Algorithmique 3 –

Différents algorithmes de parcours

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme première occurence

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme 1er occurrence

Algorithme à **coût linéaire** : **on lit parfois toutes les cases du tableau.**

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme 1er occurrence

Algorithme à **coût linéaire :** **on lit parfois toutes les cases du tableau.**

Meilleur des cas : x présent en case 0

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme première occurrence

Algorithme à **coût linéaire :** **on lit parfois toutes les cases du tableau.**

Meilleur des cas : x présent en case 0

Une seule comparaison quelque soit la taille n du tableau. Ce cas est toujours à coût …….

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme première occurrence

Algorithme à **coût linéaire :** **on lit parfois toutes les cases du tableau.**

Meilleur des cas : x présent en case 0.
Constant, en $\theta(1)$

Une seule comparaison quelque soit la taille n du tableau. Ce cas est toujours à coût constant.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme première occurrence

Algorithme à **coût linéaire :** **on lit parfois toutes les cases du tableau.**

Meilleur des cas : x présent en case 0.
Constant, en $\theta(1)$

Une seule comparaison quelque soit la taille n du tableau. Ce cas est toujours à coût constant.

Pire des cas : x n'est pas présent

Il doit comparer les n cases à x.
Ce cas est toujours à cout ….

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme 1er occurrence

Algorithme à **coût linéaire** : **on lit parfois toutes les cases du tableau.**

Meilleur des cas : x présent en case 0.
Constant, en $\theta(1)$

Une seule comparaison quelque soit la taille n du tableau. Ce cas est toujours à coût constant.

Pire des cas : x n'est pas présent
Linéaire, en $\theta(n)$

Il doit comparer les n cases à x.
Ce cas est toujours à cout …

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme 1er occurrence

Dans un cas quelconque, on a donc du linéaire ou moins puisqu'on peut avoir du constant.

On écrira donc que :
→ le coût de l'algorithme est linéaire
→ la complexité est en **O(n)**

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.1 Coût de l'algorithme 1er occurrence

Dans un cas quelconque, on a donc du linéaire ou moins puisqu'on peut avoir du constant.

On écrira donc que :
→ le coût de l'algorithme est linéaire
→ la complexité temportelle est en **O(n)**

Remarque : puisqu'on peut répondre avant d'avoir parcouru toutes les données, il s'agit bien d'un TANT QUE.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.2 Implémentation : définition ???

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 0 : Algo de la première occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du premier x trouvé, -1 si échec.
ALGORITHME :

```
i ← 0
TANT QUE i < longueur ET que t[i] ≠ x
    i ← i + 1
Fin TANT QUE
SI i < longueur
    reponse ← i
SINON
    reponse ← -1
Fin Si
Renvoyer reponse
```

1.2 Implémentation :

L'implémentation d'un algorithme consiste à le traduire dans un langage de programmation précis.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 1 : Implémentation A

```
1  def rechercher(t, x):
2      i = 0
3      longueur = len(t)
4      while i < longueur and t[i] != x:
5          i = i + 1
6      if i < longueur:
7          reponse = i
8      else:
9          reponse = -1
10     return reponse
```

1.3 Implémentation A(DOC 1)

Un vrai TANT QUE et une seule sortie.

Avantage :

Désavantage :

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 1 : Implémentation A

```
1  def rechercher(t, x):
2      i = 0
3      longueur = len(t)
4      while i < longueur and t[i] != x:
5          i = i + 1
6      if i < longueur:
7          reponse = i
8      else:
9          reponse = -1
10     return reponse
```

1.3 Implémentation A (DOC1)

Un vrai TANT QUE et une seule sortie.

Avantage :
La plus propre des trois.

Désavantage :
La plus « complexe » à mettre en place.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 2 : Implémentation B

```
1  def rechercher(t, x):
2      i = 0
3      longueur = len(t)
4      while i < longueur and t[i] != x:
5          i = i + 1
6      if i < longueur:
7          return i
8      return -1
```

1.4 Implémentation B (DOC 2)

Un vrai TANT QUE mais deux sorties.

Avantage :

Désavantage :

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 2 : Implémentation B

```
1  def rechercher(t, x):
2      i = 0
3      longueur = len(t)
4      while i < longueur and t[i] != x:
5          i = i + 1
6      if i < longueur:
7          assert sur la postcondition
8          return i
9      assert sur la postcondition
10     return -1
```

1.4 Implémentation B (DOC 2)

Un vrai TANT QUE mais deux sorties.

