exos

October 3, 2024

1 Exercices

1.1 Exercice1 Révisions

1. Soit **tab** un tableau de taille **n** , trié jusqu'au rang -1 ($<$).
   Écrire la fonction récursive **inserer(tab: list, j:int) → None** qui place l'élément de rang dans la partie triée du tableau.
2. Écrire la fonction **tri_insertion(tab: list) → None** qui trie **tab** en utilisant la fonction **inserer**.
3. Construire par compréhension un tableau de 20 entiers compris entre 0 et 100.
4. Tester la fonction de tri sur le tableau.

1.2 Exercice2 Tri stable

Un tri est stable quand les éléments de même valeur garde leur place relative.

t = [(1, 5), (3, 4), (1, 1), (2, 9), (1, 2)] *# On décide de trier t par rapport au premier entier de chaque tuple* t = [(1, 5), (1, 1), (1, 2), (2, 9), (3, 4)] *# les deux premiers tuples gardent leur place relative.*

1.Dérouler le tri par insertion de l'exercice précédent sur le tableau du code ci-dessus. On décide de trier le tableau en fonction du premier élément de chaque tuple. Le tri par insertion semble-t-il stable?

2. Modifier la fonction **inserer** pour qu'elle réalise ce tri.
3. Mêmes questions pour le tri fusion.
4. Écrire la fonction **tri_selection(tab: list) → None**.
5. Montrer grâce à un exemple que le tri par sélection n'est pas stable. vide: **-** Choisir l'élément du milieu **-** Si l'élément choisi est celui cherché: $*$ Renvoyer l'indice de l'élément **-** Sinon si l'élément choisi est inférieur à celui cherché: $*$ Ne conserver que la moitié supérieure **-** Sinon: $*$ Ne conserver que la moitié inférieure 1.Écrire la fonction impérative **dichotomie_imp(tab: list, x: int) → int** de la recherche dichotomique, qui renvoie la position de **x** dans **tab** ou **-1** s'il n'est pas présent.
6. Écrire la version récursive **dichotomie_rec(tab: list, x: int, debut: int, fin: int) → int**
7. Tester les deux fonctions sur un tableau trié de 50 entiers aléatoires.
8. Déterminer la complexité de l'algorithme de recherche dichotomique.

1.3 Exercice3 Dichotomie**

Rappel: Un algorithme de recherche dichotomique cherche un élément dans une collection **triée** , en divisant la collection en deux et en ne conservant que la partie vérifiant la recherche.

L'algorithme de la fonction de dichotomie peut s'écrire: L'algorithme de la fonction de dichotomie peut s'écrire: • Tant que le tableau n'est pas vide:

```
- Choisir l'élément du milieu
- Si l'élément choisi est celui cherché:
    * Renvoyer l'indice de l'élément
- Sinon si l'élément choisi est inférieur à celui cherché:
    * Ne conserver que la moitié supérieure
- Sinon:
    * Ne conserver que la moitié inférieure
```

1. Écrire la fonction impérative **dichotomie_imp(tab: list, x: int) → int** de la recherche dichotomique, qui renvoie la position de **x** dans **tab** ou **-1** s'il n'est pas présent.

2. Écrire la version récursive **dichotomie_rec(tab: list, x: int, debut: int, fin: int) → int**

3. Tester les deux fonctions sur un tableau trié de 50 entiers aléatoires.

4. Déterminer la complexité de l'algorithme de recherche dichotomique.

1.4 Exercice 4 Diviser pour régner

La fonction **mystere** implémente un algorithme de type diviser pour régner.

```python
def mystere(tab: list, debut: int, fin: int) -> int:
    if debut == (fin - 1):
        return tab[debut]
    else :
        milieu = (debut + fin) // 2
        gauche = mystere(tab, debut, milieu)
        droite = mystere(tab, milieu, fin)
        if (gauche > droite):
            return gauche
        else :
            return droite
```

Soit le tableau:

```python
tab = [5, 71, 23, 45, 28, 89, 63, 39]
```

1. Dessiner l'arbre des séparations engendré par la fonction sur la liste **tab**.
2. Dessiner l'arbre des recombinaisons. Quelle valeur renvoie l'appel **mystere(tab)**?
3. Que fait cette fonction?
4. Discuter de la complexité de la fonction.

[ ]:

```

```