Chapitre 2

Algorithmes de recherche

2.1 Recherche dans une liste

Pour chercher un élément dans une liste, l'algorithme le plus simple consiste à parcourir chaque élément de la liste par son indice, et à le comparer à l'élément cherché. Si on trouve l'élément cherché, on retourne alors son indice, et si l'élément n'est pas trouvé, on retournera -1 :

L : liste de longueur n
e : élément à chercher
Pour i de 0 à $n-1$ **Faire**
  Si L[i] == e **Alors**
    Retourner i
  Fin Si
Fin Pour
Retourner *-1*

Question 01 Implémenter l'algorithme précédent en une fonction `recherche(element, liste)` qui retourne l'indice de `element` si `element` est dans `liste`, et `None` dans le cas contraire.

☞ *L'indice retourné est celui de la première occurence de l'élément.*

Question 02 Combien de comparaisons sont nécessaires pour trouver `element` dans `liste` si :

1. `liste = [1,2,3,4]` et `element = 3`
2. `liste = [0,2,1,4,5,2]` et `element = 2`
3. `liste = [2,8,3,9,11,0]` et `element = 5`

Question 03 Pour une liste de longueur n, combien de comparaisons sont nécessaires pour trouver l'élément cherché :

1. dans le meilleur des cas
2. dans le pire des cas

Donner un exemple dans chacun des cas.

Question 04 Si on se place dans le pire des cas, quelle est la complexité de l'algorithme de recherche ?

☞ *Calculer le nombre maximum de comparaisons pour des listes de longueurs n, $2n$ et $3n$, et comparer les résultats .*

2.2 Recherche dans une liste triée

2.2.1 Algorithme naïf

Dans le cas où la liste dans laquelle on recherche un élément est **triée**, on peut arrêter la recherche dès que l'élément lu est supérieur à la valeur cherchée.

L : liste de longueur n
e : élément à chercher
Pour i de 0 à $n-1$ **Faire**
  Si L[i] == e **Alors**
    Retourner i
  Sinon Si L[i] > e **Alors**
    Interrompre la boucle
  Fin Si
Fin Pour
Retourner *-1*

Question 05 Écrire une fonction `recherche(element, listeTriee)` qui retourne l'indice de `element` si `element` est dans la liste **triée** `listeTriee`, et `-1` dans le cas contraire.

On pourra utiliser le mot clé `break` pour interrompre une boucle `for`.

Question 06 Combien de comparaisons sont nécessaires pour trouver `element` dans `liste` si :

1. `liste = [1,2,3,4]` et `element = 3`
2. `liste = [0,1,2,2,4,5]` et `element = 2`
3. `liste = [0,2,3,8,9,11]` et `element = 5`

Question 07 Pour une liste de longueur n, combien de comparaisons sont nécessaires pour trouver l'élément cherché :

1. dans le meilleur des cas
2. dans le pire des cas

Donner un exemple dans chacun des cas.

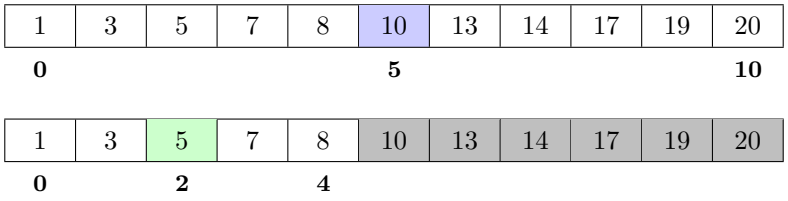
Question 08 L'algorithme est-il réellement plus performant dans le cas d'une liste triée ?

2.2.2 Recherche dichotomique

Dans le cas où la liste est triée, on dispose d'une méthode de recherche beaucoup plus efficace, la **recherche dichotomique**. Voici le principe :

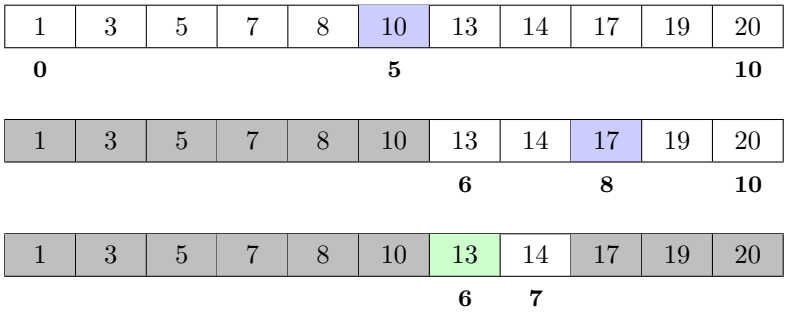
1. On compare l'élément central de la liste avec l'élément recherché.
2. Si c'est l'élément recherché, on s'arrête.
3. Si l'élément recherché est inférieur à l'élément central, on le recherche dans la première partie de la liste. Sinon, on le recherche dans la deuxième partie de la liste.
4. On recommence les étapes précédentes avec la demi-liste choisie à l'étape précédente.

