Algorithmique et programmation structurée

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Algorithmique et programmation structurée

Algorithme : origine

Latinisation du nom du mathématicien perse **Al Khawarizmi** (né dans les années 780 après J.C. et décédé dans les années 850 après J.C.).

Algorithmique et programmation structurée

Algorithme : origine

Latinisation du nom du mathématicien perse **Al Khawarizmi** (né dans les années 780 après J.C. et décédé dans les années 850 après J.C.).

Algorithme : définition

- Suite finie et ordonnée d'instructions (actions) permettant de résoudre un problème.
- Écrit en langage naturel indépendant de tout langage de programmation.

Algorithmique et programmation structurée

Instruction : définition

- Ordre donné à la machine
- Action du processeur

Algorithmique et programmation structurée

Instruction : définition

- Ordre donné à la machine
- Action du processeur

Langage de programmation

- Ensemble de notations conventionnelles destinées à traduire des algorithmes en programmes compréhensibles par la machine.
- Ayant un ensemble de mots-clés et règles à respecter (Similaire aux langues).

Algorithmique et programmation structurée

Programme (informatique) : définition

Ensemble d'instructions, écrites en respectant un langage de programmation, destinées à être exécutées par un ordinateur.

Algorithmique et programmation structurée

Programme (informatique) : définition

Ensemble d'instructions, écrites en respectant un langage de programmation, destinées à être exécutées par un ordinateur.

L'exécution peut se dérouler de deux manières différentes

- S'il s'agit d'un langage **Compilé** : le compilateur lit le programme en entier et le transforme en instructions machines (binaires).
- S'il s'agit d'un langage **Interprété** : l'interpréteur opère ligne par ligne : lit une ligne de programme, puis exécute immédiatement les instructions machines correspondantes.

Algorithmique et programmation structurée

Compilation : avantage

Plus rapide.

Compilation : inconvénient

Recompiler après chaque modification.

Algorithmique et programmation structurée

Compilation : avantage

Plus rapide.

Compilation : inconvénient

Recompiler après chaque modification.

Interprétation : avantage

Identification d'erreur plus facile.

Interprétation : inconvénient

Exécution 10 à 100 fois plus lente.

Algorithmique et programmation structurée

Algorithmique : définition

La discipline qui étudie les algorithmes et leurs applications en Informatique.

Algorithmique et programmation structurée

Algorithmique : définition

La discipline qui étudie les algorithmes et leurs applications en Informatique.

Remarque

Le terme **algorithmie** est souvent utilisé comme synonyme de l'**algorithmique**.

Algorithmique et programmation structurée

Deux façons pour représenter un algorithme

- **Organigramme** (graphique) : en utilisant des symboles comme les carrés, les losanges...
- **Pseudo-code** (textuel) : en utilisant des phrases de langue parlée sans les problèmes de syntaxe comme en programmation.

Algorithmique et programmation structurée

Deux façons pour représenter un algorithme

- **Organigramme** (graphique) : en utilisant des symboles comme les carrés, les losanges...
- **Pseudo-code** (textuel) : en utilisant des phrases de langue parlée sans les problèmes de syntaxe comme en programmation.

Remarque

- Aujourd'hui, l'organigramme est quasiment abandonné.
- Dans ce cours, nous utilisons donc le pseudo-code.

Algorithmique et programmation structurée

Algorithme : structure

- Nom de l'algorithme
- Déclaration de variables
- Instructions de l'algorithme

Algorithme nom_algorithme
Variables : liste de variables + types
Début
    instructions
Fin

Algorithmique et programmation structurée

Variable

- Concept utilisé en programmation pour stocker des valeurs.
- Référence vers un espace mémoire
- Pouvant avoir les caractéristiques suivantes :
  - nom
  - type
  - valeur
  - adresse
  - portée
- Pouvant changer de valeur (mais pas de type)

Algorithmique et programmation structurée

Déclaration de variables

- Les variables sont déclarées au tout début de l'algorithme.
- Une variable non déclarée ne peut être utilisée.

Algorithmique et programmation structurée

Règles de nommage de variables

- Le nom d'une variable doit être significatif.
- Le nom d'une variable doit commencer par une lettre.
- Il ne peut contenir que des chiffres, lettres et underscores.
- Il ne peut contenir des mots-clés (comme `si`, `alors`, `var`...).

Algorithmique et programmation structurée

Noms corrects

- `formation`
- `poe_javascript`
- `FormationJava2020`

Noms incorrects

- `206Peugeot`
- `Peugeot 206`
- `for`

Algorithmique et programmation structurée

Typage de variables : pourquoi

- Déterminer l'ensemble de valeurs qu'elle peut prendre.
- Préparer le nombre d'octet qu'il faut lui réserver.
- Charger les fonctions applicables sur ce type.

Algorithmique et programmation structurée

Types considérés dans ce cours

- nombre
- booléen
- chaîne de caractères

Algorithmique et programmation structurée

Types considérés dans ce cours

- nombre
- booléen
- chaîne de caractères

Noms incorrects

Toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration préalable.

Algorithmique et programmation structurée

Pour déclarer une variable

```
Variables OK : booléen
```

Algorithmique et programmation structurée

Pour déclarer une variable

```
Variables OK : booléen
```

Pour déclarer plusieurs variables de même type

```
Variables i, j, k : nombre
```

Algorithmique et programmation structurée

Pour déclarer une variable

```
Variables OK : booléen
```

Pour déclarer plusieurs variables de même type

```
Variables i, j, k : nombre
```

Pour déclarer plusieurs variables de type différent

```
Variables i, j, k : nombre
          OK : booléen
          ch1, ch2 : chaîne de caractères
```

Algorithmique et programmation structurée

Pour affecter une valeur à une variable, on utilise l'opérateur <−

```
i <- 0
```

Algorithmique et programmation structurée

Pour affecter une valeur à une variable, on utilise l'opérateur <−

```
i <- 0
```

On peut aussi affecter la valeur d'une variable à une autre

```
j <- i
```

Algorithmique et programmation structurée

Pour affecter une valeur à une variable, on utilise l'opérateur <−

```
i <- 0
```

On peut aussi affecter la valeur d'une variable à une autre

```
j <- i
```

On peut également affecter le résultat d'une expression à une variable

```
k <- i + j
```

Algorithmique et programmation structurée

Remarques

- En programmation, la plupart des langages utilisent l'opérateur `=` pour l'affectation. Quelques autres utilisent `:=`.
- l'affectation n'est pas commutative : `A = B` est différente de `B = A`.
- `A + 1 = 2` n'est pas possible en algorithmique ni en programmation et n'est pas équivalente à `A = 1`.
- Certains langages donnent des valeurs par défaut aux variables déclarées. Pour éviter toute surprise, il est conseillé d'initialiser les variables à la déclaration.

Algorithmique et programmation structurée

Premier algorithme

```
Algorithme premier_algorithme
Variables A, B, C : nombre
Début
  A <- 3
  B <- 7
  A <- B
  B <- A + 5
  C <- A + B
  C <- B - 1
Fin
```

Algorithmique et programmation structurée

Premier algorithme

```
Algorithme premier_algorithme
Variables A, B, C : nombre
Début
   A <- 3
   B <- 7
   A <- B
   B <- A + 5
   C <- A + B
   C <- B - 1
Fin
```

Question

Quelles sont les valeurs finales des variables A, B et C ?

Algorithmique et programmation structurée

Donnez les valeurs des variables A et B après exécution des instructions suivantes

```
Algorithme premier_exo
Variables A, B : nombre
Début
   A <- 1
   B <- 2
   A <- B
   B <- A
Fin
```

Algorithmique et programmation structurée

Donnez les valeurs des variables A et B après exécution des instructions suivantes

```
Algorithme premier_exo
Variables A, B : nombre
Début
   A <- 1
   B <- 2
   A <- B
   B <- A
Fin
```

Question

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

Algorithmique et programmation structurée

Exercice

Écrire un algorithme permettant d'échanger les valeurs de deux variables A et B.

Algorithmique et programmation structurée

Les expressions

- Une expression, en algorithmique, retourne toujours un résultat.
- Une expression est évaluée de gauche à droite tout en respectant les priorités.
- Types d'expression :
  - Arithmétique en utilisant des opérateurs arithmétiques
  - Logique en utilisant des opérateurs logiques ou de comparaison

Algorithmique et programmation structurée

Opérateurs arithmétiques

- `+` : addition
- `*` : multiplication
- `-` : soustraction
- `/` : division
- `div` : division euclidienne
- `%` ou `mod` : reste de la division
- `^` : puissance

Algorithmique et programmation structurée

x **contient la valeur** 5

```
x <- 2 + 3
```

Algorithmique et programmation structurée

`x` **contient la valeur** `5`

```
x <- 2 + 3
```

Un expression peut contenir plusieurs opérations arithmétiques évalués de gauche à droite

```
x <- 2 + 5 + 3
```

Algorithmique et programmation structurée

x contient la valeur 5

```
x <- 2 + 3
```

Un expression peut contenir plusieurs opérations arithmétiques évalués de gauche à droite

```
x <- 2 + 5 + 3
```

Attention à la priorité, x contient la valeur 17

```
x <- 2 + 5 * 3
```

Algorithmique et programmation structurée

`x` contient la valeur `5`

```
x <- 2 + 3
```

Un expression peut contenir plusieurs opérations arithmétiques évalués de gauche à droite

```
x <- 2 + 5 + 3
```

Attention à la priorité, `x` contient la valeur `17`

```
x <- 2 + 5 * 3
```

On peut aussi imposer un ordre de priorité différent en utilisant les parenthèses (`x` contient la valeur `21`)

```
x <- (2 + 5) * 3
```

Algorithmique et programmation structurée

Donnez les valeurs finales des variables A, B et C après exécution des instructions suivantes

```
Algorithme deuxième_exo
Variables A, B, C : nombre
Début
   A <- 3
   B <- 7
   C <- B % A
   B <- A + B * 2 - C
   A <- B - A
Fin
```

Algorithmique et programmation structurée

Exercice

Écrire un algorithme permettant d'échanger les valeurs de deux variables A et B sans utiliser de troisième variable.

Algorithmique et programmation structurée

Fonctions mathématiques pouvant être utilisées en algorithmique

- `racine_carrée()`
- `valeur_absolue()`
- `sinus`
- ...

Algorithmique et programmation structurée

Quelques opérations sur les chaînes de caractères

- `longueur(ch)` permet de récupérer le nombre de caractères de `ch`.
- `ch[i]` permet de récupérer le caractère situé à la position `i+ 1` dans `ch` (le premier caractère est d'indice `0`).
- `&` est l'opérateur de concaténation (fusion) pour les chaînes de caractères.

Algorithmique et programmation structurée

Lecture et écriture (entrée/sortie)

- Les instructions de lecture/écriture permettent à la machine de communiquer avec l'utilisateur.
- L'entrée standard est le clavier et la sortie standard est l'écran.
- L'opération de lecture est bloquante : L'exécution s'arrête lorsqu'on rencontre une instruction de lecture et ne se poursuit qu'après saisie d'une valeur.
- Avant de lire une variable, il est conseillé d'écrire un message à l'écran afin de prévenir l utilisateur de ce qu'il doit saisir.

Algorithmique et programmation structurée

Exemple avec opérations de lecture/écriture

```
Algorithme lecture_écriture
Variables A : nombre
Début
   écrire "saisir une valeur numérique"
   lire (A)
   écrire ("la valeur saisie est ", A)
Fin
```

Algorithmique et programmation structurée

Exemple avec opérations de lecture/écriture

```
Algorithme lecture_écriture
Variables A : nombre
Début
  écrire "saisir une valeur numérique"
  lire (A)
  écrire ("la valeur saisie est ", A)
Fin
```

On peut aussi fusionner les opérations de lecture

```
lire (A, B, C)
```

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande un nombre à l'utilisateur, puis calcule et affiche son double.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir son nom puis son prénom et qui récupère et affiche ensuite les initiales stockées dans une chaîne de caractères.

Algorithmique et programmation structurée

Structure conditionnelle

- Les structures conditionnelles servent à n'exécuter une instruction ou une séquence d'instructions que si une condition est vraie.
- Une **condition** doit toujours retourner une valeur booléenne (`vrai` ou `faux`).

Algorithmique et programmation structurée

Exécuter si une condition est vraie

```
Si (condition) alors
    ...
FinSi
```

Algorithmique et programmation structurée

Exécuter si une condition est vraie

```
Si (condition) alors
    ...
FinSi
```

Exemple

```
x <- 3
Si (x > 0) alors
   écrire (x, " est strictement positif")
FinSi
```

Algorithmique et programmation structurée

Exécuter si une condition est vraie

```
Si (condition) alors
    ...
FinSi
```

Exemple

```
x <- 3
Si (x > 0) alors
   écrire (x, " est strictement positif")
FinSi
```

Pour les conditions, on utilise les opérateurs de comparaison.

Algorithmique et programmation structurée

Opérateurs de comparaison

- $=$
- $\neq$
- $>$
- $<$
- $>=$
- $<=$

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier, puis affiche s'il s'agit d'un nombre pair.

Algorithmique et programmation structurée

Exécuter un premier bloc si une condition est vraie, un deuxième sinon

```
Si (condition1) alors
   ...
Sinon
   ...
FinSi
```

Algorithmique et programmation structurée

Exécuter un premier bloc si une condition est vraie, un deuxième sinon

```
Si (condition1) alors
   ...
Sinon
   ...
FinSi
```

Exemple

```
x <- 3
Si (x > 0) alors
  écrire (x, " est strictement positif")
Sinon
  écrire (x, " est négatif ou nul")
FinSi
```

Algorithmique et programmation structurée

Exercice

Modifier l'algorithme précédent pour afficher la parité du nombre saisi.

Algorithmique et programmation structurée

On peut enchaîner les conditions avec `sinon si` (et avoir un troisième bloc voire ... un nième)

```
Si (condition1) alors
    ...
Sinon Si (condition2) alors
    ...
Sinon
    ...
FinSi
```

Algorithmique et programmation structurée

Exemple

```
x <- -3
Si (x > 0) alors
  écrire (x, " est strictement positif")
Sinon Si (x < 0) alors
  écrire (x, " est strictement négatif")
Sinon
  écrire (x, " est nul")
FinSi
```

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui permet de résoudre une équation de second degré ($ax^2 + bx + c = 0$). L'utilisateur saisit des valeurs pour `a`, `b` et `c` et on lui affiche la ou les solutions s'il en existe.

