Algorithmique avancée et programmation C
Remise à Niveau

Nicolas Delestre

INSA
ROUEN NORMANDIE

Plan

INSA ROUEN NORMANDIE

Votre programme s'exécute, mais...

- Connaissez-vous les mécanismes utilisés ?
- Etes vous sûr que le résultat soit juste ?
- Combien de temps devrez vous attendre la fin du calcul ?
- Y a-t-il un moyen pour obtenir le résultat plus vite ?
  - Indépendamment de la machine, du compilateur...
- Qu'est ce qu'un bon ("beau" ?) programme ?

INSA ROUEN NORMANDIE

Critères de qualité

Un programme doit être :

1. lisible
2. fiable
3. maintenable
4. réutilisable
5. portable
6. correct (preuve)
7. efficace (complexité)
8. faire face à des contraintes "économiques"

donc...

On a besoin d'une méthodologie

ROUEN NORMANDIE

Méthodologie générale

La création d'un programme informatique passe par 4 phases :

1. La spécification (ou analyse)
2. La conception préliminaire (ou conception générale)
3. La conception détaillée
4. Le codage

INSA ROUEN NORMANDIE

Spécification 1 / 5

Définition

« Ensemble des activités consistant à définir de manière précise, complète et cohérente ce dont l'utilisateur a besoin » (AFNOR)

La spécification possède deux étapes :

1. Reformulation du problème
2. Formalisation du problème et de sa solution

INSA ROUEN NORMANDIE

Spécification 2 / 5

Reformulation du problème

- Souvent le problème est "mal posé"

Exemple

Rechercher l'indice du plus petit élément d'une suite

7	1	3	1	5
1	*2*	*3*	*4*	*5*
	↑	??	↑	

Spécification 3 / 5

Problème ⇒ énoncé

Énoncé = texte où sont définies sans ambiguïté

- L'entrée (données du problème)
- La sortie (résultats recherchés)
- Les relations (éventuelles) entre les données et les résultats

Dans notre exemple

Soit I l'ensemble des indices des éléments égaux au minimum d'une suite.
Déterminer le plus petit élément de I.

INSA ROUEN NORMANDIE

Spécification 4 / 5

Formalisation

- Traduction de l'énoncé dans un langage qui pourra être « compris » par tout le monde (humains et ordinateurs)
  - Méthode : Analyse descendante
  - Langage : Graphique (simplification et modification de SADT/IDEF0)

Analyse descendante

- Abstraire
  - Repousser le plus loin possible l'écriture de l'algorithme
- Décomposer
  - Décomposer la résolution en une suite de "sous-problèmes" que l'on considère comme résolus
- Combiner
  - Résoudre le problème par combinaison des abstractions des "sous-problèmes"

Spécification 5 / 5

Langage graphique

- On a besoin d'un langage pour exprimer les activités ainsi que leurs interactions
- Langage graphique :
  - Une boîte *nommée* par résolution de problème avec :
    - à gauche les types des données nécessaires à la résolution du problème : information en entrée (le type **EntreeStandard** est optionnel)
    - à droite les types des données résultats de la résolution du problème : information en sortie (le type **SortieStandard** est optionnel)
  - Lorsqu'une résolution A_0 a besoin d'une autre résolution A_1 (notion de dépendance) la boîte décrivant la résolution A_1 est dessinée en dessous de la boite A_0 et une flèche issue de la boîte A_0 va jusqu'à la boîte de A_1

INSA ROUEN NORMANDIE

Exemple 1 / 3

Problème

Afficher les nombres parfaits compris entre 1 et un nombre saisi par l'utilisateur.

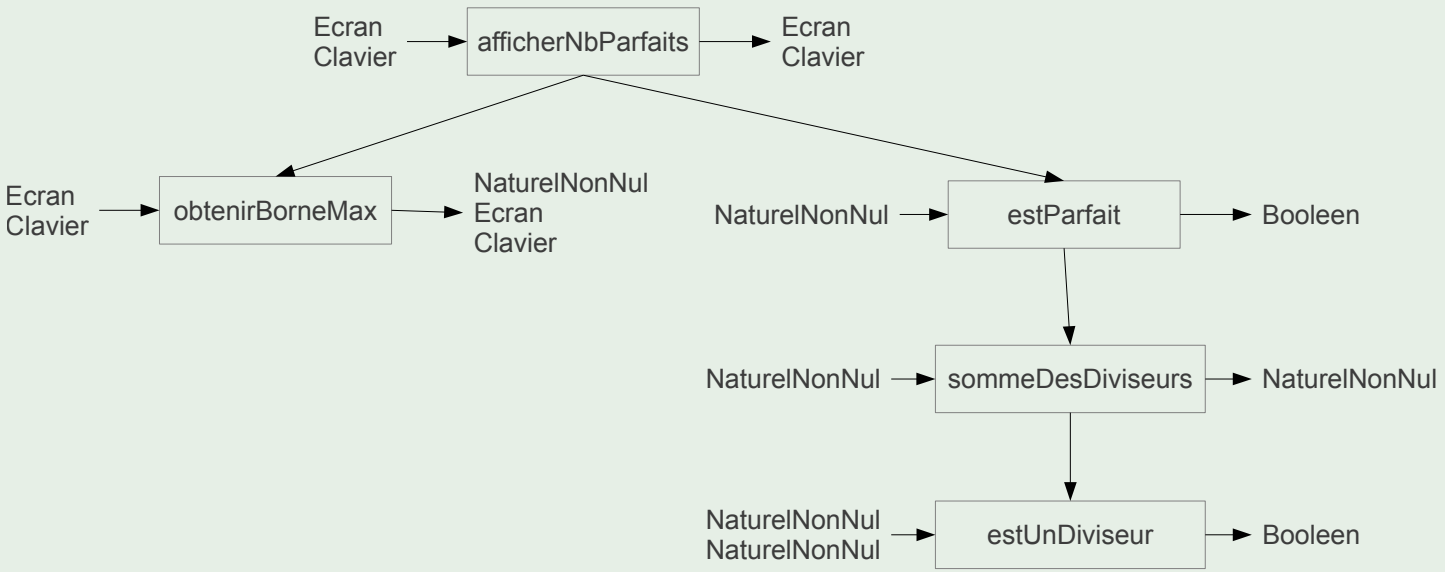
Énoncé

Afficher les nombres parfaits (nombres égaux à la somme de leurs diviseurs) compris entre 1 et un nombre n (naturel ≥ 1) saisi par l'utilisateur.

INSA ROUEN NORMANDIE

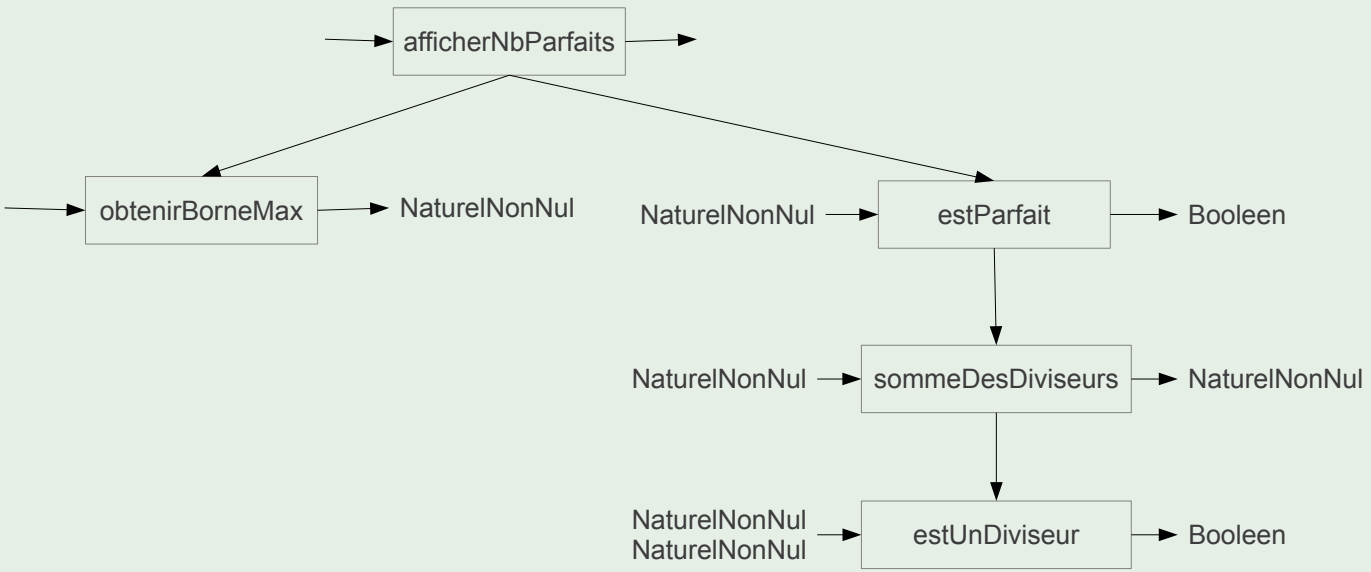
Exemple 2 / 3

Analyse descendante



Exemple 3 / 3

Analyse descendante



Conception préliminaire 1 / 5

Définition

« Ensemble des activités conduisant à l'élaboration de l'architecture du logiciel » (AFNOR)

- Choix d'un paradigme de programmation
- Traduire les fonctions de l'analyse descendante avec les « outils » du paradigme

INSA ROUEN NORMANDIE

Conception préliminaire 2 / 5

Paradigme

« ...style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées ... » (Wikipédia)

Quelques paradigmes de programmation

- **Programmation impérative**
  - **Programmation structurée**
- Programmation orientée objet (Cf. cours UMLP-BD en ASI3.2)
- Programmation déclarative
  - Programmation descriptive (Cf. cours Document en ASI4.2)
  - Programmation fonctionnelle
  - Programmation logique (Cf. cours Document en ASI4.2)

Attention

Un langage peut implanter plusieurs paradigmes

Conception préliminaire 3 / 5

Programmation impérative

« [...]la programmation impérative est un paradigme de programmation qui décrit les opérations en termes d'états du programme et de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. » (Wikipédia)

- Une instruction de base : l'affectation (nommée aussi l'assignation)
- Trois organisations d'instructions :
  - Séquentiel
  - Conditionnel
  - Itératif

INSA ROUEN NORMANDIE

Conception préliminaire 4 / 5

Programmation structurée

- « ...sous-ensemble, ou une branche, de la programmation impérative » (Wikipédia)
- Le programme est décomposé en petits sous-programmes appellés procédures ou fonctions

INSA ROUEN NORMANDIE

Conception préliminaire 5 / 5

- Dans le paradigme de la programmation structurée les « boîtes » (fonctions mathématiques) de l'analyse vont se traduire en :
  - fonction
  - procédure
- On doit préciser comment s'utilise les fonctions de l'analyse :
  - Signature de fonction et de procédure

INSA ROUEN NORMANDIE

Exemple

Fonctions et procédures de la résolution du problème : afficher des nombres parfaits

procédure afficherNombresParfaits **()**
fonction obtenirBorneMax **() : NaturelNonNul**
fonction estParfait **(**nb : **NaturelNonNul) : Booleen**
fonction sommeDesDiviseurs **(**nb : **NaturelNonNul) : NaturelNonNul**
fonction estUnDiviseur **(**a,b : **NaturelNonNul) : Booleen**

