Fiche d'identité de l' algorithme de **RECHERCHE SÉQUENTIELLE**

Principe : on cherche v dans une liste , on ….....→ renvoie VRAI si v est trouvé, faux sinon

Exemple : feuilleter un livre page après page depuis le début pour retrouver une information

Pseudo Code

….....

.....

.....

.....

.....

Python

```
def chercher1(v,list):          #recherche la présence de l'entier v dans la liste "list"
    for i in list:              # on parcourt la liste
        if i==v:                # v est trouvé dans la liste
            return True         # on indique qu'on a trouvé l'élément
    return False                # la liste est parcourue : v ne lui appartient pas

def chercher2(v,list):          #recherche la présence de l'entier v dans la liste "list"
    if v in list:               # on parcourt la liste
        return True             # v est trouvé dans la liste et on indique qu'on a trouvé l'élément
    return False                # la liste est parcourue : v ne lui appartient pas
```

TERMINAISON de l'algorithme

Nous allons nous intéresser au fait que cet algorithme s'arrête....en effet, prouver ici que l'algorithme remplit bien son rôle (Correction de l'algorithme) est « trivial » (mais ce ne sera pas le cas des algorithmes de tri!)

Pour prouver qu'un algorithme s'arrête, on choisit une variable et on vérifie que la suite formée par les valeurs de cette variable au cours des itérations converge en un nombre fini d'étapes vers une valeur satisfaisant la condition d'arrêt. Cette variable s'appelle un **variant de boucle.**

Dans ce cas précis, on peut choisir la suite du nombre d'éléments $(n_i)_{i\in\mathbb{N}^*}$ de la liste qui ….....

* Supposons que $n_1 = 0$; la liste est …....., on n'entre pas dans la boucle et ….....est retourné (l'algorithme ….....)

Si $n_1>0$ et vaut la longueur de la liste de départ : on entre dans la boucle, on compare v au 1er élément ; soit il est égal à v et on ….....de la boucle en renvoyant vrai(et l'algorithme .....) ; soit il n'est pas égal à v et n_2 = ….....

* A la $k^{ième}$ itération, trois possibilités sont présentes :
  - soit v est égal au $k^{ème}$ terme : on sort de la boucle en renvoyant ….....(et l'algorithme se termine)
  - soit v n'est pas égal au $k^{ème}$ terme et $n_{k+1} = n_k - 1 = n_{k-1} - 2 = n_{k-2} - 3 = \ldots.. = n_1 - k + 1 = n_1 - k + 1$
  - soit $n_k = 0$, tous les éléments de la liste ont été comparés à v sans succès, on sort de la boucle et …..... est retourné (l'algorithme se termine donc)

* La suite strictement décroissante $(n_i)_{i\in\mathbb{N}^*}$ est minorée par ….....: elle converge donc (on décrémente à chaque tour ce nombre de une unité)

Complexité en temps : Dans le pire des cas (celui où v n'appartient pas à la liste) , on parcourt l'ensemble de la liste de longueur n *:* on effectue n comparaisons, le temps de calcul de la fonction EST PROPORTIONNEL à la longueur de la liste. On parle d'une complexité en **O(n)** (lire « grand o de n ») : c'est un **coût linéaire**

Analyser l'algorithme suivant :

v ← élément cherché *#valeur du nombre entier cherché*

deb ← indice du premier élément de la liste

fin ← indice du dernier élément de la liste
r ← Faux *#Réponse initialisée à Faux*
i ← deb
Tant que i<= fin et que r=Faux faire
    Si list[i] = v alors
        r ← Vrai
    FinSi
    i ← i+1
FinTant que
Renvoyer la valeur de r

puis le faire tourner « à la main » sur la liste suivante comme dans l'exemple

`list=[19,2,81,70,7,97,85,26,45,86]` en recherchant 2 d'abord puis 77 ensuite.

Exemple avec 81 _____*v=81* *r=Faux* *deb=0 et fin= 9* *1 – list[0]= 19 ≠ v alors i=0+1 = 1* *2- list[1]= 2 ≠ v alors i=1+1 = 2* *3- list[2]= 81 = v alors r=Vrai et i=2+1 = 3* *4- on renvoie Vrai* ***A VOUS*** _____*v=2*	_____*v=77*

Complexité en temps

Combien de tours de boucles ont été nécessaires pour chercher 81 ? 2 ? et 77 ?

Afin d'estimer la complexité de cet algorithme, on se place dans le « pire des cas » , c'est à dire le plus grand nombre de tours de boucle possible : décrivez les conditions de cette situation pour un nombre v donné et une liste donnée…….

Dans ce pire des cas,combien d'étapes seront nécessaires pour chercher v dans une liste de longueur n?

Exercices

Exercice 1 – *CONCEVOIR ET TRADUIRE*

Écrire un algorithme qui compte le nombre d'éléments dans une liste , ce sans faire appel à une fonction dédiée (comme *len( )* en Python)
Indiquer la complexité temporelle si la liste contient n entiers.
Implémenter cet algorithme en Python

Exercice 2 – *CONCEVOIR ET TRADUIRE*

1°) Écrire un algorithme qui permette de trouver le minimum d'une liste d'entiers. Pour cela, on peut parcourir les éléments de la liste et utiliser mini, une variable initialisée au premier élément de la liste et n le nombre d 'éléments de la liste
Estimer le coût en temps d'un telle recherche
Implémenter cet algorithme en Python

2°) Améliorer cet algorithme en *minimum_1indice(liste)* pour qu'il renvoie aussi le 1er indice du minimum rencontré
A FAIRE ABSOLUMENT **Créer un algorithme *minimum_indice(liste,j)* qui sera utile pour le tri séquentiel : il prend en paramètres la liste et un rang pour renvoyer l'indice du du minimum de la liste comprise entre les indices j et n-1**

3°) Créer enfin un algorithme *minimum_indiceS(liste)* qui renvoie TOUS les indices de chacune des occurrences du minimum dans la liste.
Estimer le coût en temps d'une telle recherche

Exercice 3 – *TRADUIRE ET DÉVELOPPER*

On associe des numéros à des personnages :

Personnage	Numéro
Celine	1
Lucas	2
Stephane	3
Louison	4
Jean	5

Ceci se traduit en Python par la liste :
Pers = [['Celine',1],['Lucas',2],['Stephane',3],['Louison',4],['Jean',5]]
Écrire une fonction qui permet de rechercher si un personnage de numéro v (un entier) quelconque est présent dans la liste : la fonction renverra le nom du personnage alors et *False* sinon

Exercice 4 – *ANALYSER ET CONCEVOIR*

On prétend que la fonction suivante teste l'appartenance de la valeur v au tableau t

```
def appartient(v,t):
    for i in range(len(t)):
        if t[i] == v:
            trouvee = True
        else :
            trouvee = False
    return trouvee
```

Donner des tests pour cette fonction et en particulier des tests montrant plusieurs raisons pour laquelle cette fonction est incorrecte

Que teste cette fonction en réalité ?

Aller plus loin....et que pensez vous de

```
def appartient(v,t):
    for i in range(len(t)):
        if t[i] == v:
            trouvee = True
        else :
            trouvee = False
        return trouvee
```