Algorithmique et programmation structurée

Pour enchainer les conditions, on utilise les opérateurs logiques

- ET
- OU
- NON
- XOR (or exclusif)

Algorithmique et programmation structurée

Pour enchainer les conditions, on utilise les opérateurs logiques

- ET
- OU
- NON
- XOR (or exclusif)

Exemple

```
Si (condition1 ET NON(condition2) OU condition3) alors
   ...
FinSi
```

Algorithmique et programmation structurée

Résultats des opérations logiques

a	b	**NON**(a)	a **ET** b	a **OU** b	a **XOR** b
faux	faux	vrai	faux	faux	faux
faux	vrai	vrai	faux	vrai	vrai
vrai	faux	faux	faux	vrai	vrai
vrai	vrai	faux	vrai	vrai	faux

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres et qui affiche le signe du résultat de la multiplication sans la calculer.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir l'indice d'un mois (un nombre compris entre 1 et 12) et qui affiche le nombre de jours du mois associé.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir trois notes et qui calcule puis affiche la moyenne.

Algorithmique et programmation structurée

Une solution possible

```
Algorithme calcul_moyenne
Variables note1, note2, note3, somme, moyenne : nombre
Début
  écrire ("Saisir une première note")
  lire (note1)
  écrire ("Saisir une deuxième note")
  lire (note2)
  écrire ("Saisir une troisième note")
  lire (note3)
  somme <- note1 + note2 + note3
  moyenne <- somme / 3
  écrire ("Moyenne de notes saisies = ", moyenne)
Fin
```

Algorithmique et programmation structurée

Question

Et si on voulait calculer la moyenne de plusieurs notes (des dizaines voire des centaines), faudrait-il répéter `écrire` et `lire` autant de fois que nécessaire ?

Algorithmique et programmation structurée

Question

Et si on voulait calculer la moyenne de plusieurs notes (des dizaines voire des centaines), faudrait-il répéter `écrire` et `lire` autant de fois que nécessaire ?

Réponse

Non, on peut utiliser les **boucles**.

Algorithmique et programmation structurée

Structure itérative (boucle)

- Les structures itératives servent à répéter l'exécution d'un bloc d'instructions un certain nombre de fois.
- Trois types de boucles proposées :
  - **Tant que** : on répète des instructions tant qu'une certaine condition est vraie.
  - **Répéter jusqu'à** : on répète des instructions jusqu'à ce qu'une certaine condition soit vraie.
  - **Pour** : on répète des instructions ou avec compteur en faisant évoluer un compteur entre une valeur initiale et une valeur finale.

Algorithmique et programmation structurée

Boucle `Tant que` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
TantQue (condition)
   ...
FinTantQue
```

Algorithmique et programmation structurée

Boucle `Tant que` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
TantQue (condition)
   ...
FinTantQue
```


Remarque

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

Algorithmique et programmation structurée

Exemple

```
i <- 0
TantQue (i < 5)
  écrire (i)
  i <- i + 1
FinTantQue
```

Algorithmique et programmation structurée

Exemple

```
i <- 0
TantQue (i < 5)
  écrire (i)
  i <- i + 1
FinTantQue
```


Le résultat est

```
0
1
2
3
4
```

Algorithmique et programmation structurée

Exercice

En utilisant une boucle `TantQue`, réécrire l'algorithme précédent pour permettre à l'utilisateur de saisir trois notes et lui calculer puis afficher la moyenne.

Algorithmique et programmation structurée

La Boucle `Répéter ... jusqu'à` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
Répéter
   ...
Jusqu'à (condition)
```

Algorithmique et programmation structurée

La Boucle `Répéter ... jusqu'à` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
Répéter
   ...
Jusqu'à (condition)
```


Remarque

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

Algorithmique et programmation structurée

Exemple

```
i <- 0
Répéter
   écrire (i)
   i <- i + 1
Jusqu'à (i > 5)
```

Algorithmique et programmation structurée

Exemple

```
i <- 0
Répéter
   écrire (i)
   i <- i + 1
Jusqu'à (i > 5)
```

Le résultat est

```
0
1
2
3
4
5
```

Algorithmique et programmation structurée

Boucle `pour`

```
Pour compteur de initiale à finale  pas valeurdupas
   ...
FinPour
```

Algorithmique et programmation structurée

Boucle `pour`

```
Pour compteur de initiale à finale  pas valeurdupas
   ...
FinPour
```


Remarque

Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

Algorithmique et programmation structurée

Exemple (par défaut le pas = 1)

```
Pour i de 0 à 5
  écrire (i)
FinPour
```

Algorithmique et programmation structurée

Exemple (par défaut le pas = 1)

```
Pour i de 0 à 5
  écrire (i)
FinPour
```

Le résultat est

```
0
1
2
3
4
5
```

Algorithmique et programmation structurée

Exercice

En utilisant une boucle `pour`, écrire un algorithme qui permet d'afficher les nombres pairs compris entre 0 et 10.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui permet à l'utilisateur de saisir deux entiers x et y et qui calcule, en utilisant une boucle `pour`, puis affiche x^y.

Algorithmique et programmation structurée

Pour VS TantQue VS Répéter ... Jusqu'à

- On utilise la boucle `Pour` si et seulement si le nombre d'itérations est connu avant l'exécution de la boucle.
- Les boucles `TantQue` et `Répéter ... Jusqu'à` peuvent être utilisées que le nombre d'itération soit connu ou pas.
  - La boucle `Répéter ... Jusqu'à` exécute le bloc d'instructions au moins une fois même si la condition est fausse.
  - La boucle `TantQue` exécute le bloc d'instructions seulement si la condition est vraie.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui demande à l'utilisateur de saisir une suite de valeurs entières différentes de 0. À la fin, l'algorithme affiche le nombre de saisies positives.

Algorithmique et programmation structurée

Problème

Chaque fois que l'utilisateur saisissait une valeur la précédente est perdue (remplacée par la nouvelle).

Algorithmique et programmation structurée

Problème

Chaque fois que l'utilisateur saisissait une valeur la précédente est perdue (remplacée par la nouvelle).

Solution

Utiliser les tableaux.

Algorithmique et programmation structurée

Tableau

- Structure de données pouvant contenir plusieurs éléments de même type.
- À la déclaration d'un tableau, il faut préciser le type d'éléments et la taille (nombre d'éléments).
- Chaque élément est accessible via un indice unique (sa position dans le tableau).
- Le premier élément est d'indice 0.
- Une fonction prédéfinie `longueur` permet de récupérer la taille d'un tableau.

Algorithmique et programmation structurée

Pour déclarer un tableau de 5 nombres

```
Variables notes : tableau de 5 nombre
```

Algorithmique et programmation structurée

Pour déclarer un tableau de 5 nombres

```
Variables notes : tableau de 5 nombre
```

Pour affecter une valeur au premier élément du tableau

```
notes[0] <- 15
```

Algorithmique et programmation structurée

Pour déclarer un tableau de 5 nombres

```
Variables notes : tableau de 5 nombre
```

Pour affecter une valeur au premier élément du tableau

```
notes[0] <- 15
```

Pour afficher la taille du tableau

```
écrire (longueur(notes))
```

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui

1. permet de remplir un tableau de 5 éléments avec des valeurs saisies par l'utilisateur.
2. affiche tous les éléments.
3. calcule et affiche la somme de tous les éléments.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui

1. permet de remplir un tableau de 5 éléments avec des valeurs saisies par l'utilisateur.
2. permet à l'utilisateur de saisir une valeur à chercher dans le tableau.
3. affiche si la valeur saisie existe dans le tableau.

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui

1. permet de remplir un tableau de 5 éléments avec des valeurs saisies par l'utilisateur.
2. cherche et affiche la plus grande valeur du tableau (max).

Algorithmique et programmation structurée

Tableau à deux dimensions

- Équivalent d'une matrice en mathématiques.
- Tableaux avec des valeurs repérées par deux indices.
  - Premier indice : pour les lignes
  - Deuxième indice : pour les colonnes.

Algorithmique et programmation structurée

Pour déclarer un tableau à deux dimensions

```
Variables matrice : tableau de nombre de 3 lignes et 2 colonnes
```

Algorithmique et programmation structurée

Pour déclarer un tableau à deux dimensions

```
Variables matrice : tableau de nombre de 3 lignes et 2 colonnes
```

Pour affecter une valeur à l'élément situé à la première ligne et la deuxième colonne

```
matrice[0][1] <- 1
```

Algorithmique et programmation structurée

Pour déclarer un tableau à deux dimensions

```
Variables matrice : tableau de nombre de 3 lignes et 2 colonnes
```

Pour affecter une valeur à l'élément situé à la première ligne et la deuxième colonne

```
matrice[0][1] <- 1
```

Pour afficher respectivement la taille de la matrice, le nombre de lignes et le nombre de colonnes

```
écrire (longueur(matrice), ligne(matrice) colonne(matrice))
```

Algorithmique et programmation structurée

Exercice

Écrire un algorithme qui

1. permet de remplir une matrice carrée (nombre de lignes = nombre de colonnes) de taille 3.
2. affiche tous les éléments.
3. calcule et affiche la somme de tous les éléments.

Algorithmique et programmation structurée

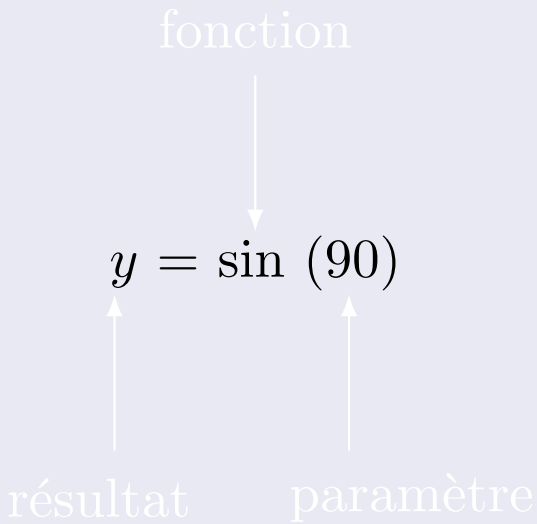
Exercice

Écrire un algorithme qui permet de

1. multiplier une matrice `A` par une constante α.
2. calculer la somme de deux matrices carrées `A` et `B`.
3. vérifier si une matrice `A` est diagonale.
4. calculer le produit de deux matrices carrées `A` et `B`.

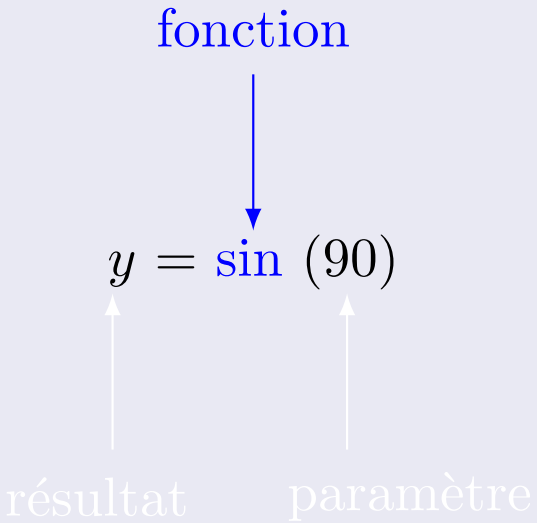
Algorithmique et programmation structurée

En mathématiques



Algorithmique et programmation structurée

En mathématiques

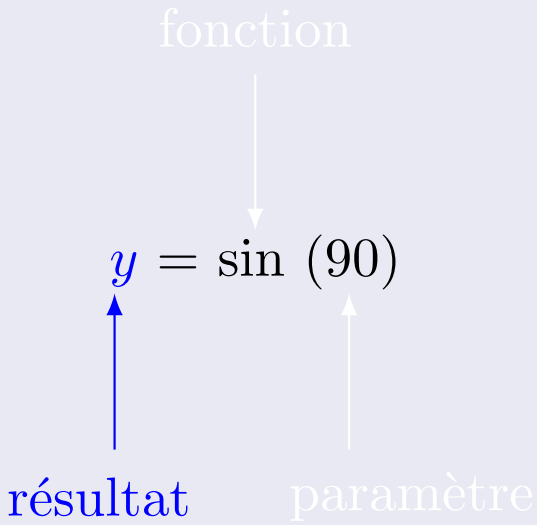


Remarque

En algorithmique aussi on peut utiliser et écrire des fonctions.

Algorithmique et programmation structurée

En mathématiques

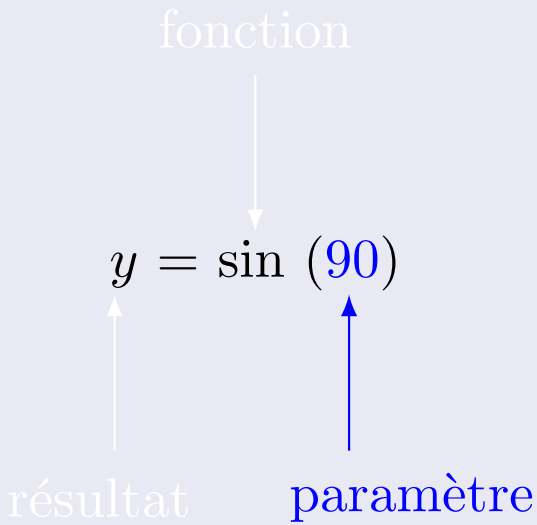


Remarque

En algorithmique aussi on peut utiliser et écrire des fonctions.

Algorithmique et programmation structurée

En mathématiques

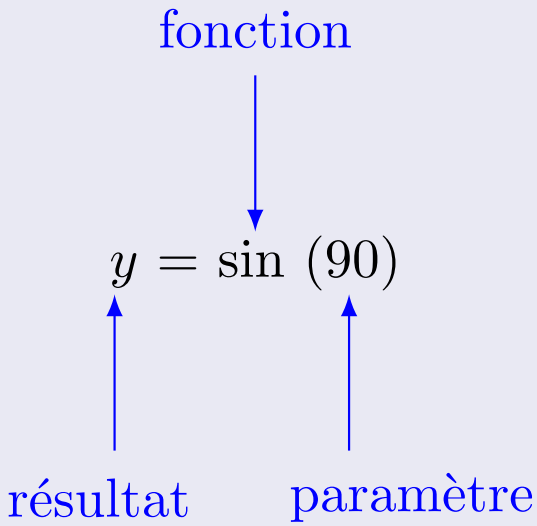


Remarque

En algorithmique aussi on peut utiliser et écrire des fonctions.

Algorithmique et programmation structurée

En mathématiques



Remarque

En algorithmique aussi on peut utiliser et écrire des fonctions.

Algorithmique et programmation structurée

Les fonctions en algorithmique, on en a déjà utilisées pas mal

- `longueur()`
- `racine_carre()`
- ...