Avantage :
Un peu plus court.

Désavantage :
→ Preuve de correction plus diffcile à démontrer.
→ La surveillance de la postcondition est plus difficile : il faut placer des deux assert.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 3 : Implémentation C

```
1  def rechercher(t, x):
2      for i in range(len(t)):
3          if t[i] == x:
4              return i
5      return -1
```

1.5 Implémentation C (DOC 3)

Un TANT QUE réalisé avec un POUR associé à RETURN…

Avantage :

Désavantage :

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 2 : Implémentation C

```
1  def rechercher(t, x):
2      for i in range(len(t)):
3          if t[i] == x:
4              return i
5      return -1
```

1.5 Implémentation C

Un TANT QUE réalisé avec un POUR associé à RETURN…

Avantage :
Court.

Désavantage :
Pas « propre » car 2 sorties et on coupe le FOR.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

```
reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI t[i] est égal à x
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse
```

1.6 Coût de l'algorithme dernière occurrence

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

```
reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI t[i] est égal à x
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse
```

1.6 Coût de l'algorithme dernière occurrence

Principe :
→ la valeur par défaut dans **reponse**.
→ on lit dans les cases une à une. Si on trouve x, on place i dans **reponse**.
→ on renvoie **reponse**

Pas de meilleur ou pire des cas.

Le coût est toujours ???

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

```
reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI t[i] est égal à x
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse
```

1.6 Coût de l'algorithme dernière occurrence

Principe :
→ la valeur par défaut dans **reponse**.
→ on lit dans les cases une à une. Si on trouve x, on place i dans **reponse**.
→ on renvoie **reponse**

Pas de meilleur ou pire des cas.

Le coût est toujours linéaire.
On notera ???

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

```
reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI t[i] est égal à x
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse
```

1.6 Coût de l'algorithme dernière occurrence

Principe :
→ la valeur par défaut dans **reponse**.
→ on lit dans les cases une à une. Si on trouve x, on place i dans **reponse**.
→ on renvoie **reponse**

Pas de meilleur ou pire des cas.

Le coût est toujours linéaire.
On notera $\theta(n)$

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI `t[i]` est égal à `x`
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse

DOC 5 : Implémentation A

```
1  def rechercher(t, x):
2      reponse = -1
3      for i in range(len(t)):
4          if t[i] == x:
5              reponse = i
6      return reponse
```

1.7 Implémentation A (DOC 5)

Remarque importante :

Dans les algorithmes, la borne finale est une valeur incluse.
De 0 à 19 veut dire qu'on veut 0, 1 …,18, 19.

Dans le range de Python, la borne finale est une valeur exclue.
`range(20)` veut donc dire 0, 1, …, 18, 19.

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI `t[i]` est égal à `x`
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse

1.7 Implémentation A (DOC 5)

Fonction parfaite algorithmiquement :

DOC 5 : Implémentation A

```
1  def rechercher(t, x):
2      reponse = -1
3      for i in range(len(t)):
4          if t[i] == x:
5              reponse = i
6      return reponse
```

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 4 : Algo de la dernière occurrence de x dans t :

ENTREES : t un tableau, sa longueur et x un élément
SORTIE : Indice du dernier x trouvé, -1 si échec.
ALGORITHME :

reponse ← -1
POUR i de 0 (inclus) à longueur-1 (inclus)
    SI `t[i]` est égal à `x`
        reponse ← i
    Fin Si
Fin Pour
Renvoyer reponse

DOC 5 : Implémentation A

```
1  def rechercher(t, x):
2      reponse = -1
3      for i in range(len(t)):
4          if t[i] == x:
5              reponse = i
6      return reponse
```

1.7 Implémentation A (DOC 5)

Fonction parfaite algorithmiquent :

1) Boucle bornée et on a un FOR.