INSA ROUEN NORMANDIE

Conception détaillée 1 / 1

Définition

« Ensemble des activités consistant à détailler les résultats de la conception préliminaire, tant sur le plan algorithmique que sur celui de la structure des données, jusqu'à un niveau suffisant pour permettre le codage » (AFNOR)

- Besoin d'utiliser un langage formel indépendant du langage informatique qui sera utilisé : Pseudo-code

INSA ROUEN NORMANDIE

Exemple

Fonction *sommeDesDiviseurs*

```
fonction sommeDesDiviseurs (nb : NaturelNonNul) : NaturelNonNul

    Déclaration   diviseur,somme : NaturelNonNul
debut
    somme ← 1
    pour diviseur ←2 à nb div 2 faire
        si estUnDiviseur(nb,diviseur) alors
            somme ← somme+diviseur
        finsi
    finpour
    retourner somme
fin
```

Codage

Définition

« Activité permettant de traduire le résultat de la conception détaillée en un programme à l'aide d'un langage de programmation donné » (AFNOR)

- Il existe des centaines de langages
- Utiliser celui qui est le plus adapté à votre problème
  - Pour nous le problème est d'utiliser le langage le plus répandu : Langage C

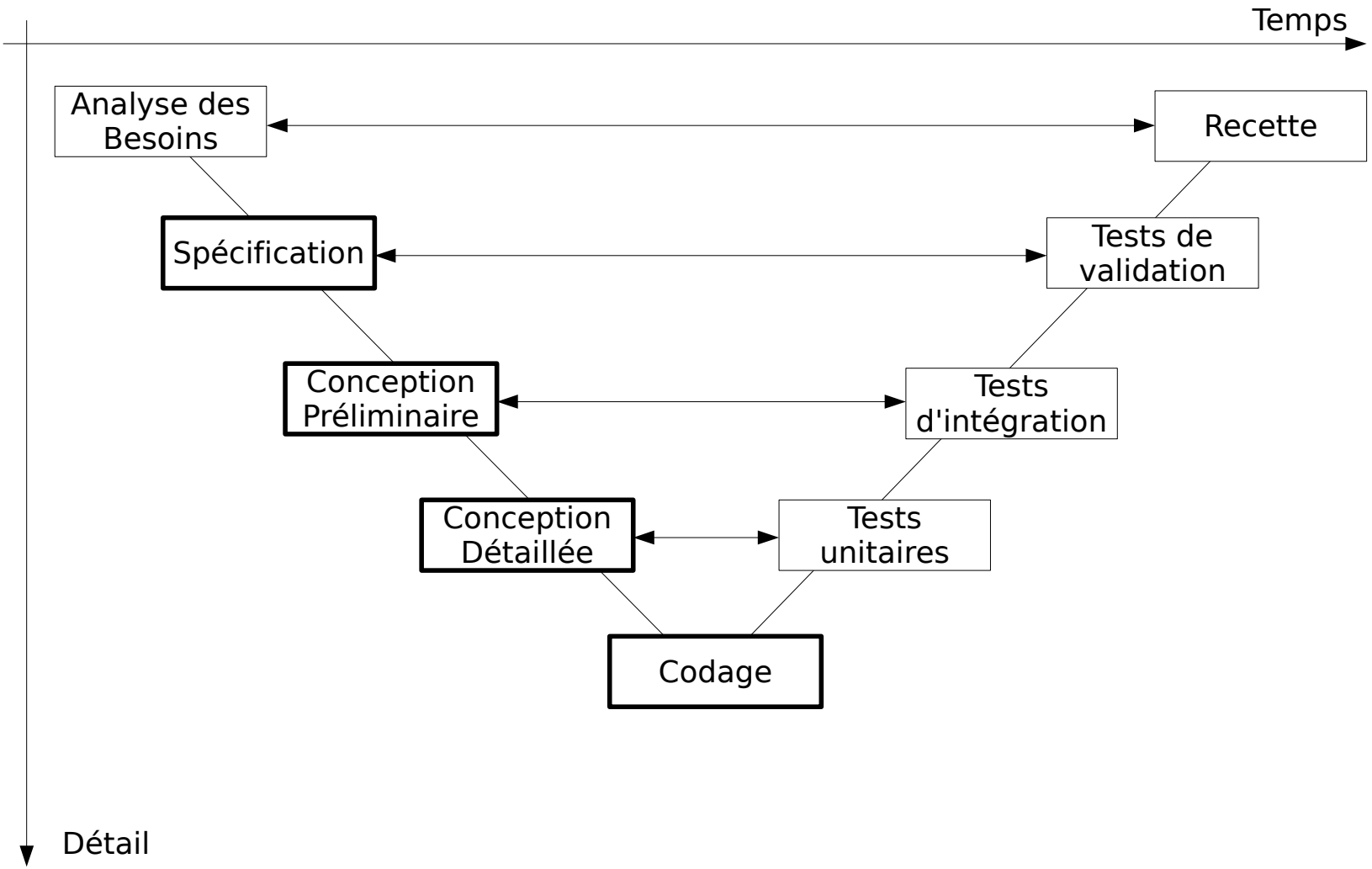
INSA ROUEN NORMANDIE

Exemple

Function *sommeDesDiviseurs*

```
1 int sommeDesDiviseurs(int n) {
2   int diviseur,somme=1;
3   for(diviseur=2;i<=n/2;diviseur++)
4     if (estDivisible(n,diviseur))
5       somme=somme+diviseur;
6   return somme;
7 }
```

Le cycle en V


Temps
Analyse des Besoins
Recette
Spécification
Tests de validation
Conception Préliminaire
Tests d'intégration
Conception Détaillée
Tests unitaires
Codage
Détail

Pseudo-code

Objectifs

Fournir une solution au problème, il faut donc un moyen d'exprimer cette solution à l'aide d'un langage qui doit être :

- formel (pas d'ambiguïtés),
- lisible et concis,
- indépendant de tous langages informatiques
- qui reprend les concepts des langages impératifs :
  - les entités manipulées sont désignées et typées
  - suite d'actions modifiant l'état d'un programme, pour passer :
    - de l'état initial -le problème -
    - à l'état final - la solution au problème-

INSA ROUEN NORMANDIE

Types

Objectifs

Classer les objets selon :

- Ensemble des valeurs que peut prendre l'objet
- Ensemble des opérations permises sur ces valeurs

- Tous les types ont un nom
- Tous les objets ont un type (qui doit être déclaré)

INSA ROUEN NORMANDIE

Classification des types

On peut classer les types suivant deux points de vue

Prédéfinis / définis explictement

- Prédéfinis : Booléen, Naturel [a], Entier, Réel [b], Caractère, Chaîne de caractères
- Définis explicitement (instruction "Type") : intervalle, énuméré, tableaux, structures

---

a. NaturelNonNul
b. RéelPositif, RéelPositifNonNul, RéelNégatif, RéelNégatifNonNul

Élémentaires / composites

- Booléen, Naturel, Entier, Réel, Caractère, intervalle, énuméré
- Chaîne de caractères, tableaux, structures

Variables

Définition

- Récipient pour des objets dont le contenu (= valeur) peut changer au cours de l'exécution du programme
- Abstraction d'un emplacement mémoire de la machine de Von Neumann

- Assignation de valeurs à des cellules de la mémoire
- Possibilité de regroupement dans des agrégats : tableau, structure

Important

Toute variable utilisée doit être déclarée

INSA ROUEN NORMANDIE

Constante

Définition

- Entité qui n'évolue pas
- Deux types de constantes :
  - Constante implicite
  - Constante explicite
- Une constante a un type et une valeur

Constante implicite

Valeur de fait : 2, 2.3, "ABCD", 'a'

Constante explicite

Constante définie explicitement, par exemple :
Constante MAX = 10

ROUEN NORMANDIE

Opération, expression

Opération

Une opération est l'association :

- d'un opérateur
- d'une ou plusieurs opérandes (suivant la cardinalité de l'opérateur). Les opérandes sont des expressions.

Une opération a :

- un type
- une valeur

Expression

Une expression est une constante ou une variable ou une opération. Donc une expression a :

- un type
- une valeur

Opérateurs des types simples

Type des opérandes	Opérateurs disponibles	Type résultat
Booléen	non, et, ou, $=$, $\neq$	Booléen
Entier, Naturel	$+$, -, *, div, mod	Entier
	/	Réel
	$=$, $\neq$, $<$, $>$, $\leq$, $\geq$	Booléen
Réel	$+$, -, *, /	Réel
	$=$, $\neq$, $<$, $>$, $\leq$, $\geq$	Booléen
Caractère	succ, pred	Caractère
	ord	Naturel
Naturel	char	Caractère
	$=$, $\neq$, $<$, $>$, $\leq$, $\geq$	Booléen
Chaîne de caractères	$+$	Chaîne
	$=$, $\neq$, $<$, $>$, $\leq$, $\geq$	Booléen

INSA ROUEN NORMANDIE

Types composites

Objectif

Pouvoir rassembler au sein d'une variable plusieurs valeurs :

- plusieurs valeurs de même type : les tableaux
- plusieurs valeurs de types différents : les structures

Par nature

- Construit à partir d'autres types
- Notion de composant
  - Type des composants
  - Organisation des composants
  - Accès aux composants

Il faut distinguer les opérations :

- sur le groupe
- sur les composants du groupe

Tableaux 1 / 2

Caractéristiques

- Collection indicée d'éléments de même type
- Taille fixe : cardinal (Indice)
- Accès aux composants
  - Elément de Indice (= expression d'un type ordonné)
  - Accès direct (temps d'accès constant)

Déclaration (tableau à une dimension)

Tableau[indiceDebut..indiceFin**] de** Type des éléments

Actions de base sur tableaux de même type

- $\leftarrow$
- Opérateurs $=$ et $\neq$
- Accès aux éléments

Tableaux 2 / 2

Exemple

- Déclaration
  Constante MAX = 10
  Type Vecteur = **Tableau[**1..MAX**] de Reel**
  v1 : Vecteur
  v2 :**Tableau[**1..MAX**] de Reel**
- Accès au i^{eme} élément
  x $\leftarrow$ v1[i]
  v2[i] $\leftarrow$ 10

Un tableau est une structure statique :

Nombre d'éléments maximum
$\geq$
Nombre d'éléments pertinents

Structure 1 / 2

Caractéristiques

- Type structure = produit cartésien de types
- Valeur du type structure = élément d'un produit cartésien de types
- Représenter un "objet" composé d'éléments (champs ou attributs) de différents types

Déclaration

Type nom **= Structure**
    $champ_1$: $type_1$
    $champ_2$: $type_2$
    ...
    $champ_n$: $type_n$
finstructure

Structure 2 / 2

Actions de base

- $\leftarrow$
- Opérateurs $=$ et $\neq$
- Accès aux éléments (notation pointée)

Exemple

Type Jour = (1..31)
Type Mois = (1..12)
Type Date = **Structure**
    jour : Jour
    mois : Mois
    annee : **Entier**
finstructure
d : Date
d.jour $\leftarrow$ 1

Actions de base 1 / 2

Action / Instruction

Élément, qui lorsqu'il est interprété, modifie l'état d'un programme

Il y a trois actions de base en algorithmique : $\leftarrow$, **lire** et **écrire**

$\leftarrow$

- **Syntaxe** : *variable(s)* $\leftarrow$ *expression*
- **But** : affecte la valeur de l'expression à la variable
  variable et *expression* doivent être du même type
- **Note** : si l'expression est une fonction qui retourne plusieurs valeurs, il peut y avoir plusieurs variables à gauche du $\leftarrow$