Algorithmique et programmation structurée

Les fonctions en algorithmique, on en a déjà utilisées pas mal

- `longueur()`
- `racine_carre()`
- ...

Remarque

Il est possible de définir de nouvelles fonctions.

Algorithmique et programmation structurée

Fonction

- Regroupement d'instructions réalisant une tâche
- Pouvant accepter un ou plusieurs paramètres et retournant une unique valeur
- Pouvant avoir ses variables locales
- Pouvant appeler une ou plusieurs autres fonctions (ou elle-même $\Rightarrow$ récursivité)
- Aidant à décomposer un problème difficile en un ensemble de fonctions (sous algorithme) de complexité inférieure
- Permettant une structuration et une meilleure lisibilité
- Facilitant la maintenance du code

Algorithmique et programmation structurée

Déclaration d'une fonction : syntaxe

```
Fonction nom_fonction (paramètres et leurs types) :
    type_valeur_retour
  ...
FinFonction
```

Algorithmique et programmation structurée

Déclaration d'une fonction : syntaxe

```
Fonction nom_fonction (paramètres et leurs types) :
    type_valeur_retour
  ...
FinFonction
```

Déclaration d'une fonction : exemple

```
Fonction puissance (x : nombre, n: nombre) : nombre
Variables resultat : nombre
  resultat <- 1
  Pour i allant de 1 à n
    resultat <- resultat * x
  FinPour
  retourne resultat
FinFonction
```

Algorithmique et programmation structurée

Explication

- `resultat` est une variable locale inaccessible à l'extérieure de la fonction.
- le mot-clé `retourne` permet de quitter la fonction. Toute instruction située après ne sera pas exécutée.

Algorithmique et programmation structurée

Appel de la fonction `puissance` dans l'algorithme principal

```
Algorithme exemple_tableau
Variables a, b, res : nombre
Début
   a <- 2
   b <- 3
   res <- puissance (a, b)
   écrire ("a ^ b = ", res)
Fin
```

Algorithmique et programmation structurée

Explication

- Les paramètres placés dans la déclaration d'une fonction sont appelés paramètres formels.
- Les paramètres placés dans l'appel d'une fonction sont appelés paramètres effectifs.
- Les paramètres formels et les paramètres effectifs peuvent avoir des noms différents mais doivent avoir des types identiques.
- Par défaut, une copie des paramètres effectifs sera placée dans l'espace mémoire réservé aux paramètres formels à chaque appel de la fonction.
- La variable retournée par la fonction et la variable recevant le résultat de la fonction peuvent avoir des noms différents mais doivent avoir des types identiques.

Algorithmique et programmation structurée

Exercice

- Écrire une fonction `calcul_max2` qui prend deux paramètres de type nombre et retourne le plus grand.
- Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres et appelle la fonction `calcul_max2` pour récupérer le plus grand et l'afficher.
- Écrire une fonction `calcul_max3` qui prend trois paramètres de type nombre et retourne le plus grand.
- Modifier l'algorithme précédent pour permettre à l'utilisateur de saisir trois nombres et utiliser `calcul_max3` pour récupérer et afficher le plus grand.

Algorithmique et programmation structurée

Exercice

Écrire une fonction qui retourne la factorielle d'un nombre reçu en paramètre.

- $0! = 1$
- $1! = 1$
- $n! = n * (n-1)!$

Algorithmique et programmation structurée

Exercice

Écrire une fonction qui retourne `vrai` si le paramètre reçu est un nombre premier (divisible seulement par 1 et lui-même), `faux` sinon.

- 5 est un nombre premier parce qu'il est seulement divisible par 1 et 5.
- 6 n'est pas premier parce qu'il est divisible par 1, 2, 3 et 6.

Algorithmique et programmation structurée

Procédure

- Fonction ne retournant pas de valeur
- Comme `écrire` et `lire`

Algorithmique et programmation structurée

Déclaration d'une procédure : syntaxe

```
Procédure nom_procédure (paramètres et leurs types)
   ...
FinProcédure
```

Algorithmique et programmation structurée

Déclaration d'une procédure : syntaxe

```
Procédure nom_procédure (paramètres et leurs types)
  ...
FinProcédure
```

Déclaration d'une procédure : exemple

```
Procédure afficher_tableau (tab : tableau de nombre)
  Variables i : nombre
  Pour i allant de 0 à longueur (tab) - 1
    écrire ("tab[", i, "] = ", tab[i])
  FinPour
FinProcédure
```

Algorithmique et programmation structurée

Remarques

- Pas de mot-clé `retourne` dans la procédure.
- Pas besoin d'une variable dans l'algorithme principal pour récupérer la valeur de retour.

Algorithmique et programmation structurée

Appel de la procédure `afficher_tableau` dans l'algorithme principal

```
Algorithme tableau_procédure
Variables t : tableau de 5 nombre
          i : nombre
Début
  Pour i allant de 0 à longueur (t) - 1
    écrire ("saisir une ", i, "ème valeur")
    lire (t[i])
  FinPour
  afficher_tableau(t)
Fin
```

Algorithmique et programmation structurée

Exercice

- Écrire une procédure qui prend en paramètre deux nombres et qui permute leurs valeurs.
- Dans l'algorithme principal, demander à l'utilisateur de saisir deux nombres, ensuite appeler la procédure pour permuter les deux valeurs et enfin afficher les.

Algorithmique et programmation structurée

Exercice

- Écrire une procédure qui prend en paramètre deux nombres et qui permute leurs valeurs.
- Dans l'algorithme principal, demander à l'utilisateur de saisir deux nombres, ensuite appeler la procédure pour permuter les deux valeurs et enfin afficher les.

Remarques

Faites une exécution à la main et vérifier que l'algorithme n'affiche pas le résultat attendu.

Algorithmique et programmation structurée

Deux modes de transmission

- **Transmission par valeur** (par défaut) : les valeurs des paramètres effectifs sont recopiées dans les paramètres formels correspondants au moment de l'appel de la procédure. Les paramètres effectifs ne subissent aucune modification.
- **Transmission par adresse** (référence) : les adresses des paramètres effectifs sont transmises à la procédure appelée. Les paramètres effectifs subissent les modifications lors de l'exécution de la procédure.

Algorithmique et programmation structurée

Exemple de paramètres passées par référence

```
Procédure permuter (a :  nombre par adresse, b :
   nombre par adresse)
Variables tmp : nombre
  tmp <- a
  a <- b
  b <- tmp
FinProcédure
```

Algorithmique et programmation structurée

Considérons l'algorithme suivant

```
Algorithme tableau_procédure
Variables t1, t2 : tableau de 5 nombre
Début
  remplir_tableau(t1)
  remplir_tableau(t2)
  Si (sont_identiques(t1, t2) = vrai) alors
    écrire ("les deux tableaux sont identiques")
  Sinon
    écrire ("les deux tableaux sont différents")
  FinSi
Fin
```

Algorithmique et programmation structurée

Exercice

- Écrire la procédure `remplir_tableau` qui permet de remplir un tableau passé en paramètre par des valeurs saisies par l'utilisateur.
- Écrire la fonction `sont_identiques` qui retourne `vrai` si les deux tableaux passés en paramètre contiennent les mêmes valeurs dans l'ordre, `faux` sinon.

Algorithmique et programmation structurée

Récursivité

- Une fonction est dite **récursive** si elle s'appelle elle-même.
- À chaque appel, une nouvelle instance indépendante sera créée avec ses propres paramètres.

Algorithmique et programmation structurée

Exemple de récursivité avec la fonction `factorielle`

```
Fonction factorielle (n : nombre) : nombre
  Si (n = 0) alors
    retourne 1
  Sinon
    retourne n * factorielle (n - 1)
  FinSi
FinFonction
```

Algorithmique et programmation structurée

Exercice

Écrire une fonction récursive qui calcule le nième terme de la suite de **Fibonacci** (n étant l'unique paramètre de la fonction).

- $U_0 = 0$
- $U_0 = 1$
- $U_n = U_{n-1} + U_{n-2}$

Algorithmique et programmation structurée

La complexité ?

Domaine de l'informatique étudiant la quantité de ressources (temps, mémoire) nécessaire pour un algorithme pour résoudre un problème.

- Complexité spatiale : pour quantifier l'espace mémoire utilisé.
- Complexité temporelle : pour quantifier la vitesse d'exécution.

Algorithmique et programmation structurée

La complexité ?

Domaine de l'informatique étudiant la quantité de ressources (temps, mémoire) nécessaire pour un algorithme pour résoudre un problème.

- Complexité spatiale : pour quantifier l'espace mémoire utilisé.
- Complexité temporelle : pour quantifier la vitesse d'exécution.

La complexité : pourquoi ?

- Classifier les problèmes selon la difficulté.
- Comparer les algorithmes.

Algorithmique et programmation structurée

Question

Comment mesurer la complexité (temporelle) ?

Algorithmique et programmation structurée

Question

Comment mesurer la complexité (temporelle) ?

Considérons l'exemple suivant

```
x <- 3 + 2
```

Algorithmique et programmation structurée

Question

Comment mesurer la complexité (temporelle) ?

Considérons l'exemple suivant

```
x <- 3 + 2
```


Complexité de l'instruction précédente

1 affectation + 1 addition = 2 opérations = 2 unités de temps

Algorithmique et programmation structurée

Considérons l'exemple suivant

```
Pour i allant de 1 à n
  x <- n + 2
FinPour
n <- 0
```

Algorithmique et programmation structurée

Considérons l'exemple suivant

```
Pour i allant de 1 à n
  x <- n + 2
FinPour
n <- 0
```


Complexité

(1 affectation + 1 addition) * n + 1 affectation = $2n$ + 1 unités

Algorithmique et programmation structurée

Considérons l'exemple suivant

```
Pour i allant de 1 à n
  x <- n + 2
FinPour
n <- 0
```

Complexité

(1 affectation + 1 addition) * n + 1 affectation = $2n$ + 1 unités

Ordre de complexité

$2n + 1 = O(n)$

Algorithmique et programmation structurée

Considérons l'exemple suivant

```
Pour i allant de 1 à n
  Si (x > 10) alors
    Pour j allant de 1 à n
      x <- n + 2
    FinPour
  FinSi
FinPour
```

Algorithmique et programmation structurée

Considérons l'exemple suivant

```
Pour i allant de 1 à n
  Si (x > 10) alors
    Pour j allant de 1 à n
      x <- n + 2
    FinPour
  FinSi
FinPour
```

Deux formes de complexité

- complexité dans le meilleur des cas :
  1 comparaison * n = n unités = O(n)
- complexité dans le pire des cas :
  (1 affectation + 1 addition + 1 comparaison) * n * n = $3n^2$ unités = O(n^2)

Algorithmique et programmation structurée

Déterminer la complexité de l'algorithme suivant

```
Si (x > 10) alors
  Pour i allant de 1 à n
    Pour j allant de 1 à n
      Pour k allant de 1 à n
        x <- n + 2
      FinPour
    FinPour
  FinPour
Sinon
  Pour i allant de 1 à n
    x <- i + 2
  FinPour
FinSi
```

Algorithmique et programmation structurée

Classes de complexité

- O(n) : linéaire
- O($log(n)$) : logarithmique
- O(n^2) : quadratique
- O(n^3) : cubique
- O($n!$) : factorielle
- O(2^n) : exponentielle

Algorithmique et programmation structurée

Le tri ?

Ordonner les éléments du tableau dans l'ordre croissant ou décroissant.

Algorithmique et programmation structurée

Le tri ?

Ordonner les éléments du tableau dans l'ordre croissant ou décroissant.

Plusieurs algorithmes proposés pour le tri d'un tableau

- Tri par sélection
- Tri à bulles
- Tri par insertion
- Tri rapide
- ...

Algorithmique et programmation structurée

Tri par sélection

- A l'itération `i`, on cherche le plus petit élément à partir de la position `i` et jusqu'à la fin du tableau.
- Le plus petit élément sera permuté avec l'élément ayant la position `i`.
- Complexité dans le meilleur des cas = O(n^2) : n étant la taille du tableau.
- Complexité dans le pire des cas = O(n^2) : n étant la taille du tableau.

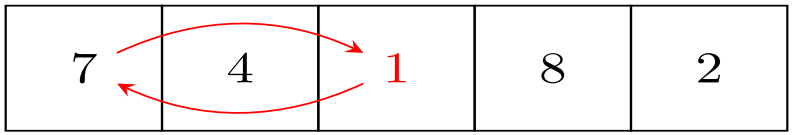
Algorithmique et programmation structurée

7	4	1	8	2

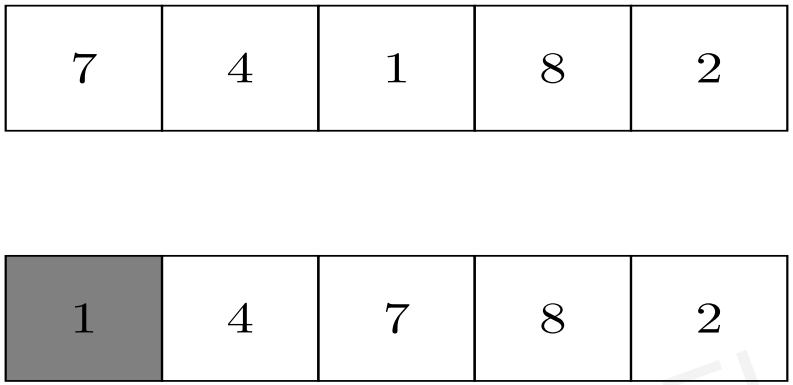
Algorithmique et programmation structurée

7	4	1	8	2

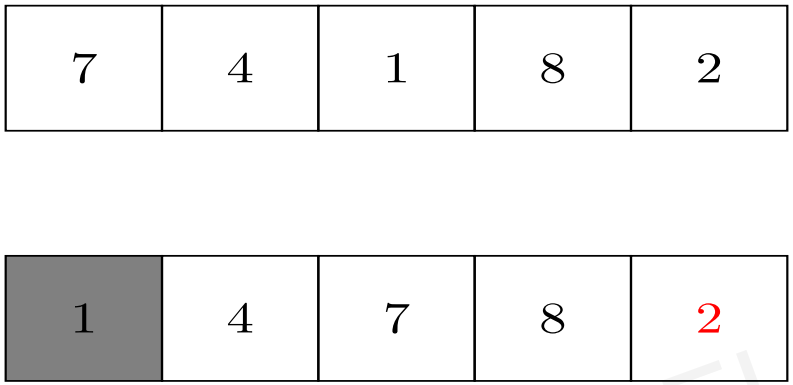
Algorithmique et programmation structurée



