Recherche dichotomique

Exercice 1 : Recherche naïve

On a déjà écrit et implémenté un algorithme pour rechercher une valeur `v` dans une liste `l`. Plus précisément, on avait écrit une fonction `recherche_sequentielle(v, l)` qui renvoie `True` si `v` appartient à `l`, et `False` sinon.

Complétez les pointillés dans le code suivant qui implémente cette fonction recherche.

```
def recherche_sequentielle(v, l):
    """Renvoie True si v appartient à la liste l, False sinon."""
    for elt in l:
        if ............:
            return ............
    return ............
```

Si la liste `l` a pour taille n, combien de valeurs de la liste sont parcourues dans le pire des cas avec cet algorithme ? Quel est ce pire des cas ?

Cet algorithme de recherche a donc un coût de l'ordre de n, on parle de coût linéaire. Peut-on faire mieux ? Autrement dit, peut-on effectuer une recherche sans parcourir tous les éléments de la liste ? La réponse est OUI, à condition que la liste soit triée !

Exercice 2 : Recherche dichotomique

Compléter la fonction recherche_dichotomique ci-dessous. Créer des assertions pour tester votre code.

```
def recherche_dichotomique(v, l):
    """renvoie True si v est dans la liste l, supposé trié, et False sinon."""
    debut = 0
    fin = len(l) - 1
    while debut <= fin:
        m = ............
        if v == l[m]:
            return ............
        elif v > l[m]:
            debut = m + 1
        else:
            fin = ............
    return ............
```

Efficacité de l'algorithme

Pour calculer le coût de cet algorithme on peut compter le nombre d'itérations de la boucle while, c'est-à-dire le nombre de valeurs de la liste `l` qui ont été examinées. On se place dans le pire des cas, donc dans le cas où la valeur `v` ne se trouve pas dans `l` puisque c'est ce qui va occasionner le plus grand nombre d'itérations.

Si on cherche de cette façon un nombre dans une liste triée de taille n = 100 000. Après 1 comparaison, on le cherche dans une liste d'au plus 50 000 nombres ; après 2 comparaisons, on le cherche dans un ensemble d'au plus 25 000 nombres ; etc. On arrive très vite au nombre cherché ou on constate qu'il ne se trouve pas dans la liste.
Ainsi le nombre maximal d'itérations correspond au nombre de fois qu'il faut diviser n par 2 pour obtenir un nombre inférieur ou égal à 1.

On voit qu'il s'agit d'un algorithme extrêmement efficace puisque même pour rechercher une valeur dans une liste de taille 1 milliard, il suffit d'examiner 30 valeurs dans le pire des cas : la recherche dichotomique sera donc instantanée. A titre de comparaison, avec l'algorithme de recherche séquentielle il faudrait examiner 1 milliard de valeurs dans le pire des cas ! Bien entendu, il faut garder à l'esprit que le tableau doit être trié pour faire une recherche dichotomique.

Le nombre d'itérations est égale au plus petit entier supérieur ou égal à $\log_2(n)$. On dit que la recherche dichotomique a une **complexité logarithmique**.