Exemple

a $\leftarrow$ 2*b

diviseur, reste $\leftarrow$ divMod(10,3)

Actions de base 2 / 2

lire

- **Syntaxe** : ***lire***(*var1, var2, ...*)
- **But** : demande à l'utilisateur de saisir des valeurs à partir de l'entrée standard

Exemple

lire(a,b)

écrire

- **Syntaxe** : ***écrire***(*expression1,expression2,...*)
- **But** : affiche les valeurs des expressions sur la sortie standard

Exemple

ecrire("a = ",a," et b = ",b)

Écrire un algorithme revient à expliciter des schémas d'actions
Trois schémas d'actions disponibles :

1. Schéma séquentiel
2. Schéma conditionnel
3. Schéma itératif

INSA ROUEN NORMANDIE

Schéma séquentiel

Schéma séquentiel

- Composition séquentielle d'action
  - $A = (A_1, A_2, \ldots, A_n)$
- Cette liste d'actions peut être considérée comme une seule action
  - l'interprétation de l'action A revient à interpréter successivement les actions A_i pour $i \in [1..n]$

Exemple

temp $\leftarrow$ a

a $\leftarrow$ b

b $\leftarrow$ temp

Schéma conditionnel 1 / 3

Objectif

- Interpréter une action A si et seulement si une condition C (expression booléenne) est valuée à VRAI
  Si C est évaluée à FAUX on interprete une action A' si elle est présente

Syntaxe

si C **alors**
    A
finsi

si C **alors**
    A
sinon
    A'
finsi

Schéma conditionnel 2 / 3

Exemple

si a<b **alors**
    max ← b
sinon
    max ← a
finsi

Exemple

max ← a
si a<b **alors**
    max ← b
finsi

INSA ROUEN NORMANDIE

Schéma conditionnel 3 / 3

Cas où

Lorsque l'on veut comparer la même variable, v, (de type Entier, Naturel, Caractère) à des valeurs successives on peut remplacer la succession de *si...alors...sinon* par :

cas où v **vaut**

v_1 :
    A_1
v_2 :
    A_2
...
v_n :
    A_n
autre :
    A_m

fincas

Schéma itératif

Objectif

Répéter une même action un certain nombre de fois

Deux cas possibles :

- On sait à l'écriture de l'algorithme le nombre d'itérations : Boucle déterministe
  - Instruction *pour*
- On ne sait pas à l'écriture de l'algorithme le nombre d'itérations : Boucle indéterministe
  La prochaine itération est conditionnée (expression booléenne)
  - Instruction *tant que* ou *répéter...jusqu'à ce que*

INSA ROUEN NORMANDIE

Boucle déterministe

Pour

pour variable ← borneDépart **à** borneArrivée [**pas de** p (par défaut p=1)] **faire**
    A
finpour

Exemple

somme ← 0
pour i ←1 **à** n **faire**
    somme ← somme+i
finpour

Boucles indéterministes

Tant que

tant que C_t **faire**
    A_t
fintantque

Répéter ...jusqu'à ce que

repeter
    A_r
jusqu'a ce que C_r

Remarques

- A_t (respectivement A_r) doit modifier les variables utilisées dans la condition C_t (respectivement C_r)
- $C_r =$ non C_t

Procédure / Fonction

À partir du schéma de l'analyse descendante, on identifie les types des sous-programmes :

- fonction
  - sous programme qui calcule quelque chose (vision mathématique) : retourne une valeur
- procédure
  - sous programme qui fait quelque chose

Paramètre formel

Paramètre déclaré dans la signature du sous-programme

Paramètre effectif (argument)

Paramètre utilisé réellement lors de l'invocation du sous-programme

Passage de paramètre 1 / 2

Lorsque l'on appelle un sous-programme on associe un paramètre effectif à un paramètre formel (suivant l'ordre de la déclaration des paramètres formels)

Définition

On nomme passage de paramètre, l'association de valeur qui est utilisée entre le paramètre effectif et le paramètre formel

Passage de paramètre en **Entrée**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à l'entrée du sous-programme
En théorie le paramètre formel n'est pas modifié par le sous programme, jamais à gauche de $\leftarrow$

ROUEN NORMANDIE

Passage de paramètre 2 / 2

Passage de paramètre en **Sortie**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à la sortie du sous-programme
Seules des variables peuvent être utilisées comme paramètre effectif
En théorie le paramètre formel, la première fois qu'il est utilisé, ne peut se trouver qu'à la gauche de ← et doit obligatoirement apparaitre

Passage de paramètre en **Entrée/Sortie**

Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à l'entrée du sous-programme
Les valeurs du paramètre effectif et du paramètre formel sont les mêmes à la sortie du sous-programme
Seules des variables peuvent être utilisées comme paramètre effectif

ROUEN NORMANDIE

Passage de paramètre et sous programme

Fonction

Seul le passage de paramètre en Entrée est autorisé pour les fonctions
Aucune information n'est donc ajoutée dans la signature de la fonction

Procédure

Les paramètres admettent les trois types de passage de paramètre, on préfixe donc le paramètre formel dans la signature par :

- **Entrée** ou **E** pour un passage de paramètre en entrée
- **Sortie** ou **S** pour un passage de paramètre en sortie
- **Entrée/Sortie** ou **E/S** pour un passage de paramètre en entrée/sortie

INSA ROUEN NORMANDIE

Fonction

fonction *nom de la fonction* **(***[paramètre(s) de la fonction]***) :** *types des valeurs retournées*

⌊**précondition(s)** *préconditions sur les paramètres*

Déclaration *variable locale 1 : type 1 ; ...*

debut

instructions de la fonction avec au moins une fois l'instruction ***retourner***

fin

Procédure

procédure *nom de la procédure* **(**[**E** *paramètre(s) en entrée ;*][**S** *paramètre(s) en sortie ;*][**E/S** *paramètre(s) en entrée/sortie*]**)**

⌊**précondition(s)** *préconditions sur les paramètres*

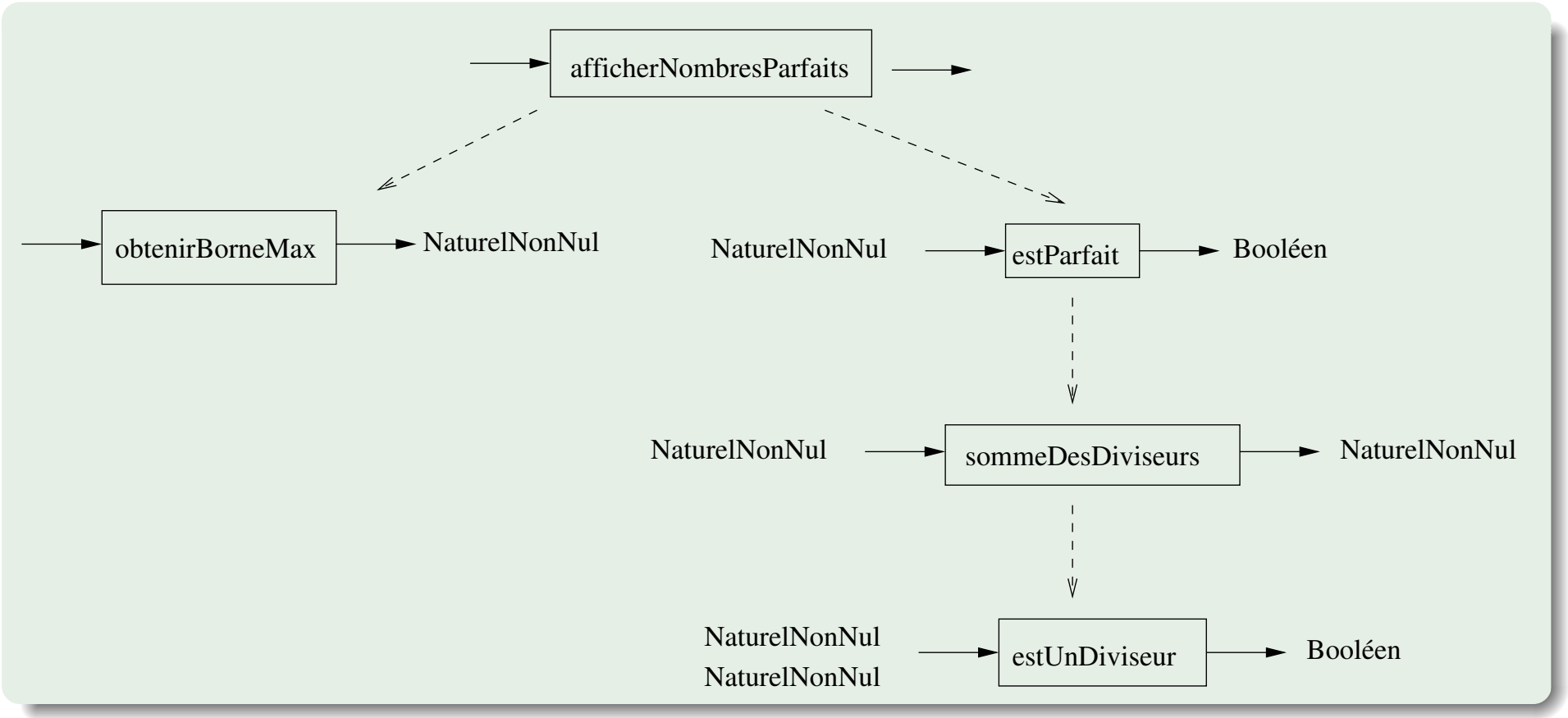
Déclaration *variable(s) locale(s)*

debut

instructions de la procédure

fin

Reprenons l'exemple



estUnDiviseur et *sommeDesDiviseurs*...

```
fonction estUnDiviseur (a, b : NaturelNonNul) : Booleen
debut
    retourner a mod b=0
fin
fonction sommeDesDiviseurs (nb : NaturelNonNul) : NaturelNonNul

    Déclaration  diviseur, somme : NaturelNonNul
debut
    somme ← 1
    pour diviseur ←2 à nb div 2 faire
        si estUnDiviseur(nb,diviseur) alors
            somme ← somme+diviseur
        finsi
    finpour
    retourner somme
fin
```

estParfait et *obtenirBorneMax*...

```
fonction estParfait (nb : NaturelNonNul) : Booleen
debut
    retourner nb=sommeDesDiviseurs(nb)
fin
fonction obtenirBorneMax () : NaturelNonNul
    Déclaration  resultat : Entier
debut
    repeter
        ecrire("Valeur maximale d'affichage des nombres parfaits")
        lire(resultat)
    jusqu'a ce que resultat>1
    retourner entierEnNaturelNonNul(resultat)
fin
```

afficherNombresParfaits...

```
procédure afficherNombresParfaits ()
    Déclaration   i,max : NaturelNonNul
debut
    max ← obtenirBorneMax()
    pour i ←1 à max faire
        si estParfait(i) alors
            ecrire(i)
        finsi
    finpour
fin
```

Algorithmes classiques

L'algorithmique c'est comme la cuisine...

