème. Le barême tiendra compte de la longueur de ce problème (la partie IV sera comptée en bonus).

Exercice : tri insertion avec dichotomie

Rappelons tout d'abord le principe général du tri insertion. Partant d'un tableau de nombres de taille n (représenté ici par une liste python) noté `T`, on souhaite le trier, sans créer de nouveau tableau, dans l'ordre croissant.

- On insère tout d'abord `T[1]` dans `T[:1]` de sorte que `T[:2]` soit trié (en fait on échange `T[0]` et `T[1]` si `T[0]>T[1]`).
- À la deuxième étape, on insère `T[2]` dans `T[:2]` de sorte que `T[:3]` soit trié.
- À la k-ème étape ($k < n$), `T[:k]` est déjà trié. On insère `T[k]` dans `T[:k]` de sorte que `T[:k+1]` soit trié.

Le principe pour insérer `T[k]` dans `T[:k]` est, après avoir enregistré la valeur de `T[k]` dans une variable, de décaler les éléments de `T[:k]` vers la droite tant qu'ils sont strictement supérieurs à `T[k]` (on peut aussi s'arrêter car on est arrivé au bout du tableau). Puis on place `T[k]` à la droite du premier élément inférieur ou égal à `T[k]` que l'on a détecté (ou en position 0 si on n'en a détecté aucun).
Prenons un exemple pour fixer les idées. Partant de la liste `[4,1,5,0,2,3]`, on obtient après 3 étapes la liste `[0,1,4,5,2,3]`. On doit alors insérer `2` dans la partie de la liste située à sa gauche. Pour cela, on va décaler `5` puis `4` d'un cran vers la droite ; on tombe alors sur le nombre `1` (inférieur à `2`) ce qui nous indique que l'on doit arrêter et placer `2` en position `2`.
Le but de l'exercice est de programmer un tri insertion en modifiant la recherche de l'endroit où l'on doit insérer `T[k]` dans `T[:k]` : cette recherche ne se fera plus en descendant `T[:k]` mais par dichotomie.

1. Compléter la fonction suivante afin qu'elle renvoie, étant donné `k` , l'indice de `T` auquel on doit insérer `T[k]` dans `T[:k]` supposé trié. La recherche de cet indice se fera par dichotomie. La fonction ne doit pas modifier le tableau.

```
def indice(T,k):#renvoie l'indice auquel il faut ins\'erer T[k]
                #dans T[:k] suppos\'e tri\'e
    if T[0] >= T[k]:
        return ...
    elif T[k-1] <= T[k]:
        return ...
    else :
        a=0;b=k-1  # on maintient T[k]>T[a] et T[k]<T[b]
        while ....
            c=(b+a)//2
            if T[c] == T[k]:
                ....
            elif ....
                ....
            elif ...
                ....
        return ....
```

2. Quelle est la complexité (en fonction de k) de l'algorithme de la fonction précédente.
3. Écrire une fonction qui trie le tableau `T` par insertion en utilisant la fonction `indice(T,k)` pour déterminer à chaque étape la place à laquelle il faut insérer `T[k]` (on ne doit bien sûr utiliser aucune fonction d'insertion prédéfinie).
4. Donner un invariant de la boucle externe de cet algorithme.
5. Montrer que si l'on ne compte que les comparaisons la complexité de ce tri est en $\mathcal{O}(n \log n)$.
6. Quelle est sa complexité si l'on tient compte de toutes les opérations élémentaires (y compris les affectations) ?