2) Un seul return

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

1.8 Implémentation B (DOC 6)

DOC 6 : Implémentation B

```
1  def rechercher(t, x):
2      reponse = -1
3      i = 0
4      while i < len(t):
5          if t[i] == x:
6              reponse = i
7          i = i + 1
8      return reponse
```

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 6 : Implémentation B

```
1  def rechercher(t, x):
2      reponse = -1
3      i = 0
4      while i < len(t):
5          if t[i] == x:
6              reponse = i
7          i = i + 1
8      return reponse
```

1.8 Implémentation B (DOC 6)

« Défaut » :
Boucle bornée dans l'idée mais WHILE.
Il faut donc penser à :
→ l'initialisation (L3) et
→ l'incrémentation (L7)

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 7 : Prédicat

```
1 def est_present(t, x):
2     return rechercher(t, x) >= 0
```

Algorithmique 3 – Différents algorithmes de parcours

I – Recherche linéaire dans un tableau

DOC 7 : Prédicat

```
1 def est_present(t, x):
2     return rechercher(t,x) >= 0
```

1.9 Fonction-prédicat (DOC 7)

Nous venons de récréer les instructions derrière

```
if x in t:
```

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 8 : Algorithme du maximum A :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
i ← 1
TANT QUE i < longueur
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
    i ← 1 + 1
Fin Pour
Renvoyer max_tempo
```

2.1 Algo max avec un Tant Que (DOC 8)

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 8 : Algorithme du maximum A :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
i ← 1
TANT QUE i < longueur
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
    i ← 1 + 1
Fin Pour
Renvoyer max_tempo
```

2.1 Algo max avec un Tant Que (DOC 8)

Principe en 3 étapes :
1) le maximum est l'élément dans la case 0.
2) On lit une par une les autres cases et on remplace maximum si on trouve plus
3) On renvoie maximum

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 8 : Algorithme du maximum A :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
i ← 1
TANT QUE i < longueur
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
    i ← 1 + 1
Fin Pour
Renvoyer max_tempo
```

2.1 Algo max avec un Tant Que (DOC 8)

→ *Le coût **linéaire**.*

→ *pas de pire ou meilleur des cas.*

→ *on écrira* $\theta(n)$

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 8 : Algorithme du maximum A :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
i ← 1
TANT QUE i < longueur
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
    i ← 1 + 1
Fin Pour
Renvoyer max_tempo
```

2.1 Algo max avec un Tant Que

→ Utiliser un TANT QUE complique l'algorithme : puisqu'on doit lire toutes les cases, nous aurions plutôt dû prendre un POUR.

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

2.2 Algo max avec un Pour (DOC 9)

DOC 9 : Algorithme du maximum B :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
POUR i de 1 (inclus) à longueur-1 (inclus)
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
Fin Pour
Renvoyer max_tempo
```

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 9 : Algorithme du maximum B :

ENTREES : t un tableau (et sa longueur)
Précondition 1 : tableau t NON VIDE
Précondition 2 : on peut utiliser > sur les éléments
SORTIE : L'élément maximum du tableau.

ALGORITHME :

```
max_tempo ← t[0]
POUR i de 1 (inclus) à longueur-1 (inclus)
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
Fin Pour
Renvoyer max_tempo
```

2.2 Algo max avec un Pour (DOC 9)

Plus simple car correspond bien à une lecture totale systématique.

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

ALGORITHME :

max_tempo ← t[0]
POUR i de 1 (inclus) à **longueur-1** (inclus)
    SI t[i] est supérieure à max_tempo
        max_tempo ← t[i]
    Fin Si
Fin Pour
Renvoyer max_tempo

DOC 10 : Implémentation

```
1   def rechercher_maximum(t):
2       max_temp = t[0]
3       for i in range(1, len(t)):
4           if t[i] > max_temp:
5               max_temp = t[i]
6       return max_temp
```

2.3 Implémentation (DOC 10)

Seul point à surveiller : la valeur limite.

→ elle est incluse dans les algorithmes (d'où un à longueur-1)

→ elle est exclue lorsqu'on utilise un range avec Python. D'où l'utilisation de :
`for i in range(1, len(t)):`

La variable de boucle i n'ira bien que jusqu'à len(t) – 1 !

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 11 : Fonction indice_maximum

```
1  def rechercher_imax(t):
2      imax = 0
3      for i in range(1, len(t)):
4          if t[i] > t[imax]:
5              imax = i
6      return imax
```

2.4 Indice maximal (DOC 11)

Cette fois, on ne stocke pas la valeur mais juste la position du maximum.

Principe en 3 étapes :

II – Recherche du maximum et du minimum

DOC 11 : Fonction indice_maximum

