Algorithmie
Epreuve pratique
Recherche dichotomique

1 Spécifications

1.1 Spécifications algorithme

Ecrire en pseudo code une fonction nommée **rech_ dicho(T,elt)** qui prend en argument un tableau T d'entiers triés et un entier elt, et qui renvoie False si elt n'est pas dans T, True dans le cas inverse.
La méthode utilisée pour la recherche dichotomique.

1.2 Spécification implémentation Python

Ecrire en langage Python une fonction nommée **rech_ dicho(T,elt)** qui prend en argument une liste Python T d'entiers triés, et un entier elt, et qui renvoie True si elt est présent dans T, False sinon. La méthode utilisée pour la recherche est recherche dichotomique.

2 Résolution

2.1 Un algorithme

```
1  FONCTION rech_ dichotomique(T : tableau trié d'entiers,elt : entier)
2  |  a ← 0
3  |  b ← longueur(T) − 1
4  |  TANT QUE a ≤ b FAIRE
5  |  |  milieu ← (a + b ) // 2 # // signifie quotient
6  |  |  SI T[milieu] = elt ALORS
7  |  |  |  RENVOYER True
8  |  |  SI T[milieu] < elt ALORS
9  |  |  |  a ← milieu + 1
10 |  |  SINON
11 |  |  |  b ← milieu - 1
12 |  RENVOYER False
```

2.2 Une implémentation en Python

```
def rec_dicho(T,elt):
    a = 0
    b = len(T) - 1
```

```
    while a <= b:
        milieu = (a + b) // 2
        if T[milieu] == elt:
            return True
        elif T[milieu] < elt:
            a = milieu + 1
        else:
            b = milieu - 1
    return False
```

2.3 Complexité

Propriété 5.1

La complexité temporelle de cet algorithme est en $\mathcal{O}(log_2(n))$. On parle de complexité logarithmique.

2.4 Terminaison

La boucle TANT QUE peut poser problème, et il faut s'assurer que dans tous les cas, la boucle n'est pas infinie.
Considérons comme variant de boucle la longueur du tableau en cours.

Exercice 5.1

Montrer que ce variant décroît strictement et finira par être strictement inférieur à 1.

3 Compléments - Rappels de première

3.1 Principe

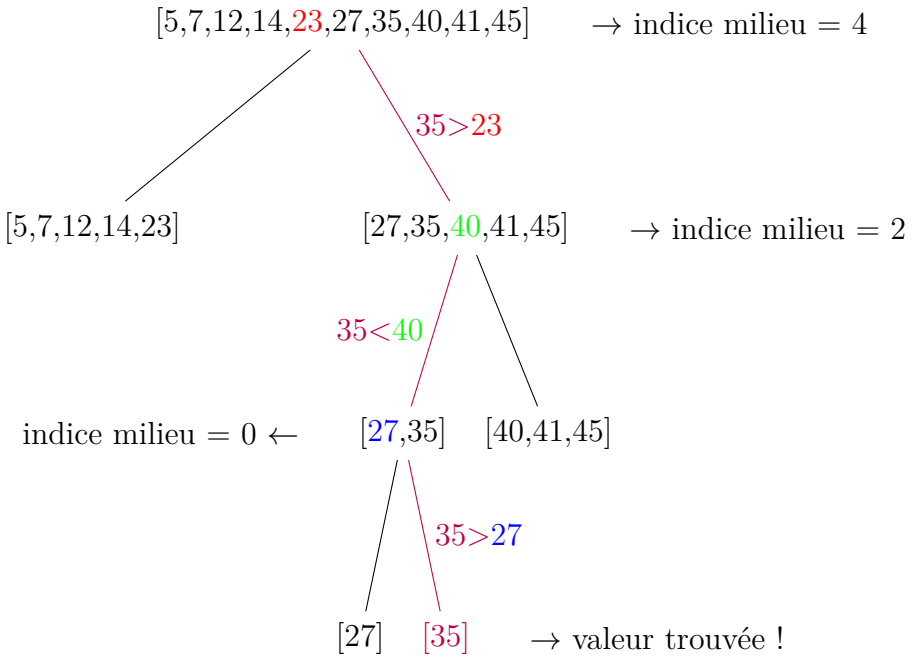
Considérons le tableau trié [5,7,12,14,23,27,35,40,41,45].