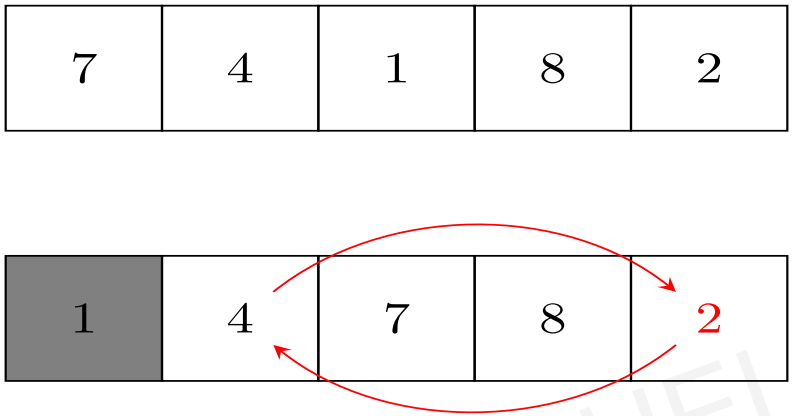
Algorithmique et programmation structurée



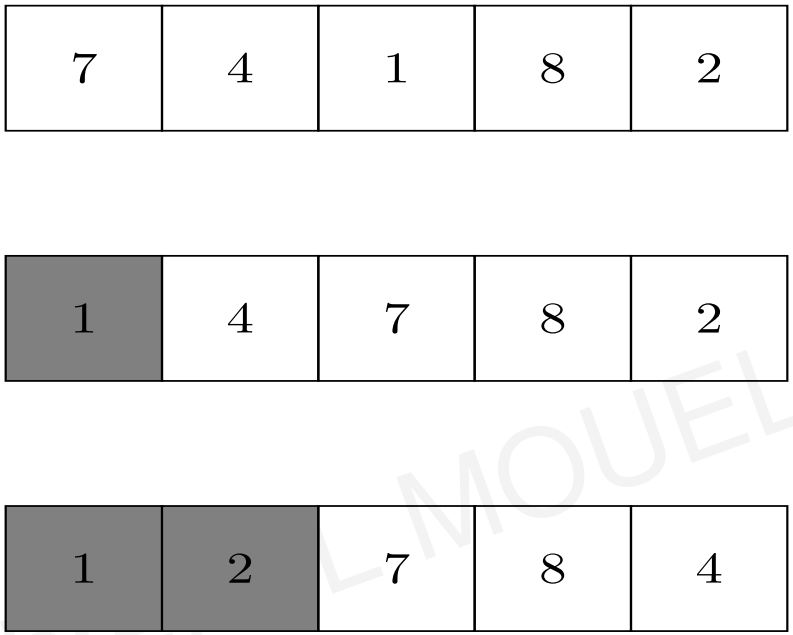
Algorithmique et programmation structurée



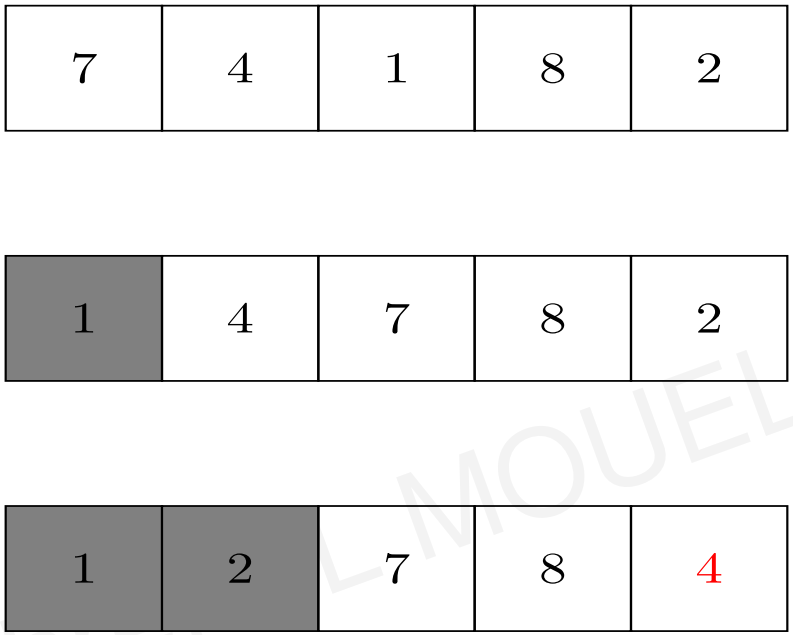
Algorithmique et programmation structurée



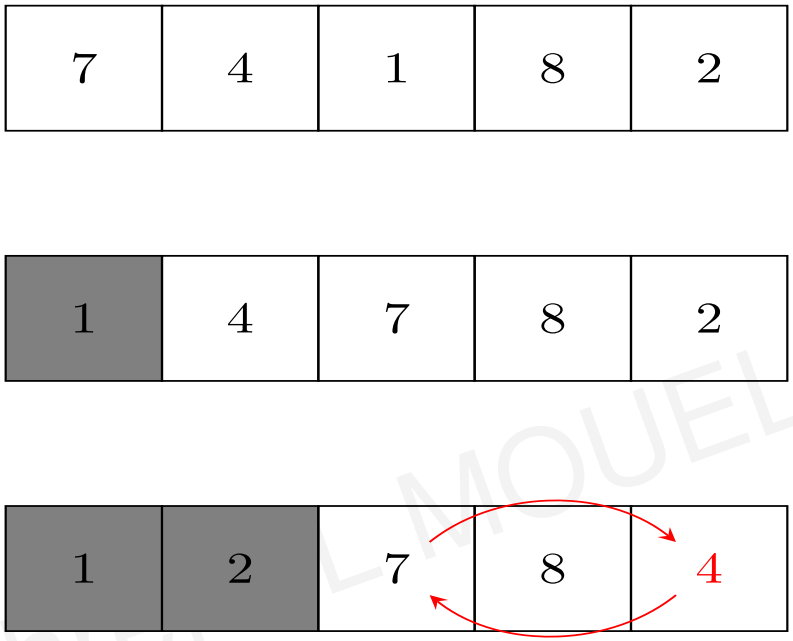
Algorithmique et programmation structurée



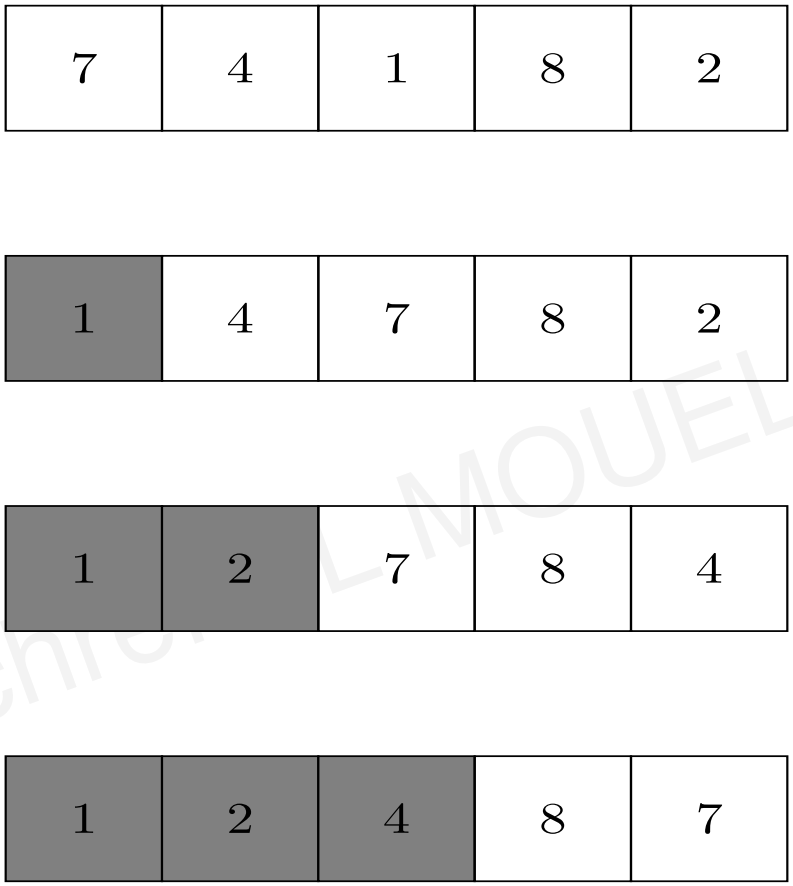
Algorithmique et programmation structurée



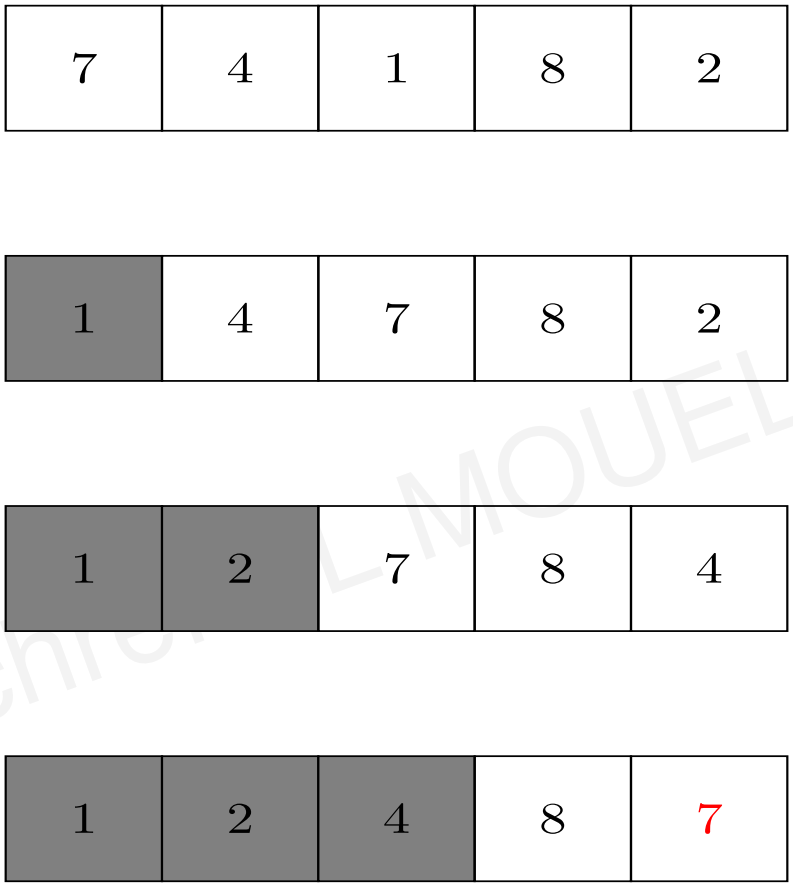
Algorithmique et programmation structurée



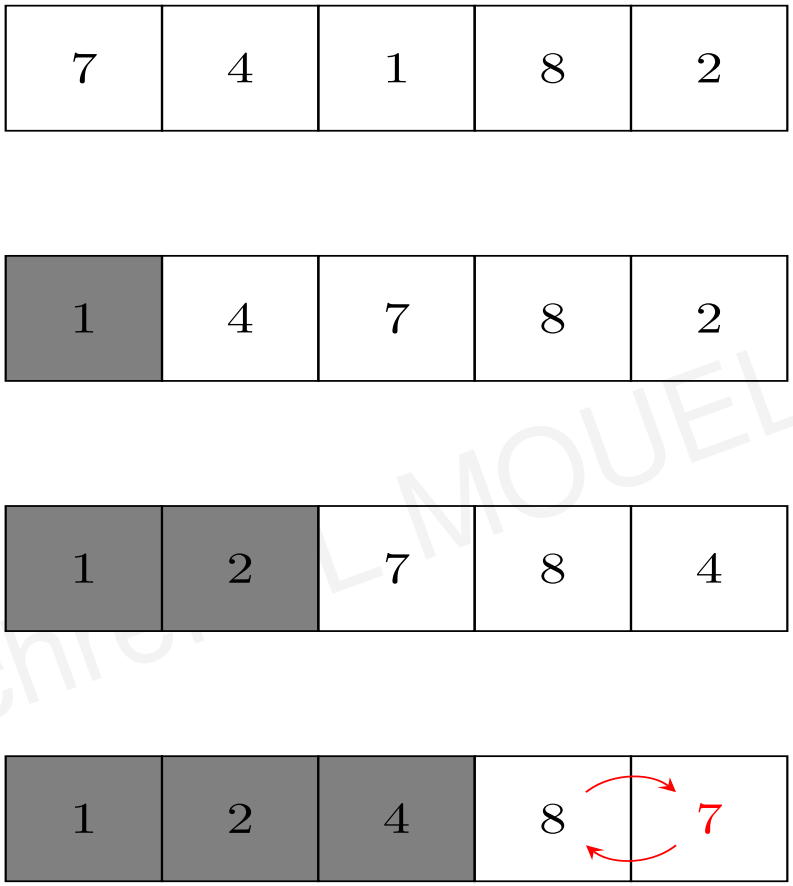
Algorithmique et programmation structurée



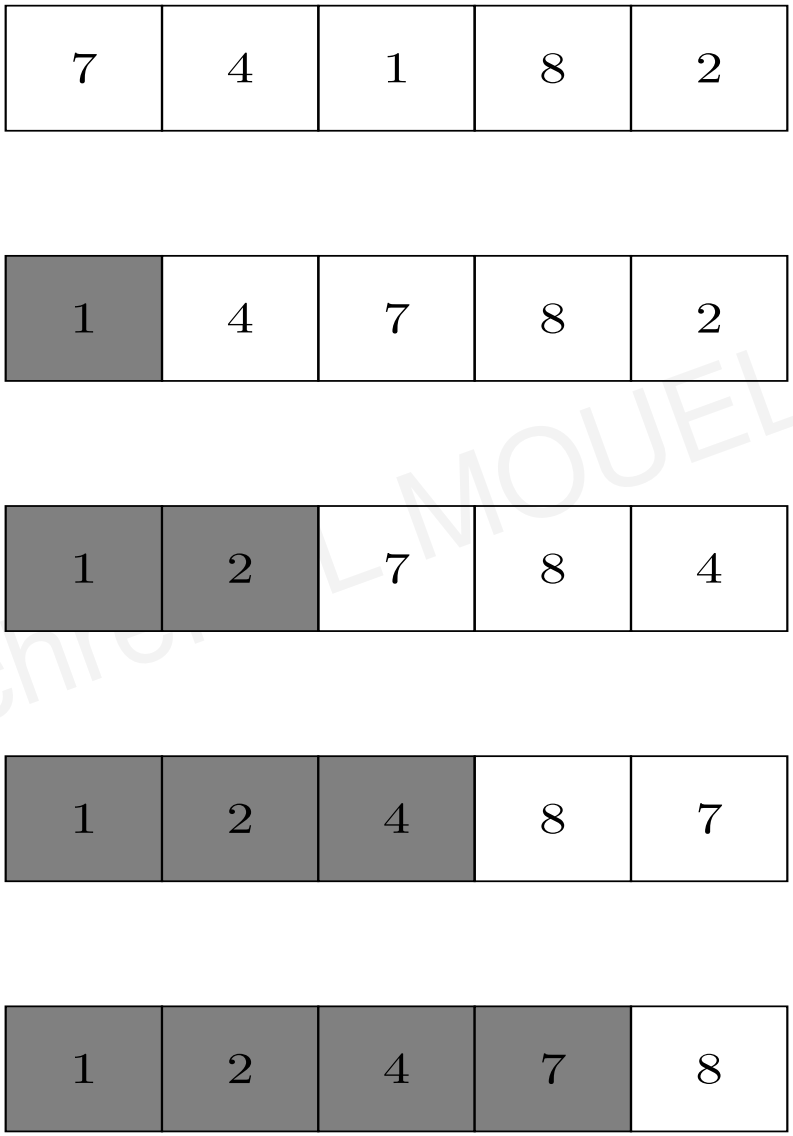
Algorithmique et programmation structurée