- Cela peut demander de l'imagination
- Mais cela demande surtout de la méthode et de la pratique
- Il existe plusieurs types de problème que l'on rencontre constamment, par exemple :
  - Problèmes de parcours
  - Problèmes de recherche
  - Problèmes d'optimisation
- Pour chaque problème il existe des algorithmes types qu'il faut adapter au contexte

INSA ROUEN NORMANDIE

Problème de parcours

Exemples

- Compter le nombre de voyelles dans une chaîne de caractères
- Mettre en majuscule les lettres d'une chaîne de caractères
- Calculer la somme des diviseurs d'un nombre
- Afficher les éléments d'un tableau

Algorithmes

- Initialiser une variable si besoin
- Parcourir l'ensemble des éléments du problème et agir sur chaque élément

INSA ROUEN NORMANDIE

Problème de recherche

Deux types de recherche

1. Rechercher le premier élément ayant une certaine caractéristique
2. Rechercher un individu qui se caractérise au regard des autres éléments

Exemples

- Trouver la première voyelle d'une chaîne de caractère (cas 1)
- Trouver le plus petit élément d'un tableau (cas 2)

Algorithmes

1. Parcourir les éléments du problème et s'arrêter dès qu'un élément résout le problème
2. Faire une hypothèse, et pour chaque élément essayer de remettre en cause l'hypothèse

Problème d'optimisation

Exemples

- Rendu de monnaie
- Chemin le plus court pour aller d'un point A à un point B

Algorithmes

- De loin les problèmes les plus complexes (souvent des problèmes NP[a])
- Quelques algorithmes types (algorithme glouton, A*, simplexe, etc.)

a. *Nondeterministic Polynomial time* : la vérification d'une solution est en complexité polynomiale, mais la recherche d'une solution est en souvent en complexité exponentielle

INSA ROUEN NORMANDIE

Calcul de complexité

Objectifs

Indépendamment de la machine, du compilateur...
Complexité :

- Taille du problème : n
- Nombre d'opérations significatives : $T(n)$
- Taille mémoire nécessaire : $M(n)$

Notations asymptotiques

- $f(n) = O(g(n))$: borne asymptotique supérieure (au pire)
- $f(n) = \Omega(g(n))$: borne asymptotique inférieure (au mieux)
- $f(n) = \theta(g(n))$: borne approchée asymptotique (en moyenne)

INSA ROUEN NORMANDIE

Taille du problème

Que représente n ?

nombre nombre de chiffres pour les représenter
Attention à bien prendre en compte (surtout dans les itérations) que la valeur maximale du nombre nb est $base^n - 1$ *(la plupart du temps* $base = 2$*)*

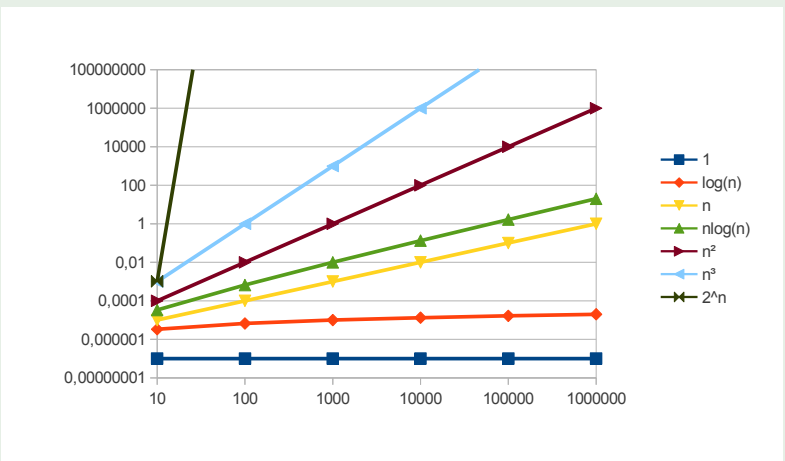
tableau nombre d'éléments

graphe nombre de nœuds et/ou d'arcs

Exemple de complexité

- 10^6 opérations par seconde
- N = taille du problème
- C = complexité de l'algorithme de traitement

N x C	1	$log_2 n$	n	$nlog_2 n$	n^2	n^3	2^n
10^2	$< 1\ \mu s$	$6{,}6\ \mu s$	0,1 ms	0,66 ms	10 ms	1 s	4.10^6 a
10^3	$< 1\ \mu s$	$9{,}9\ \mu s$	1 ms	9,9 ms	1 s	16,6 mn	?
10^4	$< 1\ \mu s$	$13{,}3\ \mu s$	10 ms	0,13 s	1,5 mn	11,5 j	?
10^5	$< 1\ \mu s$	$16{,}6\ \mu s$	0,1 s	1,66 s	2,7 h	31,7 a	?
10^6	$< 1\ \mu s$	$19{,}9\ \mu s$	1 s	19,9 s	11,5 j	31700 a	?



Calcul de complexité 1 / 2

Notation

- Soit A une action (ou instruction), on note :
  - $O(A^o)$ sa complexité dans le pire des cas
  - $\Omega(A^\Omega)$ sa complexité dans le meilleur des cas

Cas d'un schéma séquentiel $(A_1, A_2, \ldots, A_n)$

- Dans le pire des cas : $O(max(A_1^o, A_2^o, \ldots, A_n^o))$
- Dans le meilleur des cas : $\Omega(max(A_1^\Omega, A_2^\Omega, \ldots, A_n^\Omega))$

Schéma conditionnel (C et A_1 ou A_2)

- Dans le pire des cas : $O(max(C^o, A_1^o, A_2^o))$
- Dans le meilleur des cas : $\Omega(max(C^\Omega, min(A_1^\Omega, A_2^\Omega)))$

INSA ROUEN NORMANDIE

Calcul de complexité 2 / 2

Schéma itératif déterministe (A)

- Dans le pire des cas : $O(f(n) * A^o)$
- Dans le meilleur des cas : $\Omega(f(n) * A^\Omega)$
  $f(n)$ est le nombre d'itérations

Schéma itératif indéterministe (C et A)

- Dans le pire des cas : $O(f(n) * max(C^o, A^o))$
  $f(n)$ est le nombre d'itérations maximal
- Dans le meilleur des cas : $\Omega(C^\Omega)$ (tant que) ou $\Omega(max(C^\Omega, A^\Omega))$ (répéter jusqu'à ce que)

Complexité des instructions et opérations de base

- Toutes les instructions et opérations de base sont considérées comme étant en $O(1)$ et $\Omega(1)$

Exemple de calcul complexité 1 / 5

estUnDiviseur, $T(n) =$

```
fonction estUnDiviseur (a, b : NaturelNonNul) : Booleen
debut
    retourner a mod b=0
fin
```

$O(1)$ et $\Omega(1)$

Exemple de calcul complexité 2 / 5

sommeDesDiviseurs, $T(n) =$

```
fonction sommeDesDiviseurs (nb : NaturelNonNul) : NaturelNonNul
    Déclaration  diviseur, somme : NaturelNonNul
debut
    somme ← 1
    pour diviseur ←2 à nb div 2 faire
        si estUnDiviseur(nb,diviseur) alors
            somme ← somme+diviseur
        finsi
    finpour
    retourner somme
fin
```

$O(2^n)$ et $\Omega(2^n)$

Exemple de calcul complexité 3 / 5

estParfait, $T(n) =$

```
fonction estParfait (nb : NaturelNonNul) : Booleen
debut
    retourner nb=sommeDesDiviseurs(nb)
fin
```

$O(2^n)$ et $\Omega(2^n)$

obtenirBorneMax, $T(n) =$

```
fonction obtenirBorneMax () : NaturelNonNul
    Déclaration  resultat : NaturelNonNul
debut
    repeter
        ecrire("Valeur maximale d'affichage des nombres parfaits")
        lire(resultat)
    jusqu'a ce que resultat>1
    retourner entierEnNaturelNonNul(resultat)
fin
```

$O(1)$ et $\Omega(1)$

Exemple de calcul complexité 4 / 5

afficherNombresParfaits, $T(n) =$

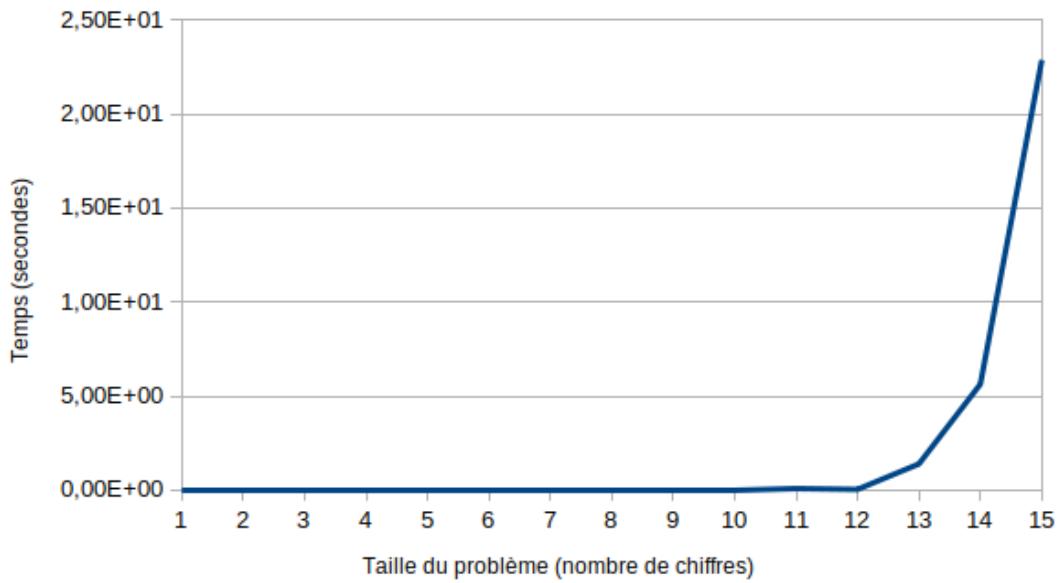
```
procédure afficherNombresParfaits ()
    Déclaration  i,max : NaturelNonNul
debut
    max ← obtenirBorneMax()
    pour i ←1 à max faire
        si estParfait(i) alors
            ecrire(i)
        finsi
    finpour
fin
```

$O(n^2)$ et $\Omega(n^2)$

Exemple de calcul complexité 5 / 5

Sur l'exemple précédent, $T(n) =$

- `estUnDiviseur` : $O(1)$,$\Omega(1)$ et $\theta(1)$
- `sommeDesDiviseurs` : $O(2^n)$,$\Omega(2^n)$ et $\theta(2^n)$
- `estParfait` : $O(2^n)$,$\Omega(2^n)$ et $\theta(2^n)$
- `obtenirBorneMax` : $O(1)$,$\Omega(1)$ et $\theta(1)$
- `afficherNombresParfaits` : $O(2^n)$,$\Omega(2^n)$ et $\theta(2^n)$

