CORRIGÉ DU 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

1.
```
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 0
    elif T[k-1]<=T[k]:
        return k
    else :
        a=0;b=k-1# on maintient T[k]>T[a] et T[k]<T[b]
        while b-a>1:
            c=(b+a)//2
            if T[c]==T[k]:
                return c
            elif T[c]<T[k]:
                a=c
            elif T[c]>T[k]:
                b=c
        return b
```

2. La complexité de la dichotomie dans un tableau de longueur k est en $\log k$.

3.
```
def triInsereDicho(T):
    n=len(T)
    for k in range(1,n):
        ind=indice(T,k)
        if ind!=k :#si ind==k, on ne fait rien
            tmp=T[k]
            j=k-1
            while j>=ind:
                T[j+1]=T[j]
                j-=1
            T[ind]=tmp
```

4. L'invariant de la boucle externe (celle qui apparaît dans la fonction `triInsereDicho`) est : `T[:k]` est trié.

5. Chaque recherche de l'indice où insérer `T[k]` nécessite un nombre de comparaison en $\mathcal{O}(\log k)$ (et donc $\mathcal{O}(\log n)$). Comme on le fait pour chaque élément du tableau, cela donne donc un nombre de comparaisons en $\mathcal{O}(n \log n)$.

6. L'ordre donné dans la question précédent donne l'impression que l'on a amélioré la complexité du tri insertion. Mais lors de l'insertion proprement dite de `T[k]` dans `T[:k]` on fait (dans le pire des cas) k affectation. En comptant ces opérations, on retombe donc sur la complexité du tri insertion classique, en $\mathcal{O}(n^2)$.