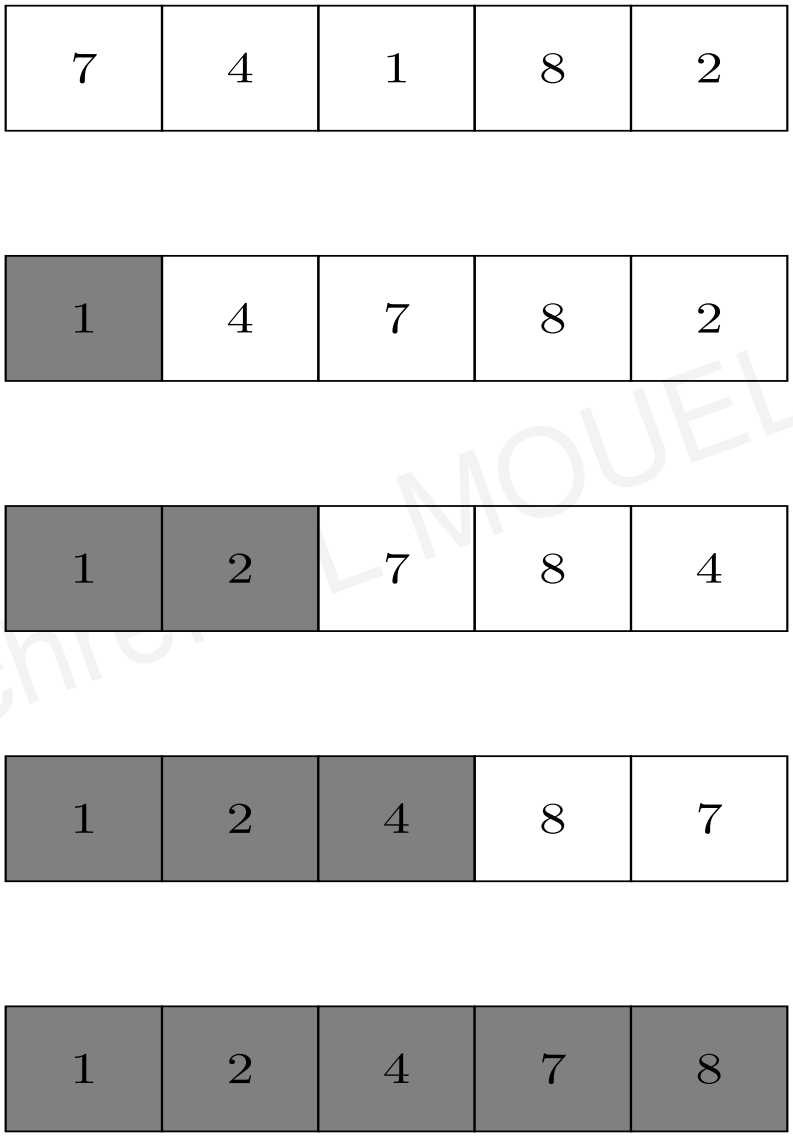
Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée

Exercice

Écrire une fonction permettant de trier un tableau par sélection.

Algorithmique et programmation structurée

Tri par sélection

```
Pour i allant de 0 à n - 2
  indice_ppe <-  i
  Pour j allant de i + 1 à n - 1
    Si (t[j] < t[indice_ppe]) alors
      indice_ppe <- j
    FinSi
  FinPour
  temp <- t[indice_ppe]
  t[indice_ppe] <- t[i]
  t[i] <- temp
FinPour
```

Algorithmique et programmation structurée

Tri à bulles

- Pour la première itération, on parcourt le tableau et on effectue une comparaison des éléments consécutifs et on permute si les éléments comparés ne sont pas dans l'ordre.
- À la fin de la première itération, le plus grand élément se trouve à la dernière position (`n-1`).
- À partir de l'itération suivante (`i`), on commence du début du tableau et on s'arrête à la position `n - i` et on refait les mêmes comparaisons et permutations.
- Complexité dans le meilleur des cas = O(n) : n étant la taille du tableau.
- Complexité dans le pire des cas = O(n^2) : n étant la taille du tableau.

Algorithmique et programmation structurée

7	4	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

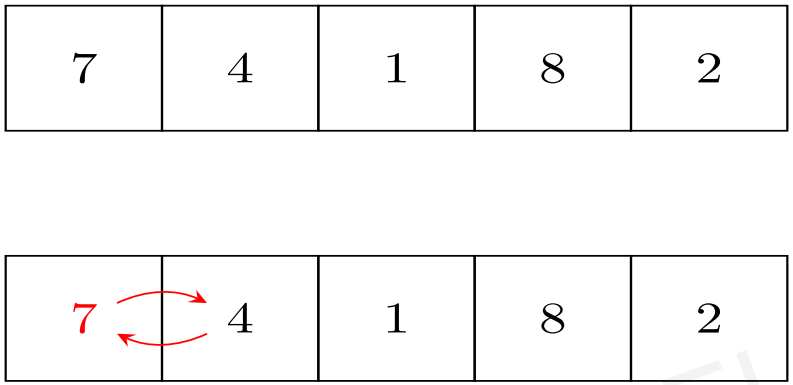
7	4	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

7	4	1	8	2

Algorithmique et programmation structurée



Algorithmique et programmation structurée

7	4	1	8	2

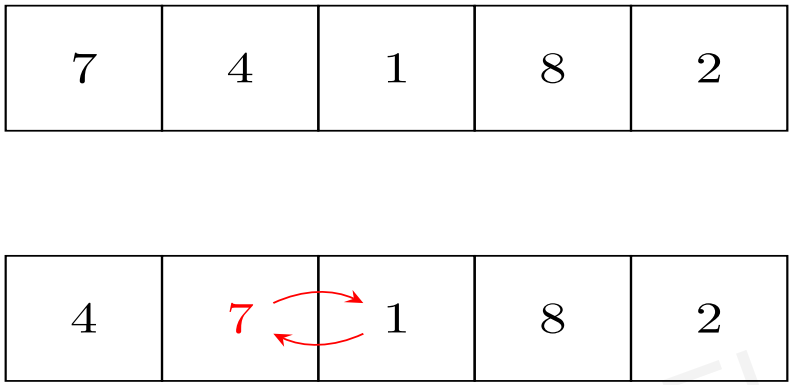
4	7	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

Algorithmique et programmation structurée



Algorithmique et programmation structurée

7	4	1	8	2

4	1	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

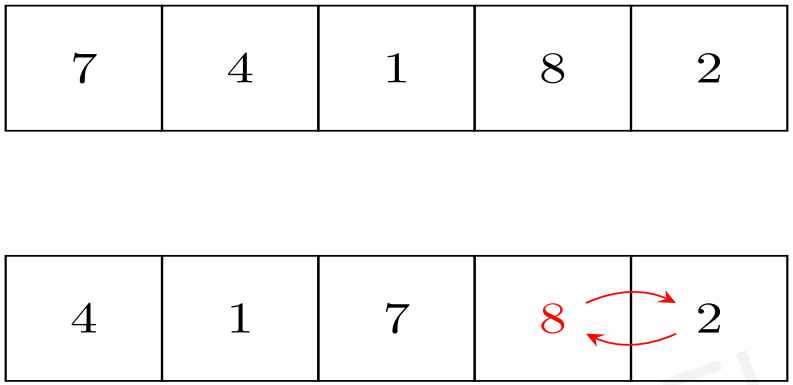
4	1	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	1	7	8	2

Algorithmique et programmation structurée

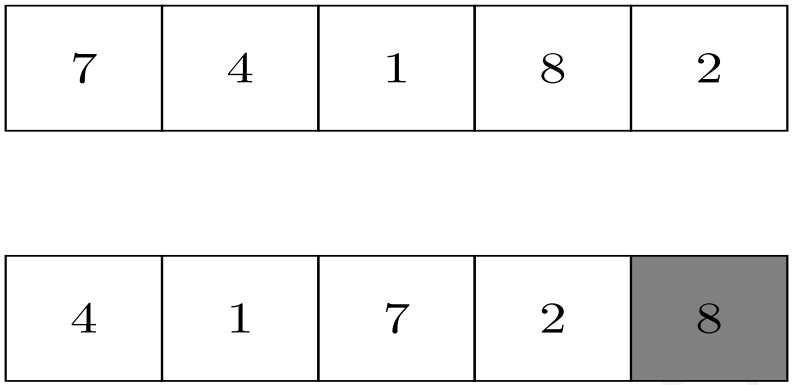


Algorithmique et programmation structurée

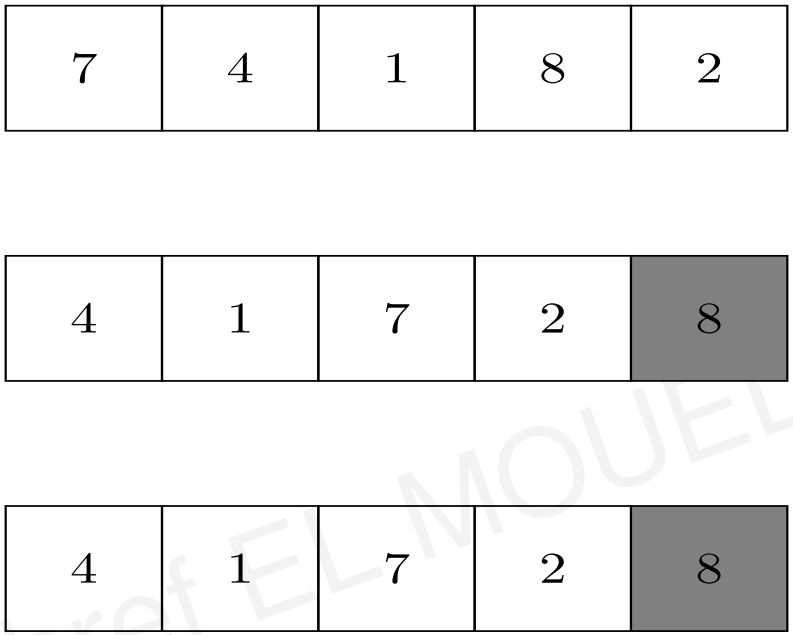
7	4	1	8	2

4	1	7	2	8

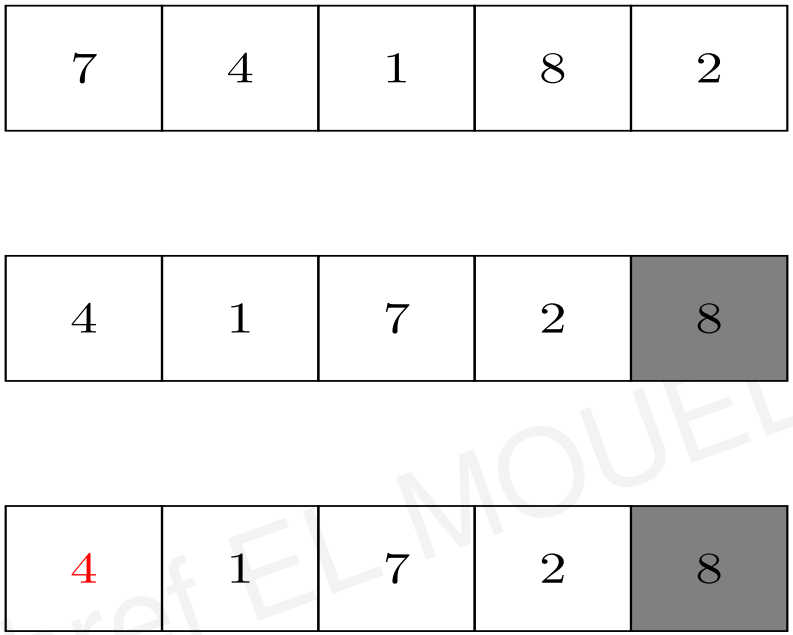
Algorithmique et programmation structurée



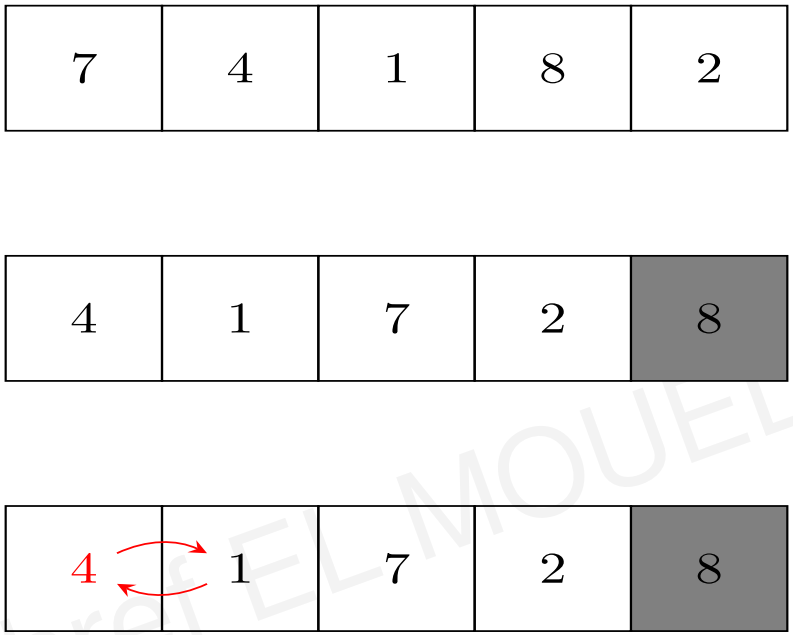
Algorithmique et programmation structurée