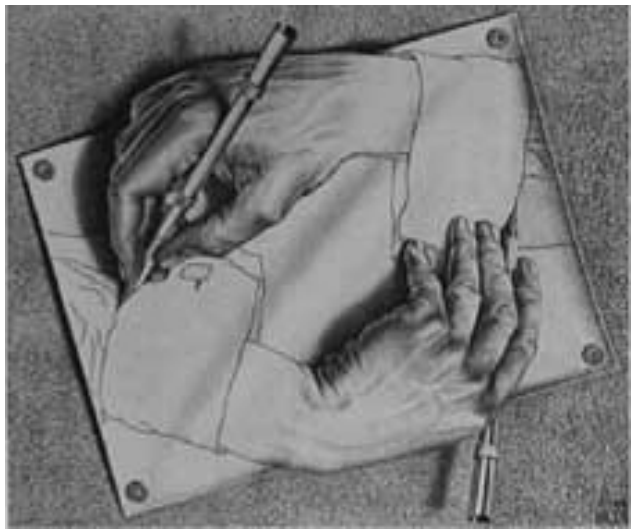


INSA ROUEN NORMANDIE

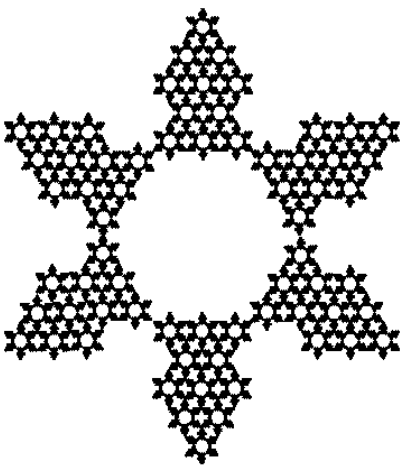
Récursivité

Définition

Une entité est récursive lorsqu'on l'utilise pour la définir



Drawing Hands, Escher (1948)



http://www.russie.net/russie/art_matriochka.htm

INSA ROUEN NORMANDIE

Exemples 1 / 2

Factorielle

$$\begin{cases} 0! = 1! = 1 \\ n! = n(n-1)! \end{cases}$$

Suite de fibonacci

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(n) = F(n-1) + F(n-2), n > 1 \end{cases}$$

Poupée russe

Une poupée russe est

- une poupée “pleine”
- une poupée “vide” contenant une poupée russe

Exemples 2 / 2

Factorielle

```
fonction fact (n : Naturel) : Naturel
debut
    si n=0 ou n=1 alors
        retourner 1
    sinon
        retourner n*fact(n-1)
    finsi
fin
```

Récursivité terminale

Définition

L'appel récursif est la dernière instruction et elle est isolée

plus(a,b)

```
fonction plus (a,b : Naturel) : natuel
debut
    si b=0 alors
        retourner a
    sinon
        retourner plus(a+1,b-1)
    finsi
fin
```

plus(4,2)=plus(5,1)=plus(6,0)=6

Récursivité non terminale

Définition

L'appel récursif n'est pas la dernière instruction et/ou elle n'est pas isolée (fait partie d'une expression)

plus(a,b)

```
fonction plus (a,b : Naturel) : natuel
debut
    si b=0 alors
        retourner a
    sinon
        retourner 1+plus(a,b-1)
    finsi
fin
```

plus(4,2)=1+plus(4,1)=1+1+plus(4,0)=1+1+4=6

Méthode

Pour écrire un algorithme récursif il faut analyser le problème pour :

- identifer le ou les cas particuliers
- identifier le cas général qui effectue la récursion

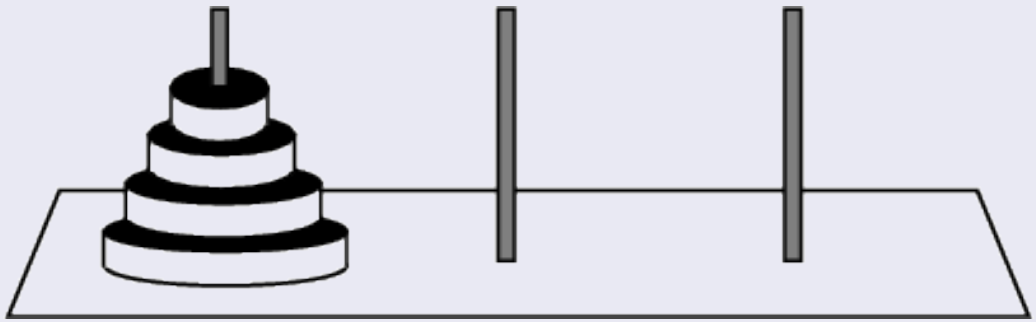
Surtout

Lorsque l'on écrit un algorithme récursif, lors de l'appel récursif, on se positionne en tant qu'utilisateur de l'algorithme : on considère donc que le problème est résolu

INSA ROUEN NORMANDIE

Les tours de Hanoï 1 / 4

Présentation



Les tours de hanoï est un jeu solitaire dont l'objectif est de déplacer les disques qui se trouvent sur une tour (par exemple ici la première tour, celle la plus à gauche) vers une autre tour (par exemple la dernière, celle la plus à droite) en suivant les règles suivantes :

- on ne peut déplacer que le disque se trouvant au sommet d'une tour ;
- on ne peut déplacer qu'un seul disque à la fois ;
- un disque ne peut pas être posé sur un disque plus petit.

Les tours de Hanoï 2 / 4

Opérations disponibles

procédure dépilerTour **(E/S** t : TourDeHanoi,**S** d : Disque**)**

⌊**précondition(s)** non estVide(t)

procédure empilerTour **(E/S** t : TourDeHanoi,**E** d : Disque**)**

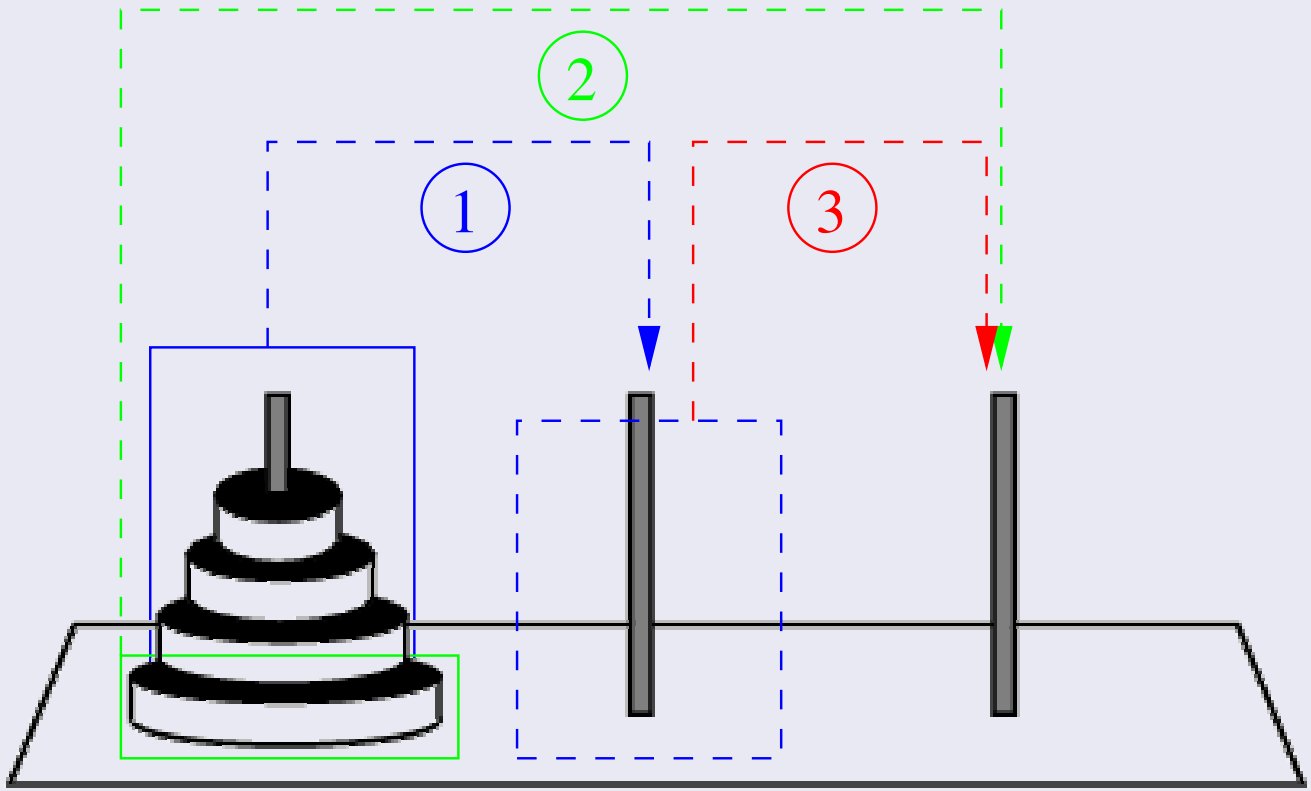
Objectif

procédure resoudreToursDeHanoi **(E** nbDisquesADeplacer : **Naturel**,
E/S source, destination, intermediaire : TourDeHanoi**)**

INSA ROUEN NORMANDIE

Les tours de Hanoï 3 / 4

Analyse du problème



Les tours de Hanoï 4 / 4

Solution

procédure resoudreToursDeHanoi **(E** nbDisquesADeplacer : **Naturel**, **E/S** source, destination, intermediaire : TourDeHanoi**)**

Déclaration d : Disque

debut

si nbDisquesADeplacer>0 **alors**

resoudreToursDeHanoi(nbDisquesADeplacer-1, source, intermediaire, destination)

depiler(source,d)

empiler(destination,d)

resoudreToursDeHanoi(nbDisquesADeplacer-1, intermediaire, destination, source)

finsi

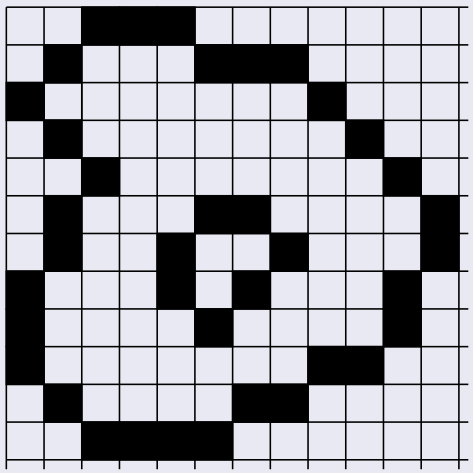
fin

INSA ROUEN NORMANDIE

Remplir une zone graphique 1 / 5

Présentation

- Un écran graphique est un quadrillage
- Chaque intersection de ce quadrillage est un `pixel` qui peut être colorisé



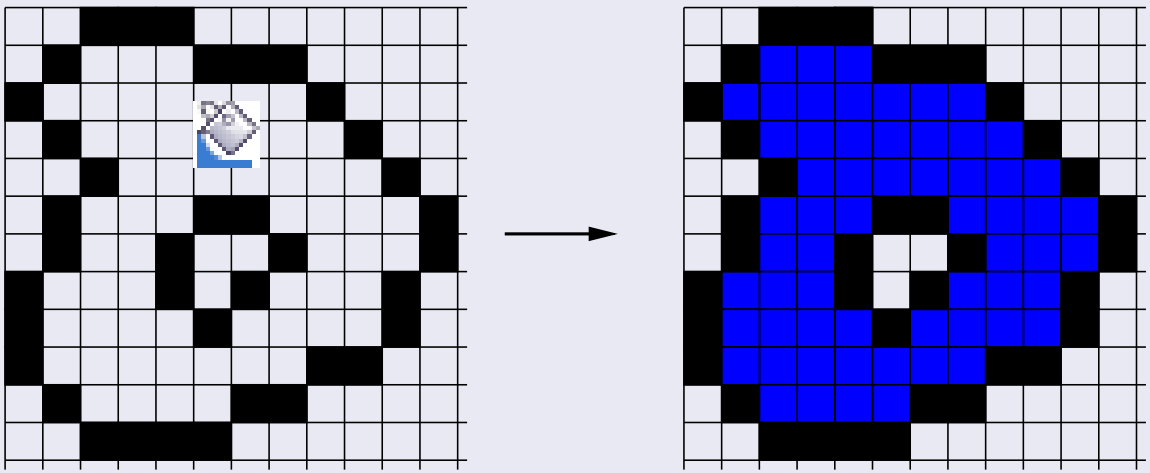
Opérations disponibles

- **procédure** fixerCouleurPixel **(E/S** e : Ecran,**E** x,y : **Naturel**, c : Couleur**)**
- **fonction** obtenirCouleurPixel **(**e : Ecran,x,y : **Naturel) :** Couleur

