1 Recherche d'un élément

1.1 Parcours séquentiel d'un tableau par élément

```
Fonction Recherche(e, tab)
/* Renvoie un booléen indiquant si e est dans tab. */
Entrées :
tab : tableau
e : élément recherché
Sorties : Vrai ou Faux
pour chaque element de tab faire
    si element = e alors
        renvoyer Vrai            /* quitte la fonction instantanément */
    fin si
fin pour chaque
renvoyer Faux
```

Voici par exemple une implémentation en Python :

```python
def cherche(tab, e):
  """ Recherche si l'élément e est présente dans le tableau tab. """
  for element in tab:
    if element == e:
      return True          # sortie de fonction anticipée
  return False
```

Voici une variation mineure qui n'exploite pas la sortie anticipée d'une fonction :

```
Fonction Recherche(e, tab)
trouvé ← Faux
pour chaque element de tab faire
    si element = e alors
        trouvé ← Vrai
        fin boucle /* quitte la boucle prématurément */
    fin si
fin pour chaque
renvoyer trouvé
```

```python
def cherche(tab, e):
  """ Recherche si l'élément e est présente dans le tableau tab. """
  trouve = False
  for element in tab:
    if element == e:
      trouve = True
      break          # termine la boucle for prématurément
  return trouve
```

1.2 Parcours séquentiel d'un tableau par indice

```
Fonction Recherche(e, tab)
/* Renvoie un booléen indiquant si e est dans tab. */
Entrées :
tab : tableau
e : élément recherché
Sorties : Vrai ou Faux
n ← taille(tab)
/* les indices d'un tableau commencent à 0 et terminent à n-1 */
pour i de 0 à n-1 faire
    si tab[i] = e alors
        renvoyer Vrai                  /* quitte la fonction instantanément */
    fin si
fin pour
renvoyer Faux
```

Voici par exemple une implémentation en Python :

```python
def cherche(tab, e):
    """ Recherche si l'élément e est présente dans le tableau tab. """
    for i in range(len(tab)):   # range(n) crée une séquence d'entiers de 0 à n-1
        if tab[i] == e:
            return True           # sortie anticipée
    return False
```

Exercice : écrire une variation qui n'exploite pas la sortie anticipée d'une fonction.

1.3 Obtenir l'indice de la 1ère occurrence de l'élément cherché

On propose de modifier légèrement ce code pour que la fonction renvoie l'indice de l'élément dans le tableau s'il est présent ou -1 (code arbitraire) dans le cas contraire.

```
Fonction cherchePosition(e, tab)
/* Renvoie l'indice de 1ère occurrence de l'élément e s'il est présent dans
   tab, ou -1 en cas d'échec. */
Entrées :
tab : tableau
e : élément recherché
Sorties : indice entier
n ← taille(tab)
pour i de 0 à n-1 faire
    si tab[i] = e alors
        renvoyer i                     /* quitte la fonction instantanément */
    fin si
fin pour
renvoyer -1                                            /* e absent de tab */
```

Voici par exemple une implémentation en Python :

```python
def cherche_position(tab, e):
  """ Renvoie l'indice de 1ère occurrence de l'élément e s'il est présent
  dans le tableau tab, ou -1 dans le cas contraire """
  for i in range(len(tab)):
    if tab[i] == e:
      return i          # 1ère occurrence trouvée : on stoppe la recherche
  return -1
```

Exercice : indiquer ce qui différencie le code proposé ci-dessous ; donner un exemple d'arguments `e` et `tab` qui ne conduisent pas à la même réponse lors de l'appel à `cherche_position` ou `cherche_position2`.

```python
def cherche_position2(tab, e):
  indice = -1      # indice où on trouve e dans tab,
                   # initialisé à -1 car e n'est peut-être pas dans le tableau
  for i in range(len(tab)):
    if tab[i] == e:
      indice = i
  return indice
```

2 Recherche d'un extremum

```
Fonction maximum(tab)
  /* Renvoie la valeur maximum du tableau tab. */
  Entrées : tab : tableau non vide d'éléments comparables
  Sorties : maxi : valeur maxi
maxi ← tab[0]          /* maxi initialisé avec le 1er élément du tableau */
pour chaque element de tab faire
  si element > maxi alors
    maxi ← element
  fin si
fin pour chaque
renvoyer maxi
```

Voici par exemple une implémentation en Python :

```python
def maximum(tab):
  """ Recherche la valeur maximum dans le tableau tab. """
  # précondition : tableau non vide
  assert len(tab) > 0, "Le tableau ne doit pas être vide" # étudié plus tard...
  maxi = tab[0]
  for element in tab:
    if element > maxi:
      maxi = element
  return maxi
```

Exercice : adapter cet algorithme à la recherche d'un minimum.

3 Calcul d'une moyenne

```
Fonction moyenne(tab)
  /* Renvoie la valeur moyenne des éléments du tableau tab. */
  Entrées : tab : tableau de nombres
  Sorties : valeur moyenne
somme ← 0                          /* somme des éléments, initialisée à 0 */
pour chaque nombre de tab faire
  | somme ← somme + nombre
fin pour chaque
nb_elements ← taille(tab)
renvoyer somme / nb_elements
```

Voici par exemple une implémentation en Python :

```python
def moyenne(tab):
    """ Calcule la moyenne des valeurs dans le tableau tab """
    somme = 0
    for nombre in tab:
        somme = somme + nombre
    return somme / len(tab)
```

4 Coût des algorithmes

Lorsqu'on écrit un algorithme, il est intéressant (indispensable ?) de réfléchir au **coût** de cet algorithme **en fonction de la taille des données** à traiter.
Les algorithmes développés ici **parcourent un tableau de façon séquentielle** dans son intégralité (même pour la recherche d'occurrence dans *le pire des cas*).
Si on double la taille du tableau, on double donc le nombre d'opérations à effectuer dans l'algorithme, et donc on double aussi le temps de calcul.

> Le coût de ces algorithmes est donc proportionnel à la taille du tableau, on dit que le **coût** de ces algorithmes est **linéaire**.