

Recherche dichotomique : application de la méthode "diviser pour régner" .

Rappel de l'algorithme de recherche dichotomique d'une valeur dans un tableau déjà trié.

Objectif: déterminer si une valeur a est présente ou pas dans un tableau trié T .

On rappelle ci-dessous l'algorithme:

La méthode consiste à couper le tableau en deux parties ,on coupe généralement au milieu du tableau qui correspond à l'indice m :

- Ou bien a est égal à $T[m]$, alors on renvoie vrai.
- Sinon si $a < T[m]$, on recommence le procédé sur la première partie du tableau.
- Sinon on recommence le procédé sur la deuxième partie.

Fonction itérative en Python

```
#une version itérative de recherche_dichotomie
def rech_dicho_iter(T,l,r,a):
    '''
    Entrees: T tableau de nombres déjà trié
             l : indice du tableau où commence la recherche
             r :indice du tableau où finit la recherche
             a : la valeur qu'on cherche
    Sortie : bool - Vrai ou faux
    la fonction est itérative - un exemple de Diviser pour regner
    '''
    while l<=r:
        m=(l+r)//2
        if T[m] == a:
            return True
        elif T[m]<a:
            l=m+1
        else:
            r=m-1
    return False
```

Fonction récursive en Python

Complétez le code de la fonction récursive suivante :

```
def rech_dicho(T,l,r,a):
    '''
    Entrees: T tableau de nombres déjà trié
             l : indice du tableau où commence la recherche
             r :indice du tableau où finit la recherche
             a : la valeur qu'on cherche
    Sortie : bool - Vrai ou faux
    la fonction est récursive - un exemple de Diviser pour regner
    '''
    if l>r:
        return False
    m=(l+r)//2

    if .....:
        return True
    elif T[m]<a:
        return .....
    else:
        return .....
```

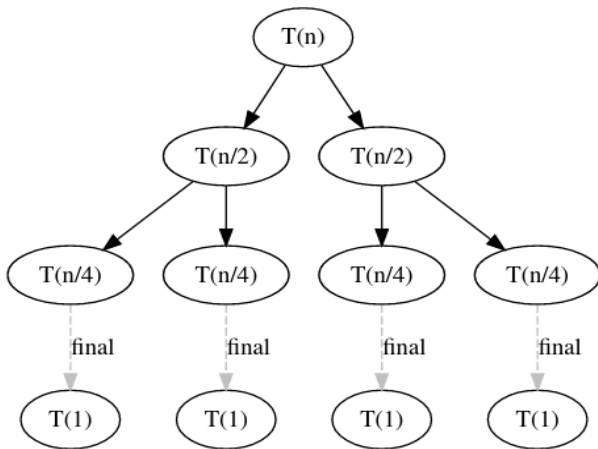
Complexité de l'algorithme de recherche dichotomique

Soit un tableau trié de taille n .

Supposons que $n = 2^b$.

A chaque appel récursif, on divise le problème en deux.

Considérons l'arbre qui nous mène au cas de base où le tableau ne contient qu'un élément.



On arrive au dernier niveau de l'arbre en b étapes, ce qui correspond comme nous le verrons plus tard, à la hauteur de l'arbre.

L'égalité $n = 2^b$ implique $b = \log_2(n)$.

L'algorithme emprunte un chemin unique depuis la racine de l'arbre pour résoudre le problème et s'arrête au maximum au bout de b étapes.

A chaque étape le coût est en $O(1)$ celui d'une opération $m = (l + r) // 2$ et de deux comparaisons au maximum ($l \leq r$ et la comparaison entre $T[n]$ et a).

Au total, on obtient une complexité en $O(\log_2(n))$.

Notons que si la taille du tableau est $l \neq 2^n$, on prend le plus petit entier n vérifiant $l \leq 2^n$.

La complexité est alors au maximum en $O(\log_2(n))$.