Remplir une zone graphique 2 / 5

Objectif

- Proposer le corps de la procédure suivante qui permet de remplir une zone
  - **procédure** remplir **(E/S** e : Ecran,**E** x,y : **Naturel**, ancienneCouleur, nouvelleCouleur : Couleur**)**



INSA ROUEN NORMANDIE

Remplir une zone graphique 3 / 5

Analyse du problème

- Remplir une zone consiste à **changer** la couleur de certains pixels en commençant par celui qui est donné :
  - Si le pixel de coordonnée (x, y) est d'une couleur différente de *ancienneCouleur*
    - **Ne rien faire**
  - Si le pixel de coordonnée (x, y) est de la même couleur que *ancienneCouleur*
    - Changer la couleur de ce pixel
    - Tenter de changer la couleur (**remplir**) des points qui se trouvent autour

INSA ROUEN NORMANDIE

Remplir une zone graphique 4 / 5

Solution

```
procédure remplir (E/S e : Ecran,E x,y : Naturel, ancienneCouleur,
nouvelleCouleur : Couleur)
debut
    si obtenirCouleurPixel(e,x,y)=ancienneCouleur alors
        fixerCouleurPixel(e,x,y,nouvelleCouleur)
        remplir(e,x,y-1,ancienneCouleur,nouvelleCouleur)
        remplir(e,x,y+1,ancienneCouleur,nouvelleCouleur)
        remplir(e,x-1,y,ancienneCouleur,nouvelleCouleur)
        remplir(e,x+1,y,ancienneCouleur,nouvelleCouleur)
    finsi
fin
```

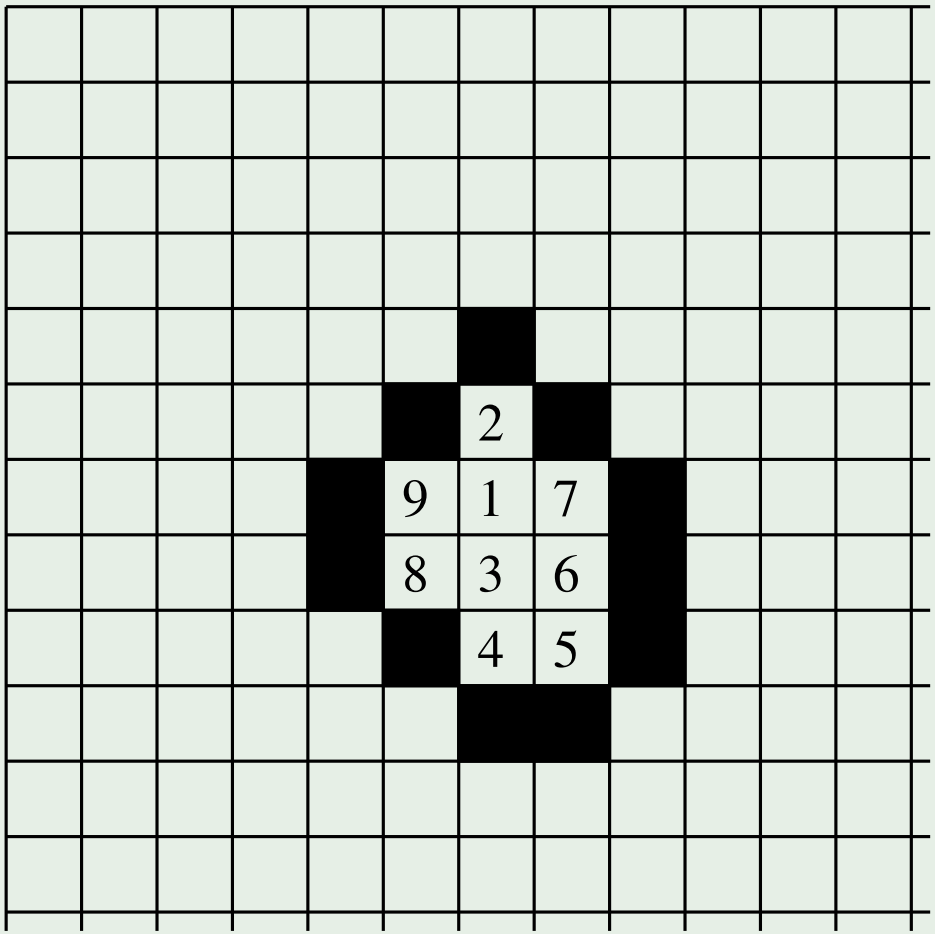
Note

On pourrait améliorer l'algorithme en vérifiant qu'on ne "sort" pas de l'écran

Remplir une zone graphique 5 / 5

Exemple d'ordre de changements de couleur des pixels

On considère que le (0,0) est en haut à gauche :



Conclusion

En conclusion

- Les algorithmes récursifs sont simples (c'est simplement une autre façon de penser)
- Les algortihmes récursifs permettent de résoudre des problèmes complexes
- Il existe deux types de récursivités :
  - terminale, qui algorithmiquement peuvent être transformée en algorithme non récursif
  - non terminale
- Les algorithmes récursifs sont le plus souvent plus gourmands en ressource que leurs équivalents itératifs

INSA ROUEN NORMANDIE

Les Tris

Qu'est ce qu'un tri ?

- C'est un algorithme :
  - Entrée : Séquence de n éléments (dont le type possède un ordre total) $< a1, a2, \ldots, an >$
  - Sortie : Permutation (réarrangement) $< a'1, a'2, ..., a'n >$, telle que : $a'1 \leq a'2 \leq \ldots \leq a'n$
- Tris sur place : Jamais plus d'un nombre constant d'éléments du tableau d'entrée n'est stocké hors du tableau
- Tous les tris ont la même signature :

  procédure tri **(E/S** t :**Tableau[**1..MAX**] d'**Element,**E** nb :**NaturelNonNul)**

  - pour simplifier dans la suite de ce document nous utilisons des tableaux d'entiers, donc :

    procédure tri **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** nb :**NaturelNonNul)**

INSA ROUEN NORMANDIE

Un exemple de tri : Le tri à bulles 1 / 2

Principe

Tant que le tableau n'est pas trié ($\exists i \in [1..nb-1], t[i] > t[i+1]$)

- on parcourt le tableau (i variant de 1 à $nb-1$) et à chaque fois que $t[i] > t[i+1]$ on échange $t[i]$ et $t[i+1]$

INSA ROUEN NORMANDIE

Un exemple de tri : Le tri à bulles 2 / 2

l'algorithme

```
procédure tri (E/S t :Tableau[1..MAX] d'Entier,E nb :Naturel)
    Déclaration  estTrie : Booleen
                 i : NaturelNonNul
debut
    repeter
        estTrie ← VRAI
        pour i ←1 à nb-1 faire
            si t[i]>t[i+1] alors
                echanger(t[i],t[i+1])
                estTrie ← FAUX
            finsi
        finpour
    jusqu'a ce que estTrie
fin
```

Complexité

$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$

INSA ROUEN NORMANDIE

Topologie des tris les plus courants

On peut classer les algortihmes de tri en fonction

- **des méthodes employées (itératif, récursif)**
- des complexités (n^2 ou $nlog_2 n$)

INSA ROUEN NORMANDIE

Principes des tris itératifs

Principes

On va parcourir entièrement le tableau, en appliquant à chaque itération i l'une des deux stratégies suivantes :

- on recherche l'élément qui doit se placer à la i^{eme} place
  - **Tri par sélection**
- on recherche la place où va être positionné le i^{eme} élément
  - **Tri par insertion**

INSA ROUEN NORMANDIE

Le tri par sélection 1 / 2

Principe (par minimum successif)

On parcourt le tableau t (i variant de 1 à $nb-1$) et à chaque itération i

- on détermine l'indice j du plus petit entier de t sur l'intervalle $[i..n]$
- on échange $t[i]$ et $t[j]$

Complexité

$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$

Le tri par sélection 2 / 2

algorithme

fonction indiceDuMinimum **(**t :**Tableau[**1..MAX**] d'Entier**, borneInf,borneSup :**Naturel) :** **Naturel**

⌊**précondition(s)** borneInf $\leq$ borneSup

Déclaration i,resultat : **Naturel**

debut
    resultat $\leftarrow$ borneInf
    pour i $\leftarrow$borneinf+1 **à** borneSup **faire**
        si t[i]<t[resultat] **alors**
            resultat $\leftarrow$ i
        finsi
    finpour
    retourner resultat
fin

procédure tri **(E/S** t :**Tableau[**1..MAX**] d'Entier**,**E** nb :**Naturel)**
    Déclaration i : **Naturel**
debut
    pour i $\leftarrow$1 **à** nb-1 **faire**
        echanger(t[i],t[indiceDuMinimum(t,i,nb)])
    finpour

Le tri par insertion 1 / 2

Principe

On parcourt le tableau t (i variant de 2 à nb) et à chaque itération i

- on détermine l'indice j de la position de t[i] dans l'intervalle $[1..i]$
- on insère $t[i]$ à la j^{eme} place grace à un décalage

Complexité

$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$

Le tri par insertion 2 / 2

algorithme

```
procédure inserer ( E/S t : Tableau[1..MAX] d'Entier, E depart : Naturel)
    Déclaration  i : NaturelNonNul
                 val : Entier
debut
    i ← depart
    val ← t[depart]
    tant que i>1 et t[i-1]>val faire
        t[i] ← t[i-1]
        i ← i-1
    fintantque
    t[i] ← val
fin
procédure tri (E/S t :Tableau[1..MAX] d'Entier,E nb :Naturel)
    Déclaration  i,j : NaturelNonNul
                 temp : Entier
debut
    pour i ←2 à nb faire
        inserer(t,i)
    finpour
fin
```

Principes des tris récursifs

Principes

L'algorithme des tris récursifs est basé sur le principe :

- Diviser : On divise le tableau en deux
- Régner : On trie ces deux tableaux
- Combiner : On combine ces deux tableaux

Il existe deux tris récursifs

- **Le tri rapide** : "l'intelligence" du tri se trouve au niveau de la division du tableau (partitionnement)
- **Le tri par fusion** : "l'intelligence" du tri se trouve au niveau la combinaison des deux tableaux

