Synthèse Algorithme de 1ière Partie1

Les Tris

Préambule

Pour pouvoir tester nos tris nous allons créer une fonction qui prend en argument n, le nombre de valeur de nombre tirés aléatoirements entre 0 et 100 et qui renvoie un tableau les contenant.

```
import random

def alea(n):
    l=[]
    for i in range(n):
        l.append(random.randint(0,100))
    return l
```

voici une version plus compacte avec un tableau par compréhension.

```
def alea(n):
   return [random.randint(0,100) for i in range(n)]
```

Tri par insertion

algorithme

```
procédure tri_insertion(tableau T)
pour i de 1 à taille de T -1
# mémoriser T[i] dans x
x ← T[i]
# décaler vers la droite les éléments de T[0]..T[i-1] qui sont plus grands que x (en partant de T[i-1])
j ← i
tant que j > 0 et T[j - 1] > x
T[j] ← T[j - 1]
j ← j - 1
fin tant que
# placer x dans le "trou" laissé par le décalage
T[j] ← x
fin pour
fin procédure
```

Complexité : La complexité du tri par insertion est $\Theta(n^2)$ dans le pire cas et en moyenne, et linéaire dans le meilleur cas.

Synthèse Algorithme de 1ière Partie1

Code Python

```
def tri_insertion(tab):
    for i in range(1,len(tab)):
        x=tab[i]
        j=i
        while j>0 and tab[j-1]>x:
            tab[j]=tab[j-1]
            j=j-1
        tab[j]=x
    return tab
```

Tri par sélection

algorithme

procédure tri_selection(tableau t)
pour i de 0 à taille de t - 2
min ← i
pour j de i + 1 à taille de t - 1
si t[j] < t[min], alors min ← j
fin pour
si min ≠ i, alors échanger t[i] et t[min]
fin pour
fin procédure

Complexité : Sa complexité est $\Theta(n^2)$ le tri par sélection effectue n(n-1)/2 comparaisons

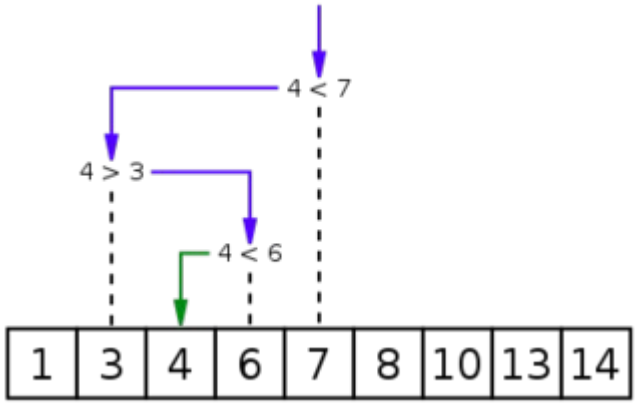
Code Python

```
def tri_selection(tab):
    for i in range(len(tab)-1):
        min=i
        for j in range(i+1,len(tab)):
            if tab[j]<tab[min]:
                min=j
        if min!=i:
            tab[i],tab[min]=tab[min],tab[i]
    return tab
```

Synthèse Algorithme de 1ière Partie1

Recherche Dichotomique

La **recherche dichotomique**, est un algorithme de recherche pour trouver la position d'un élément dans un tableau trié. Le principe est le suivant : comparer l'élément avec la valeur de la case au milieu du tableau ; si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente.



Source : wikipedia

Préambule

il nous faut un tableau trier sans doublon.

```
import random

def alea_triee(n):
  return sorted([random.randint(0,100) for i in range(n)])
```

Méthode itérative

pseudo code :

```
#déclarations
 début, fin, valeur, milieu: Entiers
 t : Tableau [0..N] d'entiers classé
#initialisation
 début ← 0
 fin ← N-1
 Saisir valeur
#Boucle de recherche
# La condition début inférieur ou égal à fin permet d'éviter de faire
# une boucle infinie si 'val' n'existe pas dans le tableau.
 Tant que début < fin:
     milieu ← partie_entière((début + fin)/2)
     si t[milieu] == valeur:
       retourner ← vrai
     sinon si val > t[mil]:
          début ← milieu+1
     sinon:
          fin ← milieu-1
 retourner faux
```

Synthèse Algorithme de 1ière Partie1

Code Python :

```
def rechercheDichotomiqueIterative(tableau,nombre):
        a=0
        b=len(tableau)
        while(a<b):
                m=(a+b)//2
                if nombre==tableau[m]:
                        return True#si l'on veut renvoyer l'indice on renvoie m
                elif nombre>tableau[m]:
                        a=m+1
                else:
                        b=m-1
        return False
```

Méthode récursive

```
recherche_dichotomique_récursive( liste_triée,élément):
  m = longueur de liste_triée / 2;#division euclidienne
  si taille de liste_triée = 0:
     renvoyer faux ;
  si liste_triée[m] = élément :
     renvoyer vrai ;
  si liste_triée[m] > élément :
     renvoyer recherche_dichotomique_récursive(liste_triée[0:m-1],élément) ;
  sinon :
     renvoyer recherche_dichotomique_récursive(liste_triée[m+1:longueur liste_triée],élément) ;
```

Code Python :

```
def rechercheDichotomiqueRecursive(tableau,nombre):
        if len(tableau)==0:
                return False
        m=len(tableau)//2
        if nombre==tableau[m]:
                return True#si l'on veut renvoyer l'indice on renvoie m
        if nombre>tableau[m]:
                return rechercheDichotomiqueRecursive(tableau[m+1:],nombre)
        else:
                return rechercheDichotomiqueRecursive(tableau[:m],nombre)
```

Bout de programme pour tester la recherche dichotomique(les 2 méthodes)

```
tableau=alea_triee(10)
print(tableau)
nombre=int(input("entrez le nombre à rechercher:\n"))
print("recherche dichotomique itérative")
print("le nombre ",nombre,"est dans le tableau :",rechercheDichotomiqueIterative(tableau,nombre))
print("recherche dichotomique récursive")
print("le nombre ",nombre,"est dans le tableau :",rechercheDichotomiqueRecursive(tableau,nombre))
```