

A) Double garage ...

1) Est-ce que les séquences suivantes sont valides ?

a) Oui !

b) Non : La quatrième sortie intervient sur un garage vide, en effet v_1 , v_2 et v_3 sont déjà sorties.

2) La règle :

Le fonctionnement de ce garage est assimilable à celui d'une pile à deux entrées. Si l'on symbolise l'action "entrer une voiture" par les symboles E_1 ou E_2 (empiler) suivant l'entrée empruntée et l'action "sortir une voiture" par le symbole D (dépiler).

La règle peut être formulée comme suit, une séquence est admissible :

- Si elle contient autant de D que de E_1 et de E_2 cumulés (à la fin la pile est vide)
- Si toutes les actions qui lui correspondent peuvent être accomplies dans l'ordre indiqué par la séquence : on ne peut dépiler que s'il reste au moins un élément. Donc la somme des E_1 et E_2 doit toujours être supérieure ou égale à celle des D .

B) Dichotomie ...

1. L'arbre de décision de la recherche dichotomique d'un élément dans une liste de 16 éléments est le suivant :

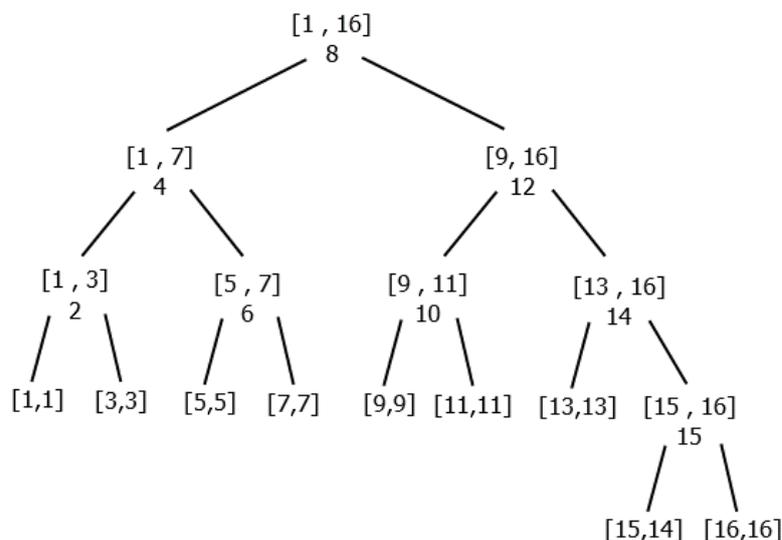


Figure 1 _ Arbre de décision de la recherche dichotomique

Chaque nœud représente l'intervalle de recherche (les bornes gauche et droite) ainsi que l'indice calculé du médian. On considère ici une version de l'algorithme qui s'arrête dès que les bornes se croisent ou sont identiques.

2. Nombre de comparaisons : $32 = 2 \times (15+1)$ ($\log_2(32768) = 15$)
3. Longueur de la liste : 65536 (32768×2)

C) Recherche des indices ...

Spécifications :

La fonction **search_indexes(L, val)** retourne les valeurs du premier et du dernier indice de l'élément **val** dans la liste triée **L**, le couple **(-1,-1)** si **val** n'est pas présent.

```
def search_indexes (L, val):
    (start, end) = (-1, -1)
    i = 0
    l = len(L)
    while i < l and L[i] < val:
        i += 1
    if i < l and L[i] == val:
        start = i
        while i < l and L[i] == val:
            i += 1
        end = i - 1
    return (start, end)
```

D) Somme consécutive ...

Spécifications :

La fonction **consecutive_sum(L, S)** vérifie s'il existe une suite d'éléments consécutifs dont la somme est **S** (avec $S > 0$) dans la liste d'entiers naturels **L**.

```
def consecutive_sum (L, S):
    s_cur = 0
    (i, n) = (0, len(L))
    while i < n and s_cur != S:
        s_cur = 0
        j = i
        while j < n and s_cur < S:
            s_cur += L[j]
            j += 1
        i += 1
    return s_cur == S
```

E) Séparation ...

Spécifications :

La fonction **separate(L1, L2, n)** retourne à partir de **L1** et **L2**, listes triées, le couple de listes triées **(R1, R2)** telles que :

- **R1** contient les éléments plus petits que **n**
- **R2** contient les éléments plus grands ou égaux à **n**

```
def fill_rest (L, start , R1 , R2 , x):
    n = len(L)
    while start < n and L[ start ] < x:
        R1. append (L[ start ])
        start += 1
    for i in range (start , len(L)):
        R2. append (L[i])

def separate (L1 , L2 , n):
    (R1 , R2) = ([], [])
    (i1 , i2) = (0, 0)
    (n1 , n2) = (len(L1), len(L2))
    while i1 < n1 and i2 < n2:
        if L1[i1] < L2[i2 ]:
            cur = L1[i1]
            i1 += 1
        else :
            cur = L2[i2]
            i2 += 1
        if cur < n:
            R1. append (cur)
        else :
            R2. append (cur)
    fill_rest (L1 , i1 , R1 , R2 , n)
    fill_rest (L2 , i2 , R1 , R2 , n)
    return (R1 , R2)
```