INSA ROUEN NORMANDIE

Le tri rapide (Quick sort) 1 / 8

Principe

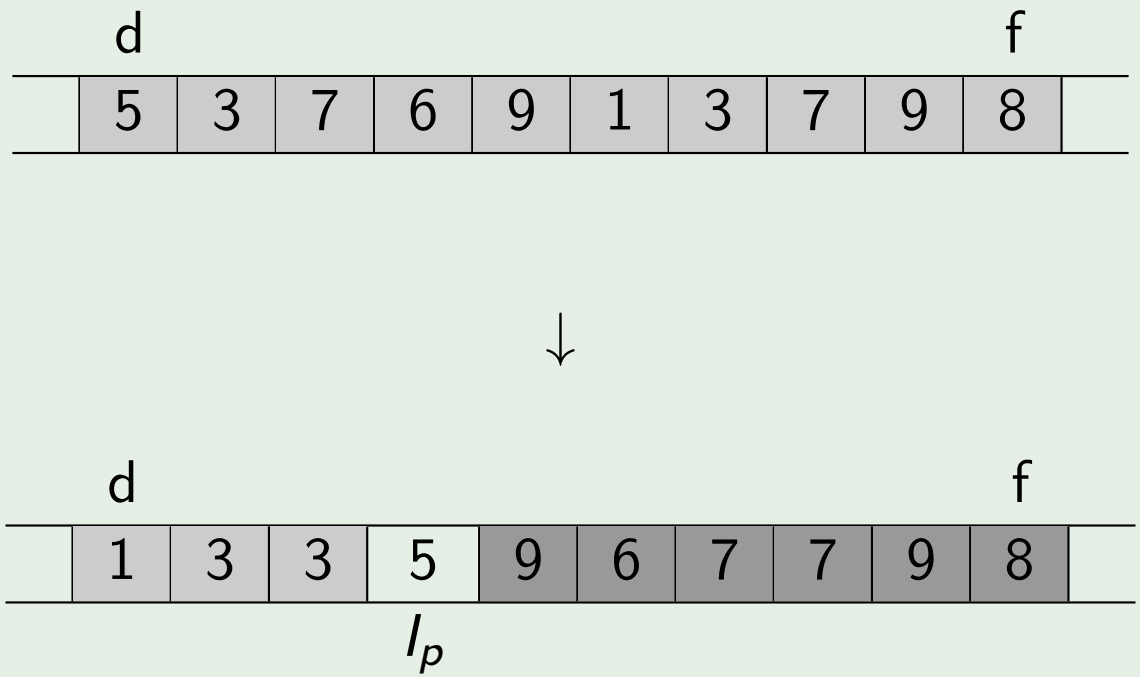
1. Partitionner le tableau afin que tous les éléments du sous-tableau gauche soient plus petits ou égaux à un élément (le pivot)
2. Trier le sous-tableau gauche et le sous-tableau droit

algorithme

```
procédure tri (E/S t :Tableau[1..MAX] d'Entier,E nb :NaturelNonNul)
debut
    triRapideRecursif(t,1,nb)
fin
procédure triRapideRecursif (E/S t :Tableau[1..MAX] d'Entier,E d,f :NaturelNonNul)
    Déclaration   indicePivot : Naturel
debut
    si d<f alors
        partionner(t,d,f,indicePivot)
        triRapideRecursif(t,d,indicePivot-1)
        triRapideRecursif(t,indicePivot+1,f)
    finsi
fin
```

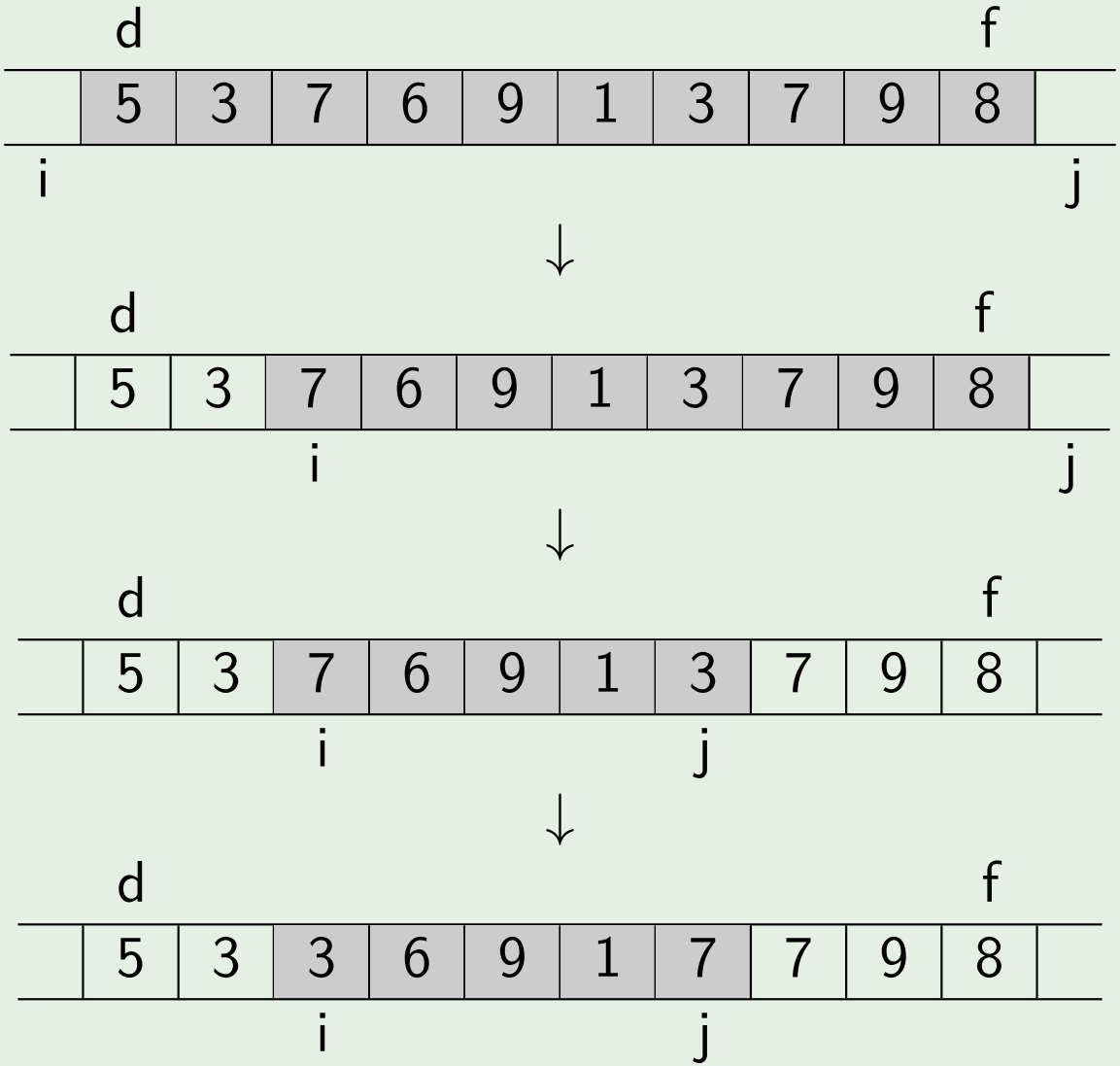
Le tri rapide (Quick sort) 2 / 8

Exemple de partitionnement (pivot = premier élément)



