Algorithmique
Recherches : séquentielles
Introduction

— On appelle **recherche associative** le fait que le critère de recherche ne porte que sur la valeur de la clé de l'élément recherché.
— La recherche est qualifée de :

positive lorsque la clé recherchée est présente dans la collection de données.

négative lorsque la clé recherchée est absente de la collection de données.

Les recherches présentées ici se font sur des listes.
Pour comparer les différentes méthodes, on considère comme opération fondamentale la comparaison de la clé x cherchée à une de celles de la liste. Les complexités seront exprimées en fonction de n, longueur de la liste.

1 Recherche séquentielle sur liste non triée

Principe :
On se place sur le premier élément de la liste.
Tant qu'il reste des éléments, et que l'élément courant **n'est pas** x, on avance à l'élément suivant.
Si la liste a été parcourue entièrement, la recherche est négative. Elle est positive sinon : l'élément sur lequel on s'est arrêté est celui cherché.

```
fonction rechercher(liste L, élément x) : booleen
variables
    entier i
debut
    i ← 1
    tant que i <= longueur(L) faire
        si x = ième(L, i) alors
            retourne vrai
        fin si
        i ← i + 1
    fin tant que
    retourne faux
fin
```

```
/* Autre version : */
    i ← 1
    tant que i <= longueur(L) et x <> ième(L, i) faire
        i ← i + 1
    fin tant que
    retourne (i <= longueur(L))
```

Complexité :
Recherche négative : il faut n comparaisons dans tous les cas.
Recherche positive : Au pire des cas, l'élement est en fin de liste, il faut n comparaisons. Sinon, c'est la position de l'élément : sur un grand nombre de recherches, on peut supposer que le nombre de comparaisons sera en moyenne $n/2$.

Modification : La recherche auto-adaptative

On peut améliorer la recherche positive en faisant en sorte que les éléments les plus fréquemment recherchés soient en début de liste en faisant avancer vers les premières places les éléments au fur et à mesure de leur recherche : la **recherche auto-adaptative**. Il en existe trois variantes possibles :

L'agressive :
Chaque fois que l'on trouve un élément, on le place directement en tête de liste.

La molle :
Chaque fois que l'on trouve un élément, on le fait progresser d'une place vers la tête de liste.

La plus raisonnable :
Chaque fois que l'on trouve un élément, on le fait progresser d'un certain nombre de places vers la tête de liste.

2 Recherche séquentielle sur liste triée

Les cas de recherches négatives peuvent être améliorés lorsque la liste est triée : il est inutile de continuer la recherche si la valeur cherchée a été dépassée.

Principe :

On se place sur le premier élément de la liste.
Tant qu'il reste des éléments, et que l'élément courant **n'a pas dépassé** x, on avance à l'élément suivant.
On a trouvé l'élément si on n'a pas parcouru toute la liste et si l'élément sur lequel on s'est arrêté est x.

```
fonction rechercher(liste L, élément x) : booleen
variables
    entier i
debut
    i ← 1
    tant que i <= longueur(L) et x >= ième(L, i)faire
        si x = ième(L, i) alors
            retourne vrai
        fin si
        i ← i + 1
    fin tant que
    retourne faux
fin
```

```
/* Autre version : */
    i ← 1
    tant que i <= longueur(L) et x > ième(L, i) faire
        i ← i + 1
    fin tant que
    retourne (i <= longueur(L) et x = ième(L, i))
```

Complexité :

Que la recherche soit positive ou négative, la complexité est la même [1] :
— toujours n dans le pire des cas ;
— $n/2$ dans le cas moyen.

Conclusion

Les différentes versions de la recherche séquentielle présentées ici peuvent s'implémenter avec des listes contiguës ou chaînées, mise à part la recherche auto-adaptative agressive, qui sur une liste contiguë nécessite des décalages coûteux.

Complexité linéaire

Dans le pire des cas, il faut toujours n comparaisons entre la valeur cherchée x et les éléments de la liste. Dans le cas moyen (en supposant que sur un grand nombre de recherches, on a autant de chances d'atteindre chacune des positions dans la liste) :

	positive	*négative*
non triée	$n/2$	n
triée	$n/2$	$n/2$

Même si la recherche dans une liste triée est légèrement plus efficace, cela reste en $O(n)$. Sur un grand nombre d'éléments, il est conseillé d'utiliser des recherches plus efficaces !

1. À noter que la première version fait 2 comparaisons par itération, alors que la deuxième n'en fait qu'une...