Si on veut trouver une valeur, 35 par exemple, il est possible d'utiliser l'algorithme de recherche d'une occurrence. Dans le pire des cas, on parcourt le tableau en entier. Dans notre exemple, il faut faire 7comparaisons.

Mais comme la liste de départ est triée, la recherche ***dichotomique*** permet d'améliorer la performance de la recherche.
On souhaite rechercher l'entier 35 :

- Si la liste est vide, la recherche est finie et on renvoir False
- Sinon, on cherche la valeur la plus "centrale" dans la liste et on la compare avec l'élément recherché.
  - Si la valeur est celle recherchée, la recherche est finie, et on renvoie True
  - Si la valeur est strictement inférieure à l'élément recherchée, reprendre la recherche avec la la seconde moitié de la liste

– Si la valeur est strictement supérieure à l'élément cherché, reprendre la recherche avec la première moitié de la liste.



☞ Il suffit ici de 3 tours de boucles.

3.2 Etude théorique

Considérons un tableau de taille n.

☞ On admet que la partie entière de $log_2(n)$ augmentée de 1 donne le nombre de chiffres de l'écriture binaire de n.

Exemples

- $log_2(4) = 2$, et l'écriture binaire de 4 (100) contient 3 chiffres.
- $log_2(64) = 6$, et l'écriture binaire de 64 (1000000) contient 7 chiffres.
- $log_2(24) \simeq 4.58$, et l'écriture binaire de 24 (11000) contient 5 chiffres.

A chaque étape de la recherche dichotomique, on divise la longueur de la liste par 2 (à 1 près).

Or, diviser un entier par 2 revient à supprimer le bit de poids faible dans son écriture binaire (toujours à 1 près !)

- 24 est représenté en binaire par 11000. 12 est représenté en binaire par 1100. (on a supprimé le dernier chiffre).
- et 6 est représenté par 110 (on a supprimé le dernier chiffre).
- 3 est représenté par 11 (on a supprimé le dernier chiffre).
- si on continue le procédé en enlevant une nouvelle fois le dernier chiffre, on arrive à 1 comme écriture binaire, qui correspond au nombre 1. C'est bien la moitié de 3, à 1 près.

Ainsi, si la longueur du tableau est n, et puisqu'on divise par 2 la longueur du tableau à chaque étape, on enlève le bit de poids faible à l'écriture binaire de n à chaque étape.
Comme l'écriture binaire de n comporte $log_2(n) + 1$ chiffres, il faudra au maximum $log_2(n)$ étapes !

Ainsi, le nombre maximal d'étapes pour la recherche dichotomique dans un tableauu de longueur n est $log_2(n)$.

Autrement dit, le nomnre maximum de comparaisons dans un tableau de longueur$n$ est $log_2(n)$.

on a donc une complexité temporelle proportionnelle à $log_2(n)$. On parle de complexité logarithmique.
On note $\mathcal{O}(log_2(n))$.

3.3 Illustration sur des exemples

Exercice 5.2

Compléter le tableau suivant, vous comprendrez l'intérêt de la recherche dichotomique par rapport à une recherche séquentielle...Bien sûr, il faut que le tableau soit trié !!
Vous indiquerez les temps en nanoseconde !
⚠ Il faudra sûrement utiliser un chronomètre plus fin avec perf_ counter ou perf_ counter_ ns

Taille de la liste	0	1	2	4	128	1 000	10 000	100 000	1 000 000	n
Recherche séquentielle										$\mathcal{O}(n)$
Recherche dichotomique										$\mathcal{O}(log_2(n))$

***