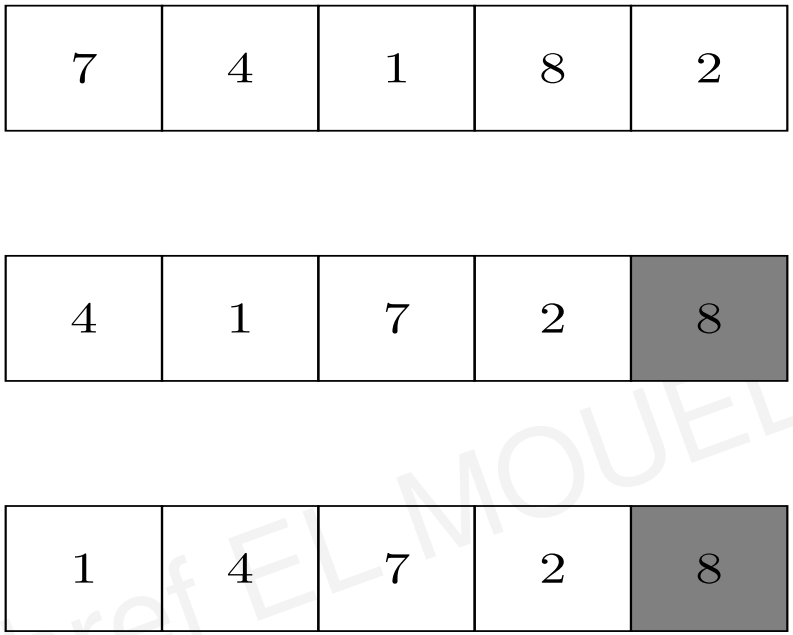
Algorithmique et programmation structurée



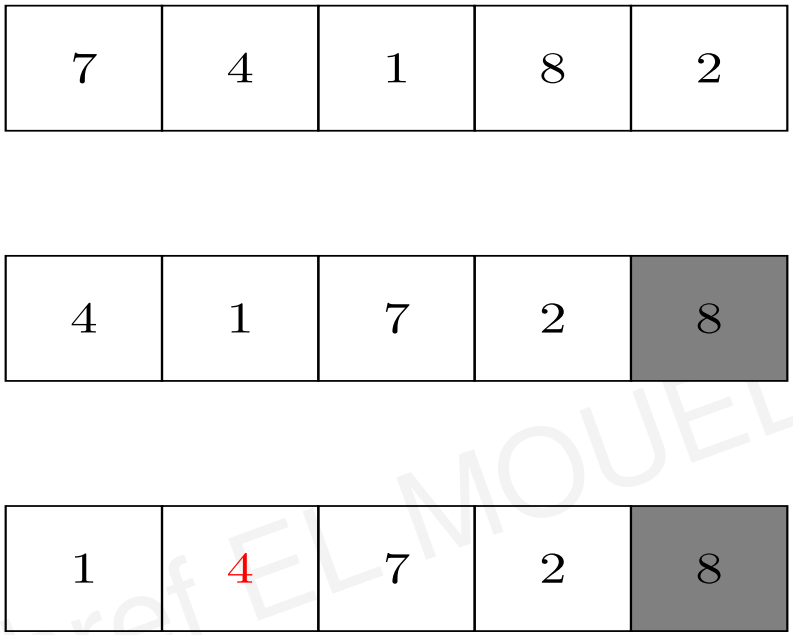
Algorithmique et programmation structurée



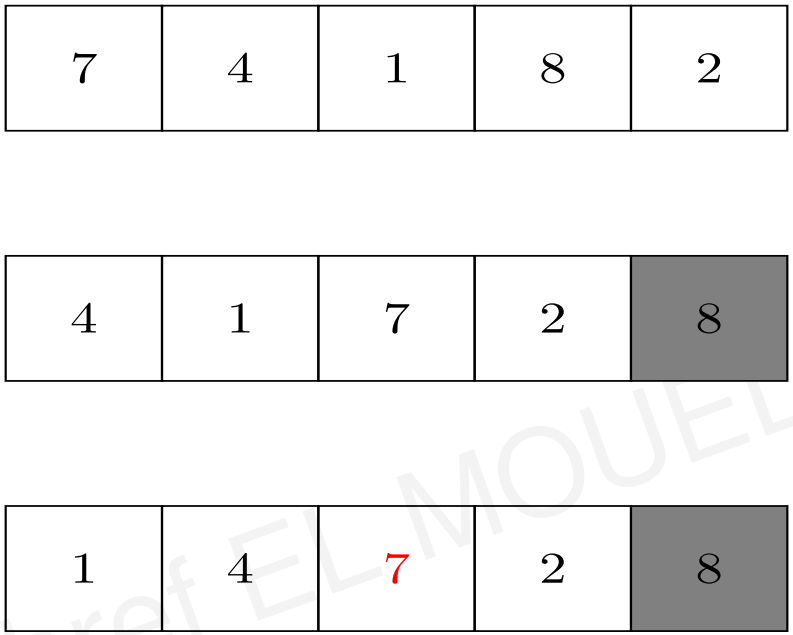
Algorithmique et programmation structurée



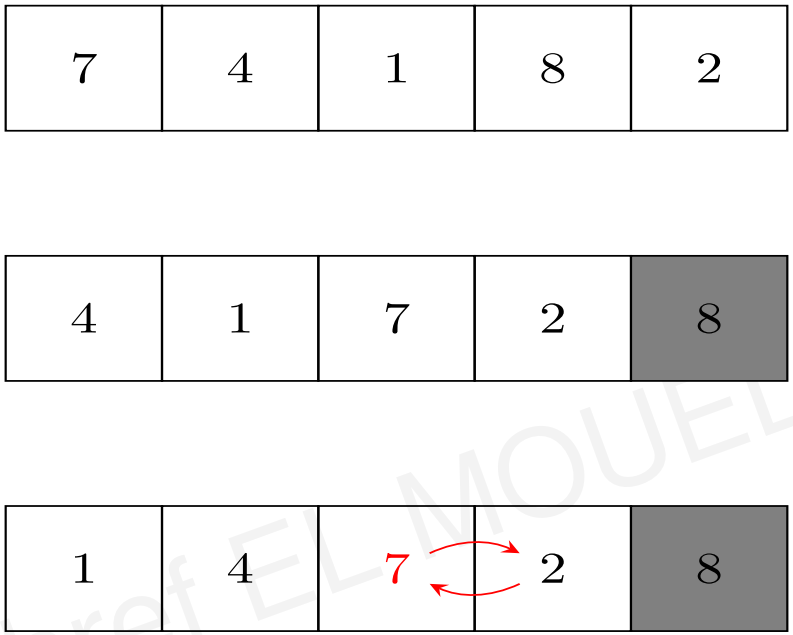
Algorithmique et programmation structurée



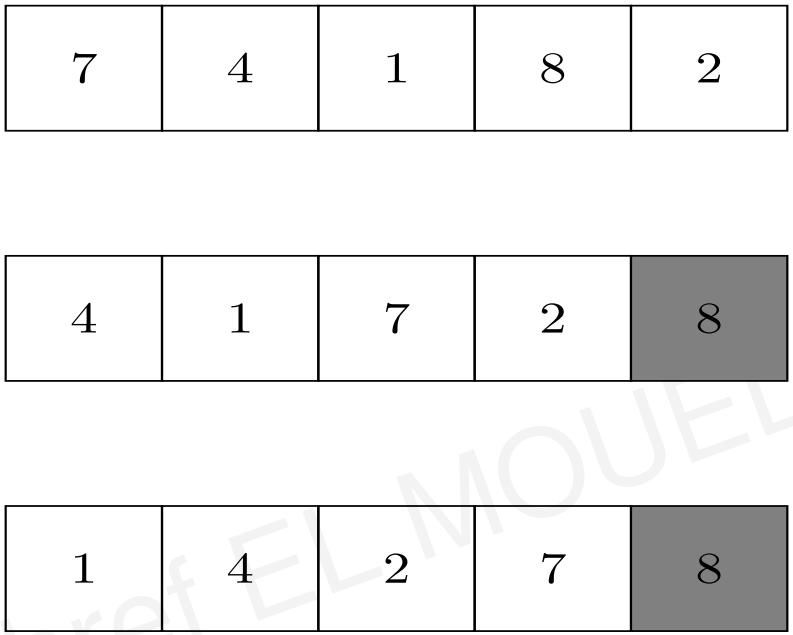
Algorithmique et programmation structurée



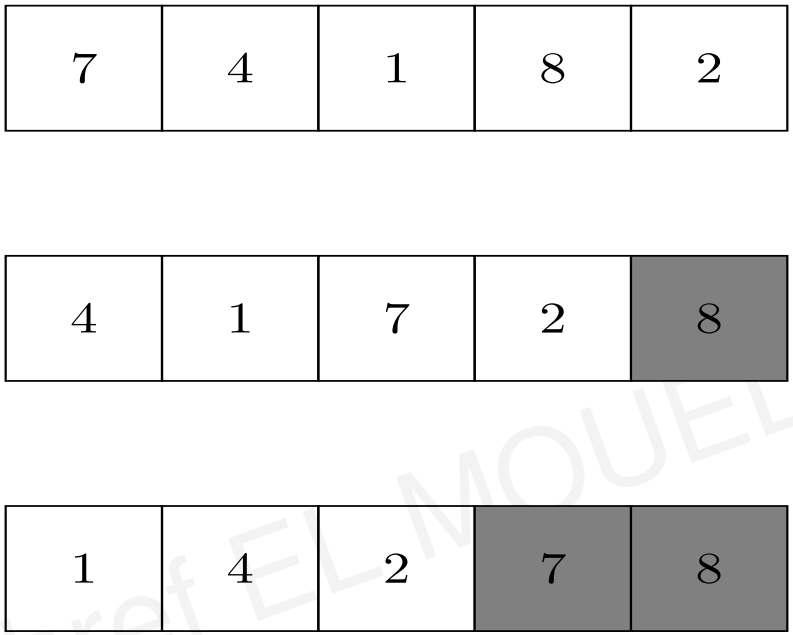
Algorithmique et programmation structurée



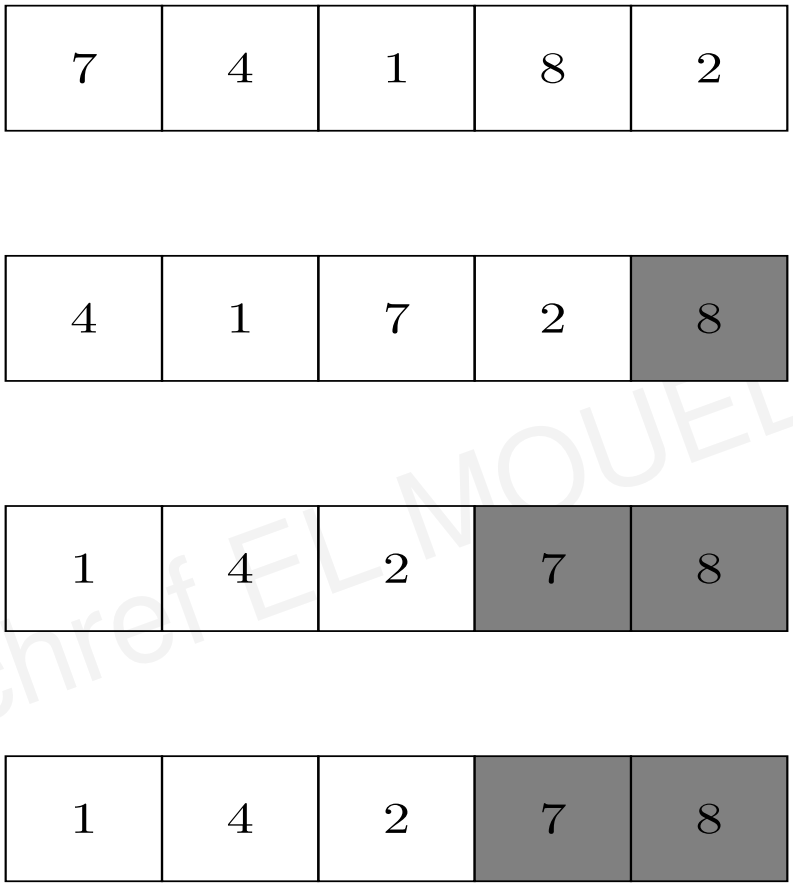
Algorithmique et programmation structurée



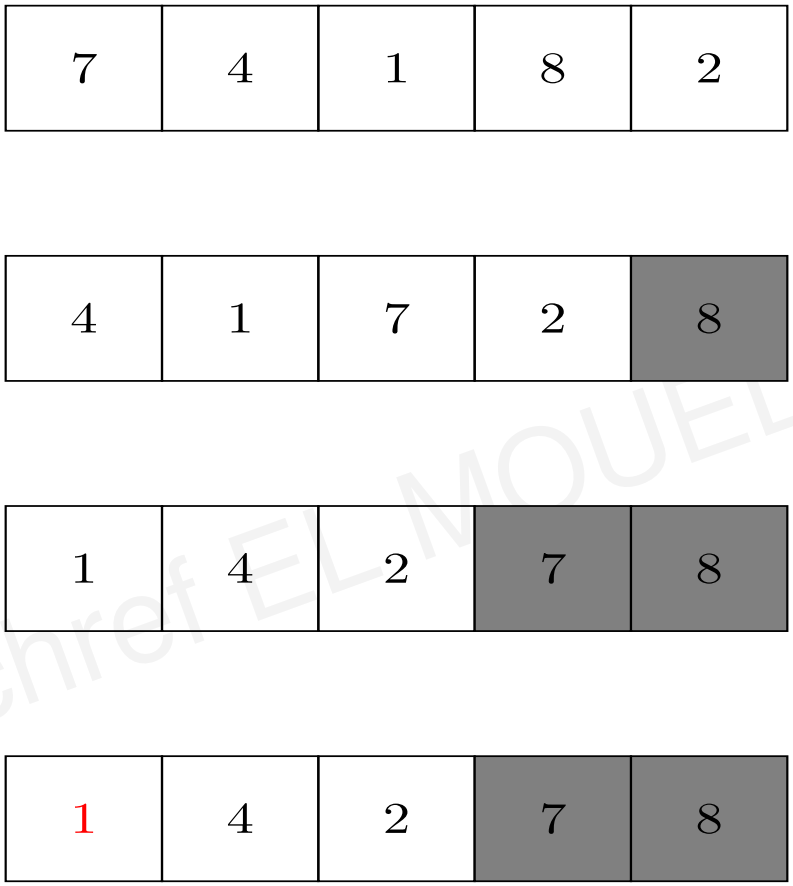
Algorithmique et programmation structurée



