Recherche dichotomique

Présentation

L'algorithme classique de recherche d'un élément `x` dans un tableau `tab` consiste à parcourir tous les éléments du tableau. Cela prend donc dans le pire des cas autant d'étapes que le nombre d'éléments du tableau : si le tableau contient *n* éléments, la complexité est linéaire. On peut réduire considérablement le nombre d'étapes si la liste de départ est triée en faisant une **recherche dichotomique**.

les Objectifs :

- Implémenter un programme qui recherche une élément dans un liste le plus rapidement possible

Programme 1 :

Créer un Tableau aléatoire à l'aide de la méthode de compréhension de liste.

Trier le Tableau avec la fonction native sort()

Implémenter une fonction **recherche**(Tableau, element) qui recherche une occurence "element" en parcourant le Tableau de façon séquentiel.

Calculer le temps d'exécution du programme dans le cas le plus défavorable.

```
from time import time
from random import*
Tableau = [.....]
```

Effectuer des tests avec l'instruction `assert`

Principe de la recherche dichotomique

L'idée centrale de cette approche repose sur l'idée de réduire de moitié l'espace de recherche à chaque étape : on regarde la valeur du milieu et si ce n'est pas celle recherchée, on sait qu'il faut continuer de chercher dans la première moitié ou dans la seconde.

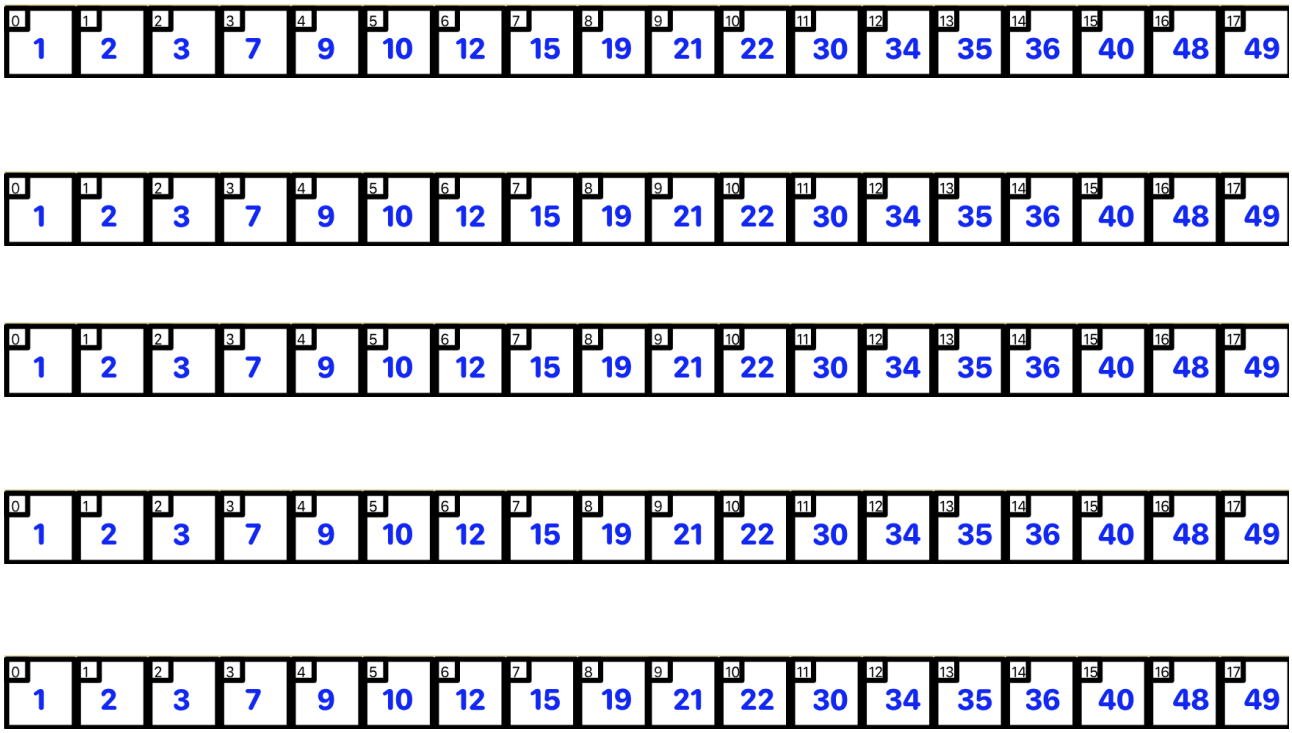
Plus précisément, en tenant compte du caractère trié du tableau, il est possible d'améliorer l'efficacité d'une telle recherche de façon conséquente en procédant ainsi :

- on détermine l'élément au milieu du tableau ;
- si c'est la valeur recherchée, on s'arrête avec un succès ;
- sinon, deux cas sont possibles :
    - si la valeur du milieu est plus grande que la valeur recherchée, comme le tableau est trié, cela signifie qu'il suffit de continuer à chercher dans la première moitié du tableau ;
    - sinon, il suffit de chercher dans la moitié droite.

Pour la suite, nous comparerons la performance de nos programmes de tri au tri natif de python.

Représentation de la recherche dichotomique

Faire la simulation de la recherche dichotomique pour trouver la valeur 34. (Chaque tableau correspond à une étape).



Programme 2 :

Créer un Tableau aléatoire à l'aide de la méthode de compréhension de liste.

Trier le Tableau avec la fonction native sort()

Implémenter une fonction recherche_dichotomique(Tableau, element) qui recherche une occurence "element" en parcourant le Tableau de façon dichotomique.

Calculer le temps d'exécution du programme dans le cas le plus défavorable.

```
from random import *
from math import *
from time import time
def recherche_dichotomique(tableau,element):
```

Complexité temporelle

Nous pouvons chercher quel sera le nombre d'opérations à effectuer au maximum pour trouver l'élément que l'on recherche et en profiter pour comparer la recherche dichotomique et la recherche séquentielle (qui cherche depuis le début et ensuite case par case) :

Taille du tableau	0	2	8	16	32	64	O( )
Séquentielle							
Dichotomique							

Représenter graphiquement les deux méthodes