1 Recherche d'une occurrence

Parcours séquentiel d'un tableau par élément :

```
def cherche(t, n):
  """ recherche si l'occurrence n est présente dans le tableau t """
  # précondition : t est un tableau de valeurs de type quelconque
  for element in t:
    if element == n:
      return True # occurrence trouvée : on stoppe la recherche
  return False
```

Parcours séquentiel d'un tableau par indice :

```
def cherche(t, n):
  """ recherche si l'occurrence n est présente dans le tableau t """
  # précondition : t est un tableau de valeurs de type quelconque
  for i in range(len(t)):
    if t[i] == n:
      return True # occurrence trouvée : on stoppe la recherche
  return False
```

On propose de modifier légèrement ce code pour que la fonction renvoie la position de l'occurrence dans le tableau si elle est présente ou -1 dans le cas contraire.

```
def cherche_position(t, n):
  """ renvoie la position si l'occurrence n si elle est présente
  dans le tableau t, ou -1 dans le cas contraire """
  # précondition : t est un tableau de valeurs de type quelconque
  for i in range(len(t)):
    if t[i] == n:
      return i # occurrence trouvée : on stoppe la recherche
  return -1
```

2 Recherche d'un extremum

```
def maximum(t):
  """ recherche la valeur maximum dans le tableau t """
  # préconditions : tableau non vide,
  # contenant des éléments tous comparables entre eux
  maxi = t[0] # valeur maxi par défaut
  for i in range(1, len(t)): # parcours par indice
    if t[i] > maxi:
      maxi = t[i]
  return maxi
```

On adaptera aisément cet algorithme à la recherche d'un minimum.

3 Calcul d'une moyenne

```
def cherche(t):
  """ calcule la moyenne des valeurs dans le tableau t """
  # précondition : le tableau contient des valeurs numériques
  somme = 0 # somme des valeurs du tableau (nulle a priori)
  for e in t: # parcours par élément
    somme = somme + e
  # la moyenne est la somme divisée par le nb de valeurs
  return somme / (len(t))
```

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.
Dans les exemples présentés, les algorithme développés parcourent un tableau de façon séquentielle dans son intégralité (même pour la recherche d'occurrence dans *le pire des cas*), élément par élément. 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.