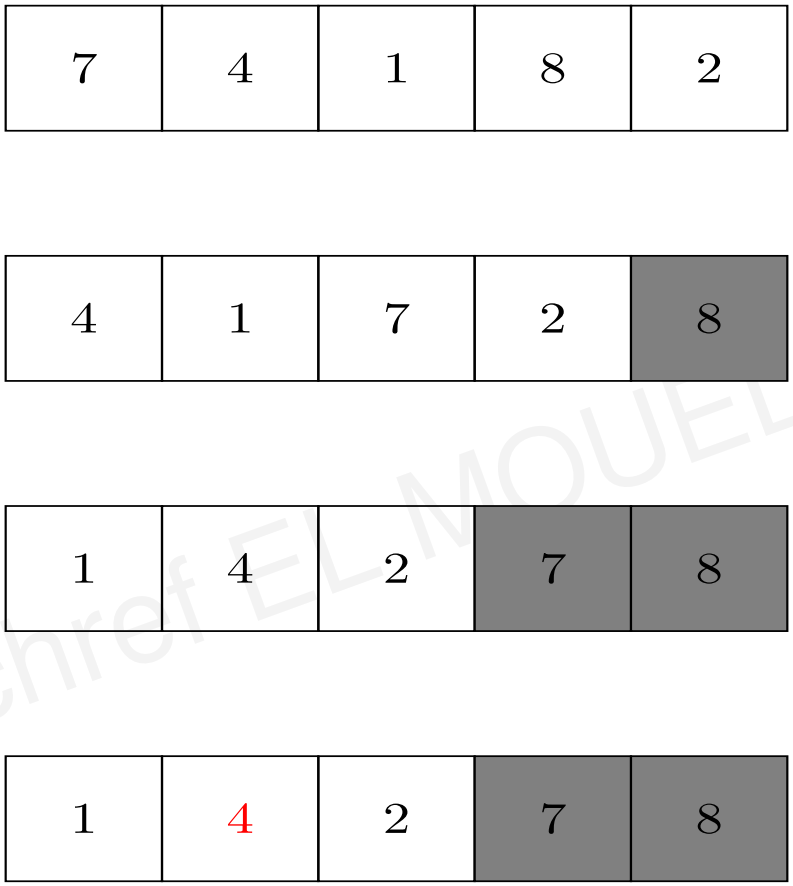
Algorithmique et programmation structurée



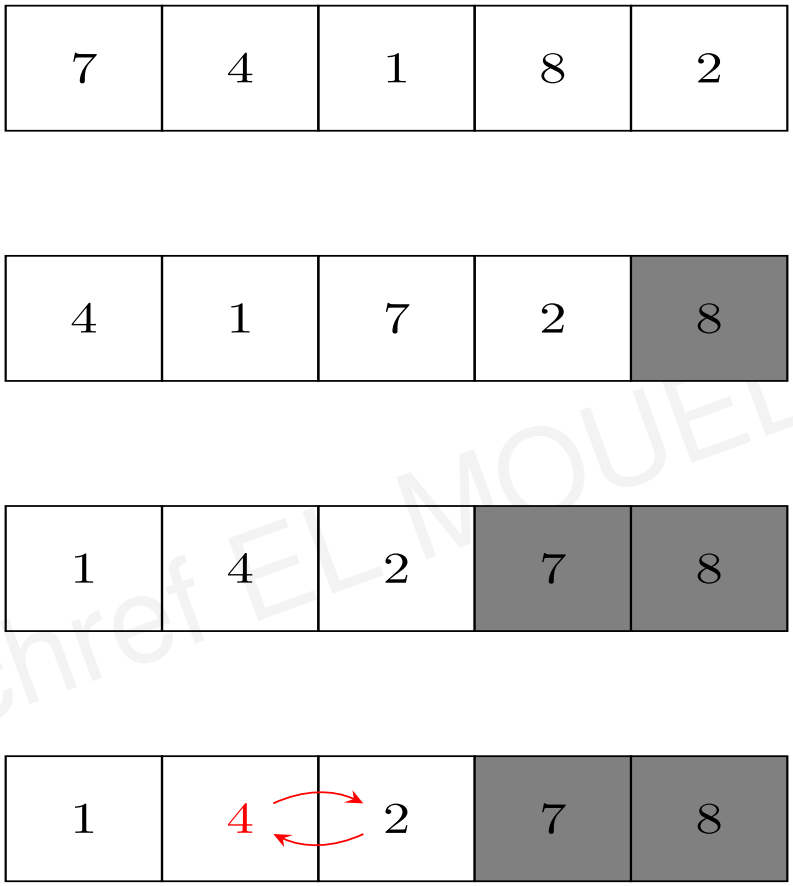
Algorithmique et programmation structurée



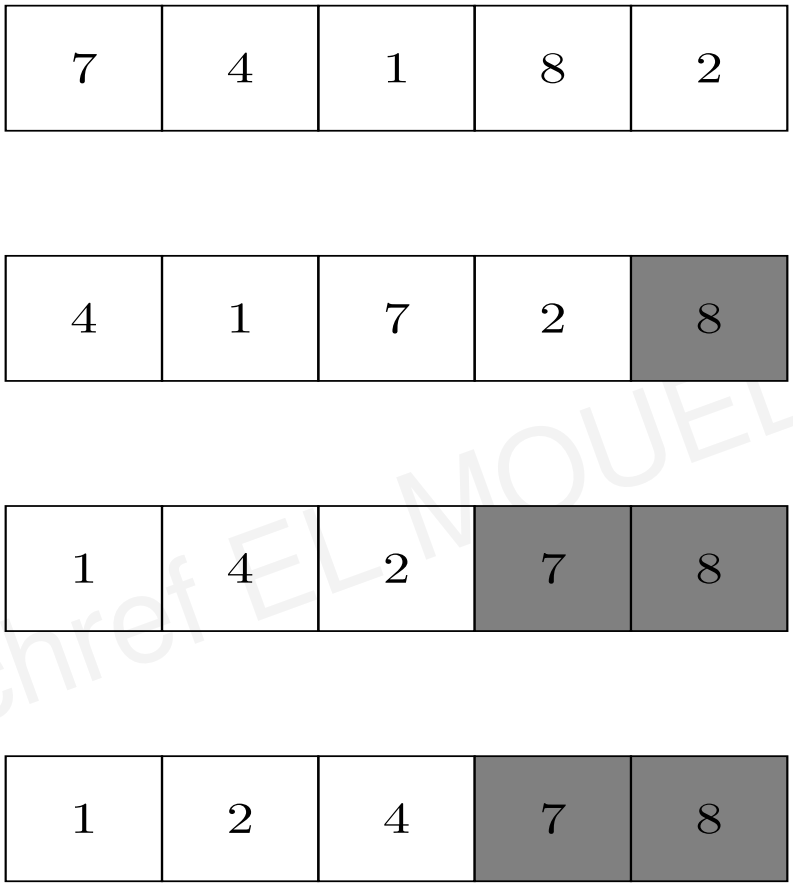
Algorithmique et programmation structurée



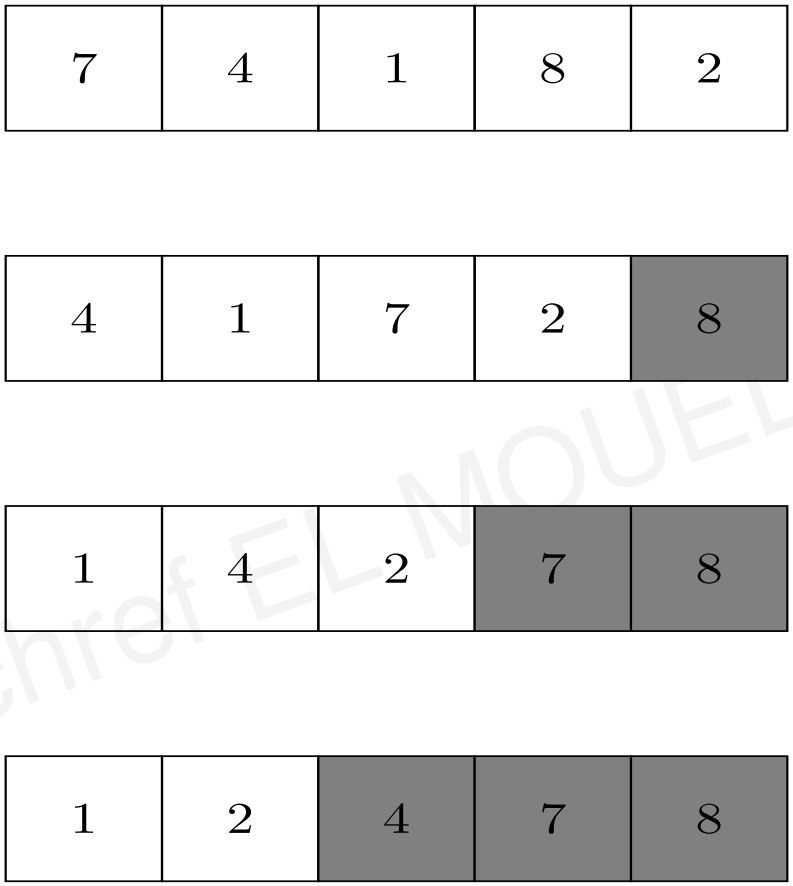
Algorithmique et programmation structurée



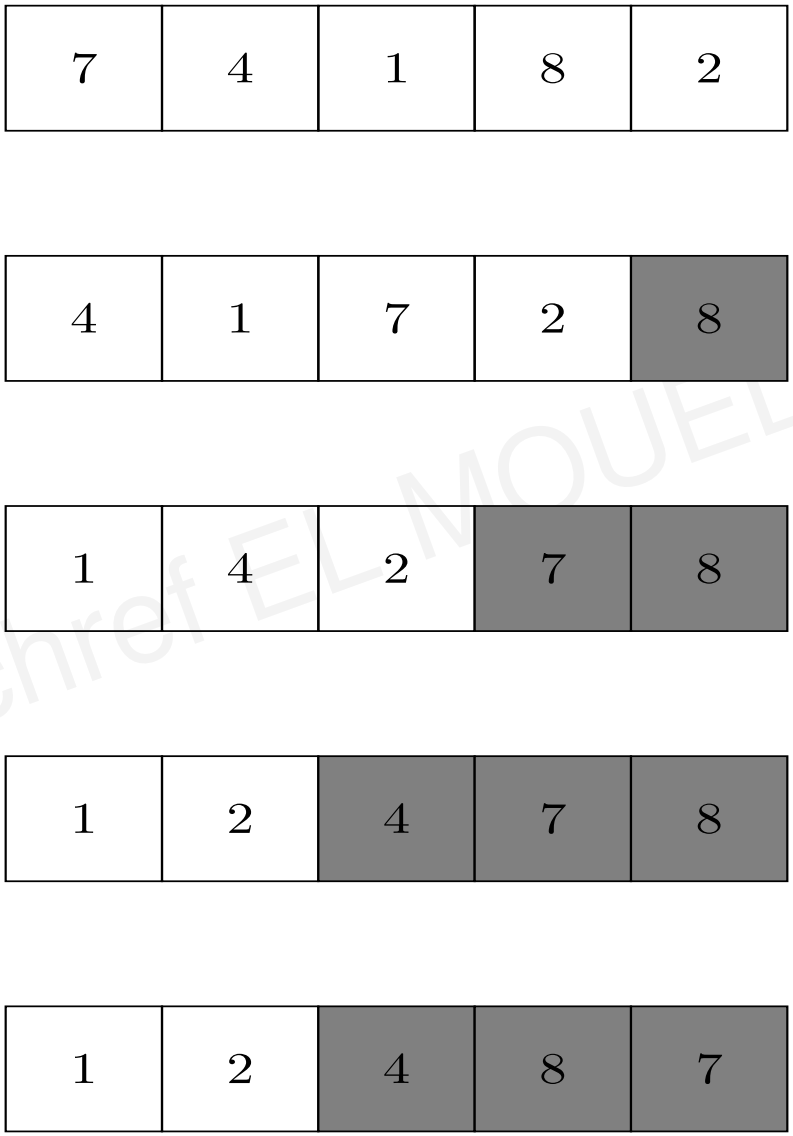
Algorithmique et programmation structurée



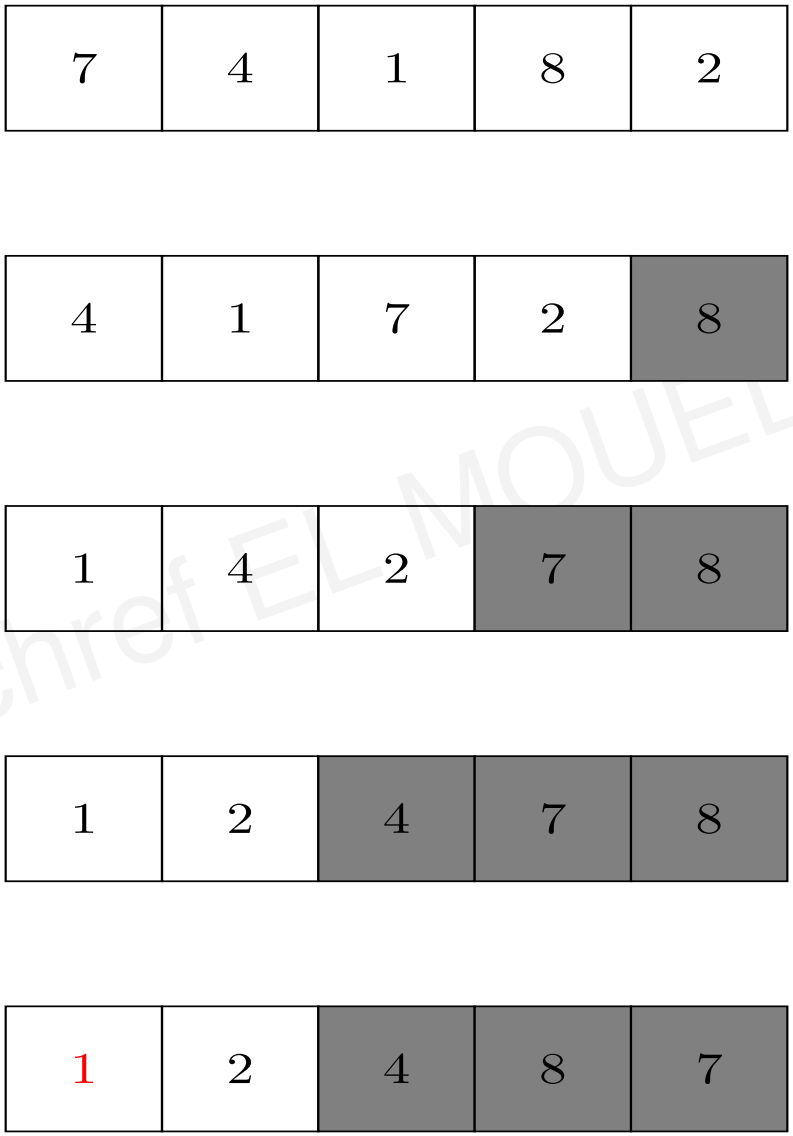
Algorithmique et programmation structurée