Exemple. On cherche l'élément `5` dans la liste `L = [1, 3, 5, 7, 8, 10, 13, 14, 17, 19, 20]`.



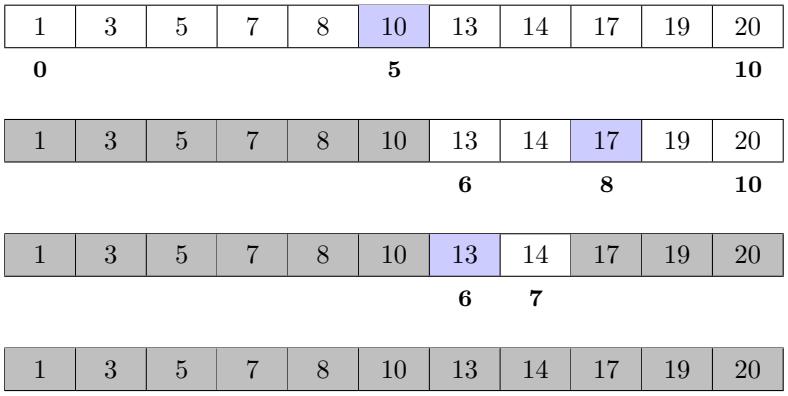
5 est bien dans la liste, et son indice est 2. Trois comparaisons (`10 == 5` , `5 < 10` , `5 == 5`) suffisent pour trouver ce résultat.

Exemple. On cherche l'élément `13` dans la liste `L = [1, 3, 5, 7, 8, 10, 13, 14, 17, 19, 20]`.



Lorsque la liste admet un nombre **pair** d'éléments, l'élément central est l'élément « à gauche »du centre (ce choix est arbitraire).
Ici, en cinq comparaisons, l'indice de 13 est trouvé : c'est 6.

Exemple. On cherche l'élément `11` dans la liste `L = [1, 3, 5, 7, 8, 10, 13, 14, 17, 19, 20]`.



Après 6 comparaisons, la « demi-liste »restante est vide : 11 n'appartient donc pas à la liste initiale.

Question 09 Combien de comparaisons sont nécessaires pour trouver `element` dans `liste` si :

1. `liste = [1,3,5,7,9]` et `element = 7` .
2. `liste = [2,7,8,10,10,11,15,20,21]` et `element = 15` .

Question 10 Quel est le nombre maximum de comparaisons nécessaires pour trouver l'élément cherché :

1. dans une liste de longueur 5 ?
2. dans une liste de longueur 10 ?
3. dans une liste de longueur 20 ?

Comparer ces résultats avec ceux obtenus dans le cas d'une recherche « classique ».

Question 11 On peut montrer que pour une liste de longueur n, le nombre maximum de comparaisons est égal à $2k$, où k est le plus petit entier vérifiant $n < 2^k$.

1. Quel est le nombre maximum de comparaisons pour :
   (a) une liste de longueur 100 ?
   (b) une liste de longueur 200 ?
   (c) une liste de longueur 400 ?
2. Comparer les trois résultats précédents.

☞ *Pour cet algorithme, on parle de complexité* ***logarithmique*** *: lorsqu'on multiplie la taille de la liste par 2, le nombre d'opérations élémentaires* ***augemente de 2*** *(une constante) ! On parle de complexité en* $O(\log(n))$.

Question 12 Implémenter l'algorithme précédent dans une fonction `recherche_dichotomique(element, listeTriee)` .

☞ *On pourra utiliser des variables représentant respectivement les indices gauche, central et droite de la « demi-liste »formée à chaque étape.*

```
"""
Implémentation de l'algorithme de recherche dichotomique
"""

def recherche_dichotomique(element, listeTriee):
    indice_debut = ...
    indice_fin = ...
    indice_milieu = ...

    while ... :
        if listeTriee[indice_milieu] == element:
            return indice_milieu
        elif ... :
            indice_debut = indice_milieu + 1
        else:
            indice_fin = ...

        indice_milieu = ...

    # Si l'élément n'est pas trouvé, on retourne -1

    return -1
```

Question 13

1. Générer une liste de 1 000 000 d'éléments aléatoires avec `liste = [random.randint(1,1e8) for _ in range(1000000)]` (sans ou blier d'importer `random`). Estimer le temps mis par Python pour générer cette liste, en utilisant le module `time` .
2. Trier cette liste avec `listeTriee = sorted(liste)` . Estimer le temps mis par Python pour trier cette liste, en utilisant le module `time` .
3. Recherche l'élément 1 dans cette liste, en utilisant `recherche(1, liste)` et `recherche_dichotomique(1, listeTriee)` . Comparer les temps de recherche.