```
1  def rechercher_imax(t):
2      imax = 0
3      for i in range(1, len(t)):
4          if t[i] > t[imax]:
5              imax = i
6      return imax
```

2.4 Indice maximal (DOC 11)

Cette fois, on ne stocke pas la valeur mais juste la position du maximum.

Principe en 3 étapes :
1) l'indice imax est l'indice 0.
2) On lit une par une les autres cases et on modifie imax si la case contient plus que la case du maximum.
3) On renvoie imax.

Cout linéaire en $\theta(n)$ *(puisqu'on lit toutes les cases nécessairement).*

Algorithmique 3 – Différents algorithmes de parcours

II – Recherche du maximum et du minimum

DOC 11 : Fonction indice_maximum

```
1  def rechercher_imax(t):
2      imax = 0
3      for i in range(1, len(t)):
4          if t[i] > t[imax]:
5              imax = i
6      return imax
```

2.4 Indice maximal (DOC 11)

Rien de difficile sauf si on confond :

→ *le tableau : t*

→ *l'indice d'une case du tableau : i*

→ *la valeur contenue dans cette case : t[i]*

II – Recherche du maximum et du minimum

```
1  def rechercher_maximum(t):
2      max_temp = t[0]
3      for i in range(1, len(t)):
4          if t[i] > max_temp:
5              max_temp = t[i]
6      return max_temp
```

2.5 Trouver le minimum ?

Exactement comme le maximum mais on ne cherche pas à remplacer si c'est plus grand mais au contraire si la case est plus petite que l'élément en mémoire.

A part le nom éventuel des variables, la seule différence tient donc à l'opérateur < qui va remplacer l'opérateur > !

Il s'agit encore d'un algorithme linéaire en $\theta(n)$ *(puisqu'on lit toutes les cases nécessairement).*

Algorithmique 3 – Différents algorithmes de parcours

III – Recherche d'une somme et d'une moyenne

DOC 12 : calculer une somme

```
1  def calculer_somme(t):
2      s = 0
3      for i in range(len(t)):
4          s = s + t[i]
5      return s
```

3.1 Somme (DOC 12)

Principe en 3 étapes :

Algorithmique 3 – Différents algorithmes de parcours

III – Recherche d'une somme et d'une moyenne

DOC 12 : calculer une somme

```
1  def calculer_somme(t):
2      s = 0
3      for i in range(len(t)):
4          s = s + t[i]
5      return s
```

3.1 Somme (DOC 12)

Principe en 3 étapes :
1) On initialise la somme s à 0.
2) Pour chaque case du tableau, on rajoute la valeur de la case actuelle t[i] à la somme.
3) On renvoie la somme.

Il s'agit encore d'un algorithme linéaire en $\theta(n)$

Algorithmique 3 – Différents algorithmes de parcours

III – Recherche d'une somme et d'une moyenne

DOC 12 : calculer une somme

```
1  def calculer_somme(t):
2      s = 0
3      for i in range(len(t)):
4          s = s + t[i]
5      return s
```

DOC 13 : calculer une somme v2

```
1  def calculer_somme(t):
2      s = 0
3      for v in t:
4          s = s + v
5      return s
```

3.2 Somme (version 2) (DOC 13)

On lit les cases sans modification : on peut utiliser cette forme de boucle FOR.

for i in range(len(t)) → *indice*
for v in t → *valeur directement*

Algorithmique 3 – Différents algorithmes de parcours

III – Recherche d'une somme et d'une moyenne

DOC 12 : calculer une somme

```
1  def calculer_somme(t):
2      s = 0
3      for i in range(len(t)):
4          s = s + t[i]
5      return s
```

3.3 Moyenne (DOC 14)

Une fonction – une tâche.
*Appel à **calculer_somme()** pour la somme.*
*Appel à **len()** pour le nombre de cases.*

***Précondition : t NON VIDE** pour éviter de diviser par 0 tout simplement.*

DOC 14 : Valeur moyenne si t NON VIDE

```
1  def calculer_moyenne(t):
2      return calculer_somme(t) / len(t)
```

Coût linéaire car :
calculer_somme() est linéaire
len() est linéaire ou constant