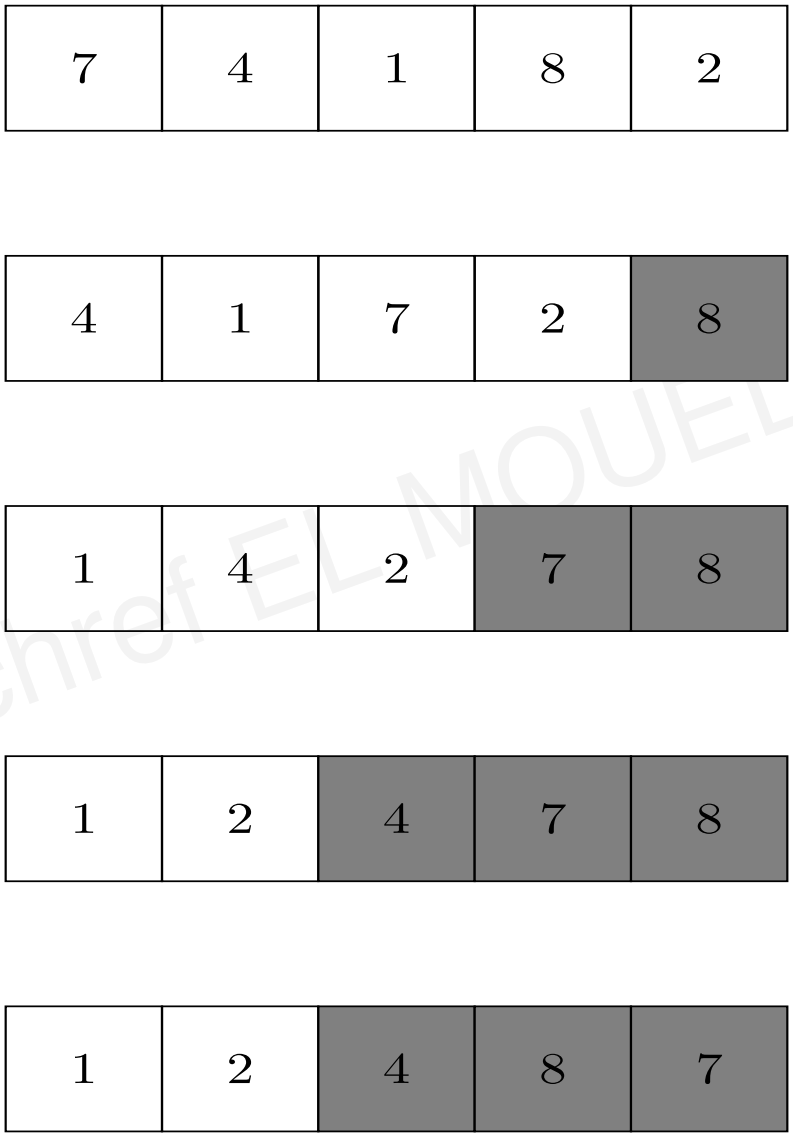
Algorithmique et programmation structurée



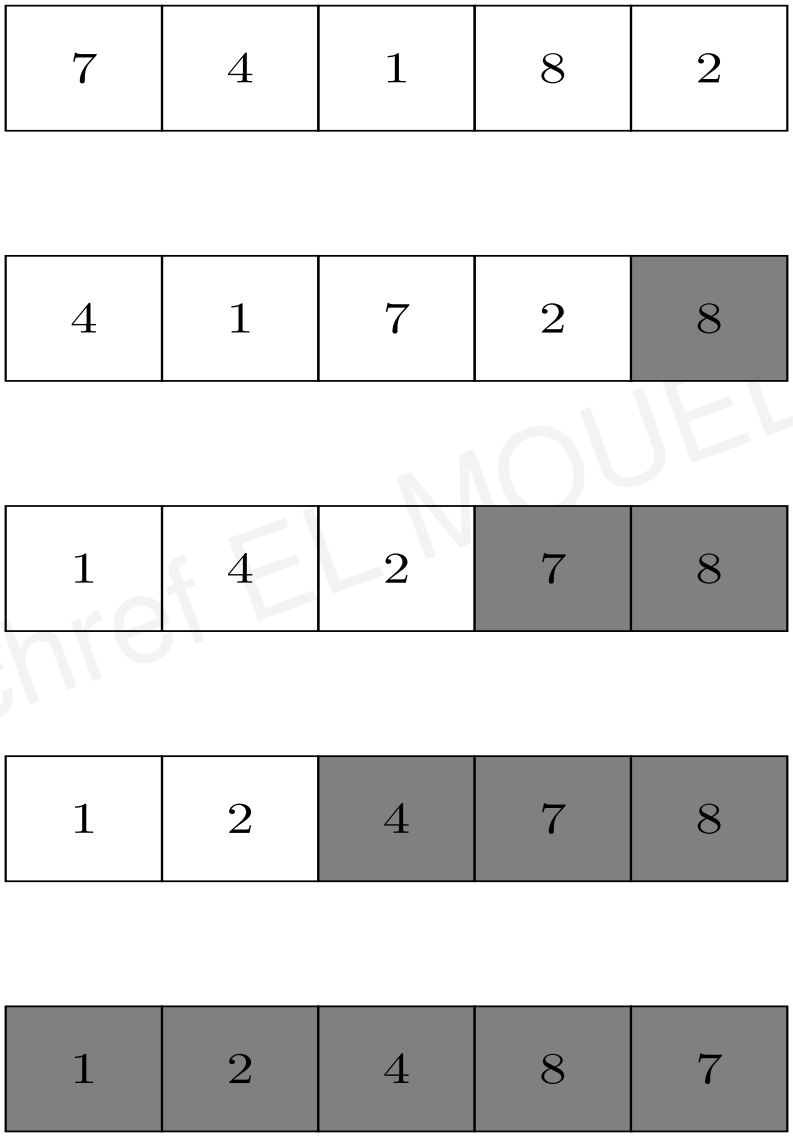
Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée

Exercice

Écrire une fonction permettant de trier un tableau en respectant l'approche tri à bulles.

Algorithmique et programmation structurée

Tri à bulles

```
Pour i allant de n - 1 à 1
  Pour j allant de 0 à i - 1
    Si (t[j + 1] < t[j]) alors
      temp <- t[j]
      t[j] <- t[j + 1]
      t[j + 1] <- temp
    FinSi
  FinPour
FinPour
```

Algorithmique et programmation structurée

Tri par insertion

- On parcourt le tableau de la position 1 jusqu'à la position `n-1`.
- À l'itération `i`, les éléments ayant une position inférieure à `i` sont déjà triés.
- À l'itération `i`, on compare l'élément d'indice `i` avec les précédents et on l'insère de sorte que le tableau reste trié en décalant à droite les éléments qui lui sont supérieurs.
- Complexité dans le meilleur des cas = O(n) : n étant la taille du tableau.
- Complexité dans le pire des cas = O(n^2) : n étant la taille du tableau.

Algorithmique et programmation structurée

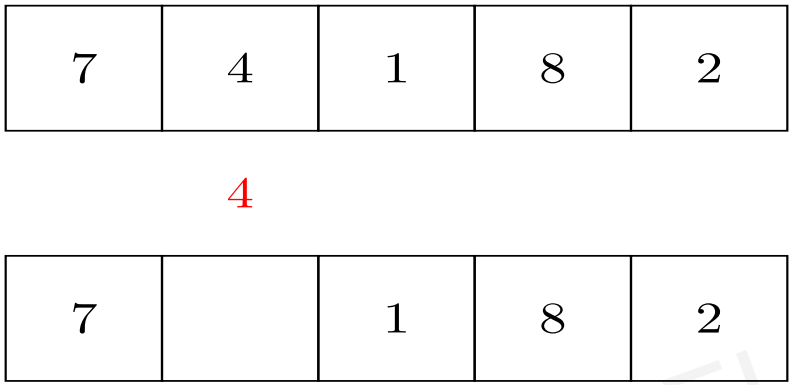
7	4	1	8	2

Algorithmique et programmation structurée

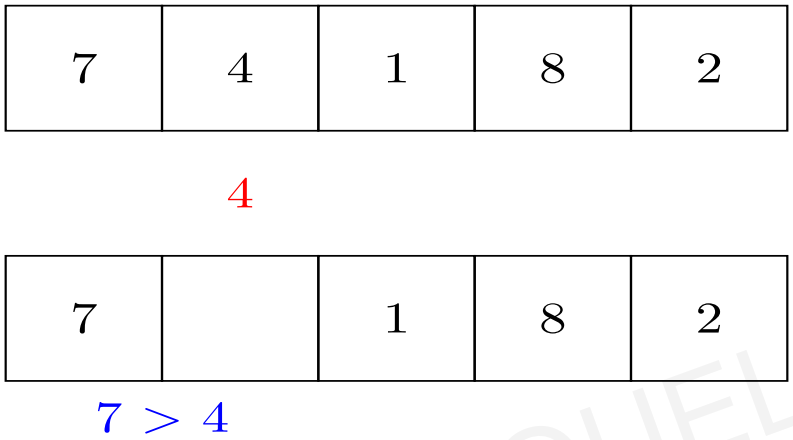
7	4	1	8	2

7	4	1	8	2

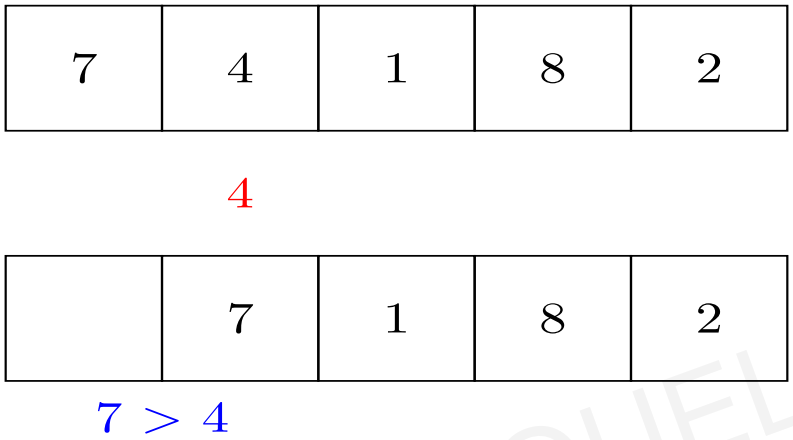
Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée



Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

4	7	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

4	7	1	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

4	7		8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

4	7		8	2

$7 > 1$

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

4		7	8	2

$7 > 1$

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

4		7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

4		7	8	2

$4 > 1$

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1

	4	7	8	2

$4 > 1$

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

8

1	4	7		2

$7 < 8$

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

1	4	7	8	2

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

2

1	4	7	8	

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

$8 > 2$ $\qquad$ 2

1	4	7	8	

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

$8 > 2$ $\qquad$ 2

1	4	7		8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

2

1	4	7		8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

$7 > 2$ $\qquad$ 2

1	4	7		8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

2

1	4		7	8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

$4 > 2$ $\qquad$ 2

1	4		7	8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

2

1		4	7	8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

$1 < 2$ $\qquad$ 2

1		4	7	8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

1	2	4	7	8

Algorithmique et programmation structurée

7	4	1	8	2

4	7	1	8	2

1	4	7	8	2

1	4	7	8	2

1	2	4	7	8

Algorithmique et programmation structurée

Exercice

Écrire une fonction permettant de trier un tableau par insertion.

Algorithmique et programmation structurée

Tri par insertion

```
Pour i allant de 1 à n - 1
  x <- t[i]
  j <- i
  TantQue (j > 0 et t[j - 1] > x)
    t[j] <- t[j - 1]
    j <- j - 1
  FinTantQue
  t[j] <- x
FinPour
```

Algorithmique et programmation structurée

Recherche dichotomique

- Méthode de recherche dans un tableau trié.
- Principe :
  - Comparer la valeur recherchée avec celle du milieu.
  - Si les valeurs sont égales, la recherche est terminée.
  - Si elle est supérieure, on refait la recherche dans la deuxième moitié du tableau.
  - Si elle est inférieure, on refait la recherche dans la première moitié du tableau.
- Complexité dans le pire des cas = O($log_2(n)$) : n étant la taille du tableau.

Algorithmique et programmation structurée

Valeur recherchée : 11,

1	4	5	7	8	9	10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11, déb = 0, fin = 11, milieu = 5

1	4	5	7	8	9	10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 0, fin = 11, milieu = 5

1	4	5	7	8	9	10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 0, fin = 11, milieu = 5

1	4	5	7	8	9	10	11	12	14	15

$9 < 11$

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = , fin = 11, milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 11, milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 11, milieu = 8

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 11, milieu = 8

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 11, milieu = 8

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

$11 < 12$

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = , milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 7, milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 7, milieu = 6

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 7, milieu = 6

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 6, fin = 7, milieu = 6

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

$10 < 11$

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = , fin = 7, milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 7, fin = 7, milieu =

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 7, fin = 7, milieu = 7

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 7, fin = 7, milieu = 7

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

11

Algorithmique et programmation structurée

Valeur recherchée : 11 déb = 7, fin = 7, milieu = 7

1	4	5	7	8	9	10	11	12	14	15

10	11	12	14	15

10	11

11

11 = 11 : trouvé

Algorithmique et programmation structurée

Exercice

Écrire une fonction permettant d'effectuer une recherche dichotomique dans un tableau trié.

Algorithmique et programmation structurée

Recherche dichotomique

```
inf <- 0
sup <- n - 1
trouvé <- faux
TantQue (inf <= sup ET trouvé = faux)
  milieu <- (inf + sup) / 2
  Si (x = t[milieu]) alors
    trouvé <- vrai
  Sinon Si (x > t[milieu]) alors
    inf <- milieu + 1
  Sinon
    sup <- milieu 1
  FinSi
FinTantQue
Si (trouvé = vrai) alors
  écrire (x, " : trouvé dans t")
Sinon
  écrire (x, " : non trouvé dans t")
FinSi
```