ISM MPSI/PCSI, Informatique
Année 2023/2024

Corrigé du TP Informatique 27

Exercice 1

1. On saisit :

```
def mini(T,deb,fin):
    ind=deb
    for k in range(deb,fin):
        if T[k]<T[ind]:
            ind=k
    return ind
```

2. On saisit :

```
def tri_select(T):
    n=len(T)
    for i in range(n):
        ind=mini(T,i,n)
        T[ind],T[i]=T[i],T[ind]
```

3. L'implémentation effectuée modifie directement le liste à trier et on ne crée aucune variable de taille de même ordre que celle de l'argument `T` ce qui prouve le caractère en place de ce tri. La complexité temporelle de `mini(T,deb,fin)` est clairement un grand O de `fin-deb` et celle de `tri_select` est donc pour une liste `T` de taille n :

$$\sum_{i=0}^{n-1} \mathrm{O}(n-i) = \mathrm{O}\left(\sum_{i=0}^{n-1}(n-i)\right) = \mathrm{O}(n^2)$$

Le tri par sélection est en place de complexité temporelle en $\mathrm{O}(n^2)$.

Exercice 2

1. On saisit :

```
def tri_ins1(T):
    n=len(T)
    for i in range(1,n):
        j=i
        while j>0 and T[j-1]>T[j]:
            T[j],T[j-1]=T[j-1],T[j]
            j-=1
```

2. L'implémentation effectuée modifie directement le liste à trier et on ne crée aucune variable de taille de même ordre que celle de l'argument `T` ce qui prouve le caractère en place de ce tri. On considère une liste de taille n. Si celle-ci est déjà triée, on ne rentre jamais dans la boucle `while` et le coût temporel est en

$$\sum_{i=1}^{n-1} \mathrm{O}(1) = \mathrm{O}(n)$$

Dans le pire des cas, on rentre `i` fois dans la boucle `while` d'où un coût en

$$\sum_{i=1}^{n-1}\sum_{j=1}^{i} \mathrm{O}(1) = \sum_{i=1}^{n-1} \mathrm{O}(i) = \mathrm{O}\left(\sum_{i=1}^{n-1} i\right) = \mathrm{O}(n^2)$$

> Le tri par insertion est en place de complexité temporelle en $\mathrm{O}(n)$ dans le meilleur des cas et en $\mathrm{O}(n^2)$ dans le pire des cas.

3. On saisit :

```
def tri_ins2(T):
    n=len(T)
    for i in range(1,n):
        c=T[i]
        j=i-1
        while j>=0 and T[j]>c:
            T[j+1]=T[j]
            j-=1
        T[j+1]=c
```

Cette amélioration ne change pas la classe de complexité.

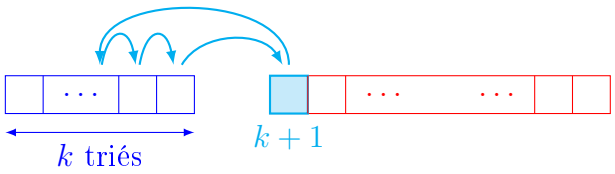

FIGURE 1 – Tri par insertion amélioré

Exercice 3

1. On saisit :

```
def rech_dicho(T,deb,fin,elt):
    """Recherche dichotomique dans liste triée T[deb:fin]"""
    while deb<fin:
        m=(deb+fin)//2
        if T[m]<elt:
            deb=m+1
        else:
            fin=m
    return deb
```

2. On saisit :

```
def tri_ins3(T):
    n=len(T)
    for i in range(1,n):
        j=rech_dicho(T,0,i,T[i])
        c=T[i]
        for k in range(i,j,-1):
            T[k]=T[k-1]
        T[j]=c
```

3. Le bénéfice est mince ! En effet, dans le pire des cas où `j=0` à chaque étape, on a

$$\sum_{i=1}^{n-1} [\mathrm{O}(\log i) + \mathrm{O}(i)] = \sum_{i=1}^{n-1} \mathrm{O}(i) = \mathrm{O}(n^2)$$

En fait, l'insertion requiert de décaler toutes les valeurs supérieures à celles qu'il faut insérer et cette phase a un coût linéaire qui annule en grande partie le bénéfice de la recherche en coût logarithmique. Cette version est toutefois un peu meilleure que les deux précédentes puisqu'elle réalise moins de comparaisons pour déterminer la position de l'élément courant à insérer.

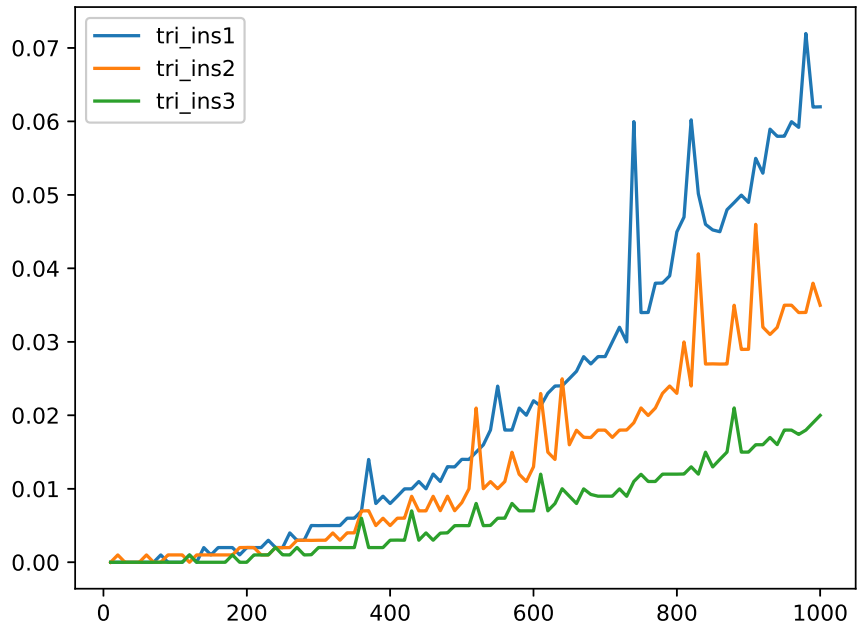

FIGURE 2 – Comparaison des différents tris par insertion