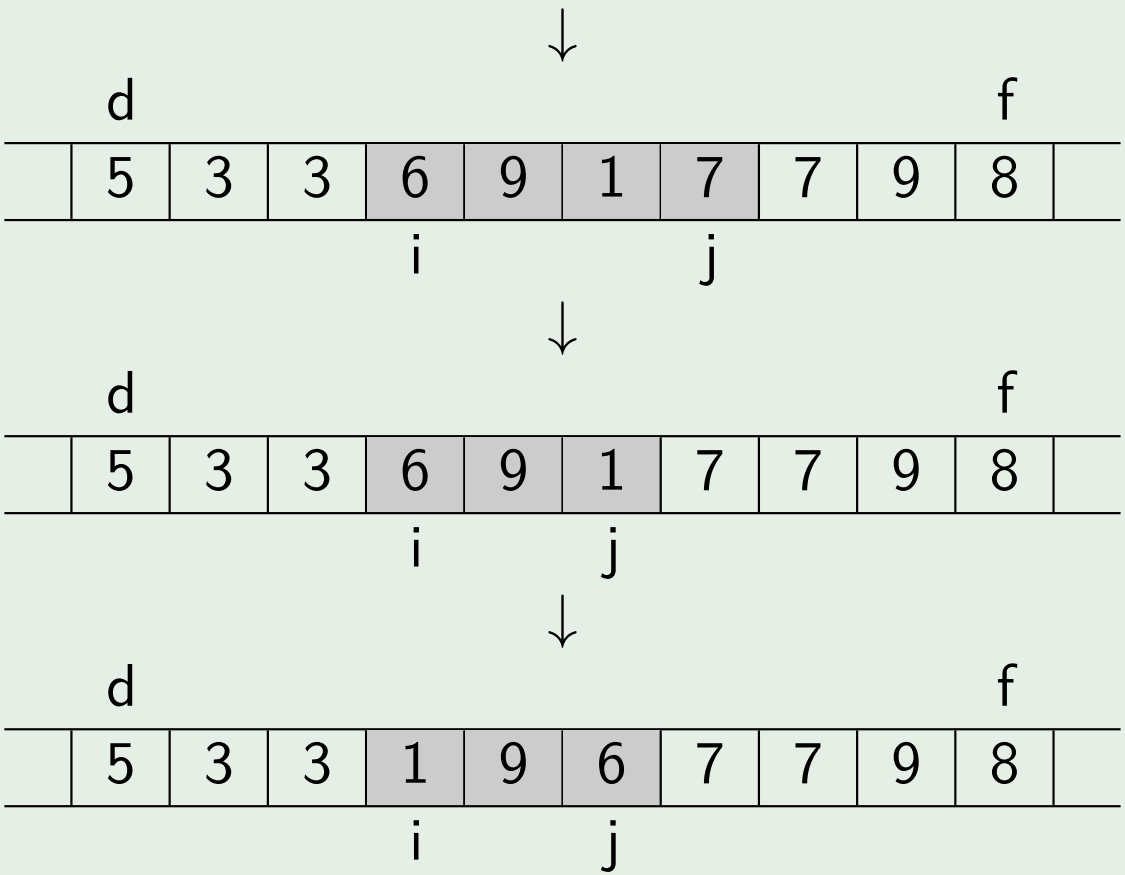
Le tri rapide (Quick sort) 3 / 8

Exemple de fonctionnement du partitionnement



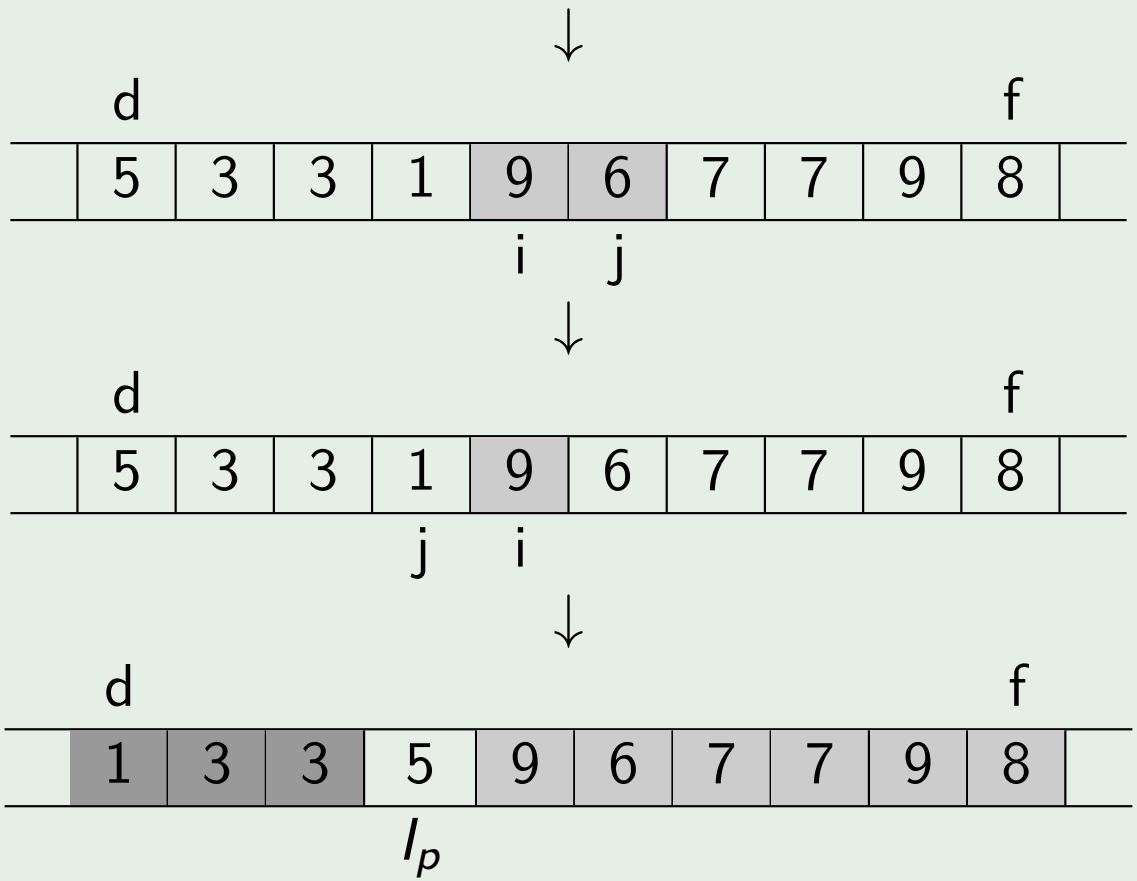
Le tri rapide (Quick sort) 4 / 8

Exemple de fonctionnement du partitionnement



Le tri rapide (Quick sort) 5 / 8

Exemple de fonctionnement du partitionnement



Le tri rapide (Quick sort) 6 / 8

procédure *partitionner* (première version)

```
procédure partitionner (E/S t : Tableau[1..MAX] d'Entier ; E debut,fin : Naturel ; S
indicePivot : Naturel)
    Déclaration  i,j : Naturel ; pivot : Entier
debut
    pivot ← t[debut]
    i ← debut
    j ← fin
    tant que i≤j faire
        tant que t[i]≤pivot et i≤j faire
            i ← i+1
        fintantque
        tant que t[j]>pivot et i≤j faire
            j ← j-1
        fintantque
        si i≤j alors
            echanger(t[i],t[j])
        finsi
    fintantque
    indicePivot ← j
    echanger(t[debut],t[j])
fin
```

Le tri rapide (Quick sort) 7 / 8

procédure *partitionner* (deuxième version)

```
procédure partitionner (E/S t : Tableau[1..MAX] d'Entier ; E debut,fin : Naturel ; S
indicePivot : Naturel)
    Déclaration  i,j,pivot : Naturel
debut
    pivot ← t[debut]
    i ← debut
    j ← fin
    tant que i≤j faire
        si t[i]≤pivot alors
            i ← i+1
        sinon
            si t[j]>pivot alors
                j ← j-1
            sinon
                echanger(t[i],t[j])
            finsi
        finsi
    fintantque
    indicePivot ← j
    echanger(t[debut],t[j])
fin
```

Le tri rapide (Quick sort) 8 / 8

Calcul de la complexité

La procédure de partionnement a une complexité en n. La complexité du tri rapide dépend donc du nombre d'appels récursifs (hauteur h de l'arbre de récursion)

- Dans le meilleur des cas, le partitionnement coupe le tableau en deux parties de même longueur (à plus ou moins 1 près)
  On a : $n = 2^h$, donc $h = log_2 n$
  Donc on a $\Omega(nlog_2 n)$
- Dans le pire des cas, le partionnement coupe le tableau en deux sous tableaux, l'un de longueur 1 et l'autre $n - 1$
  Dans ce cas $h = n$
  Et donc on a $O(n^2)$
- En moyenne on a $\Theta(nlog_2 n)$

Complexité

$\Omega(nlog_2 n)$	$\Theta(nlog_2 n)$	$O(n^2)$

INSA ROUEN NORMANDIE

Le tri par fusion 1 / 5

Principe

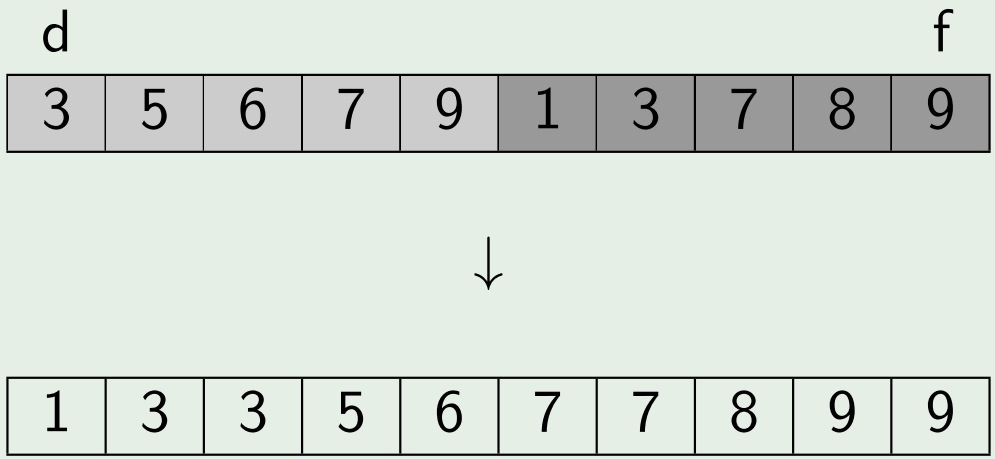
1. Diviser le tableau en deux sous-tableaux de même longueur (à plus ou moins 1 près)
2. Trier le sous-tableau gauche et le sous-tableau droit
3. Fusionner les deux sous-tableaux

algorithme

```
procédure tri (E/S t :Tableau[1..MAX] d'Entier,E nb :NaturelNonNul)
debut
    triFusionRecursif(t,1,nb)
fin
procédure triFusionRecursif (E/S t :Tableau[1..MAX] d'Entier,E d,f :NaturelNonNul)
debut
    si d<f alors
        triFusionRecursif(t,d,(d+f) div 2)
        triFusionRecursif(t,((d+f) div 2)+1,f)
        fusionnner(t,d,(d+f) div 2,f)
    finsi
fin
```

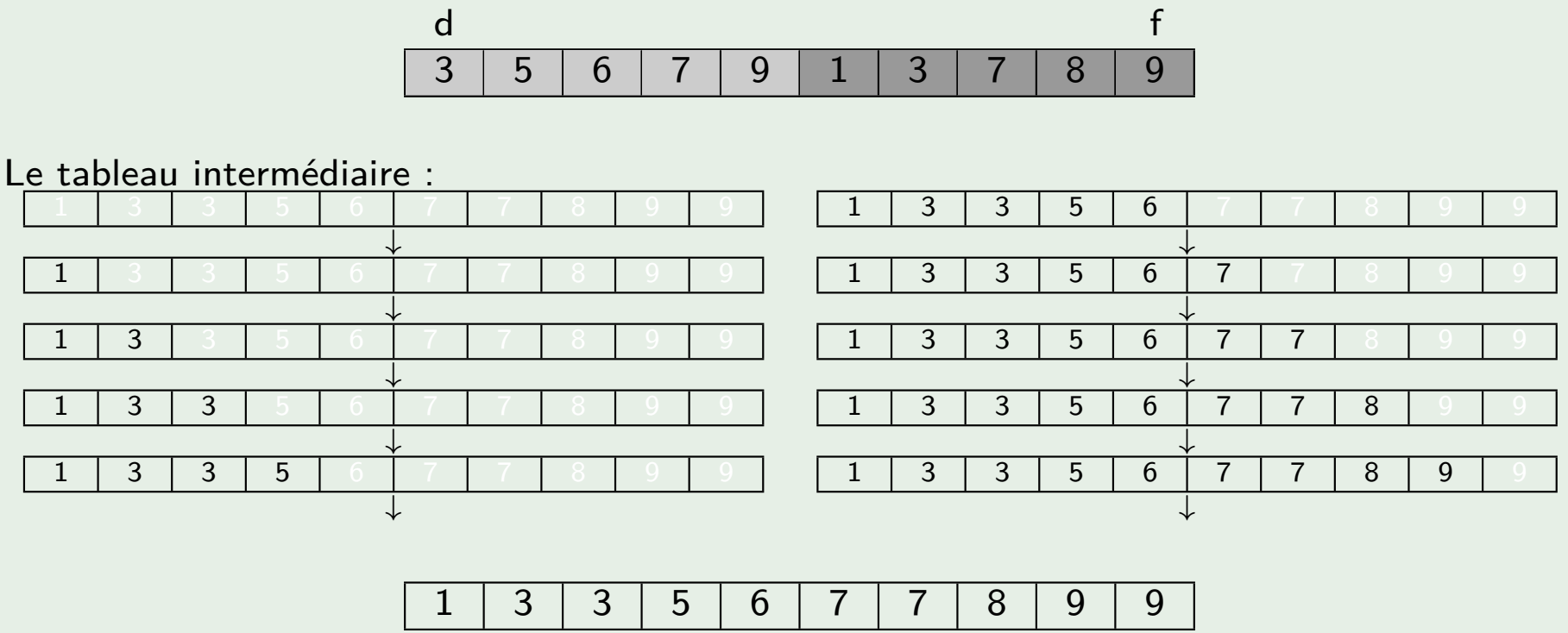
Le tri par fusion 2 / 5

Fonctionnement de *fusionner*



Le tri par fusion 3 / 5

Exemple de fonctionnement de *fusionner*



Le tri par fusion 4 / 5

Procédure *fusionner*

```
procédure fusionner (E/S t : Tableau[1..MAX] d'Entier ; E debut,milieu,fin : NaturelNonNul)
    Déclaration   i,j,k : NaturelNonNul,
                  temp : Tableau[1..MAX] d'Entier
debut
    i ← debut
    j ← milieu+1
    pour k ←1 à fin-debut+1 faire
        si i≤milieu et j≤fin alors
            si t[i]≤t[j] alors
                temp[k] ← t[i]
                i ← i+1
            sinon
                temp[k] ← t[j]
                j ← j+1
            finsi
        sinon
            si i≤milieu alors
                temp[k] ← t[i]
                i ← i+1
            sinon
                temp[k] ← t[j]
                j ← j+1
            finsi
        finsi
    finpour
    pour k ←1 à fin-debut+1 faire
        t[debut+k-1] ← temp[k]
    finpour
fin
```

Le tri par fusion 5 / 5

Calcul de la complexité

Soit h la hauteur de l'arbre de récursion
Ici on a toujours $n = 2^h$, donc $h = log_2 n$
Donc en temps on a $\Omega(nlog_2 n)$, $O(nlog_2 n)$ et $\Theta(nlog_2 n)$
Mais on a besoin d'un tableau intermédiaire pour fusionner

Complexité

$\Omega(nlog_2 n)$	$\Theta(nlog_2 n)$	$O(nlog_2 n)$

Conclusion

Il existe plusieurs algorithmes de tri que l'on peut classer suivant :

- les méthodes utilisées (itératifs ou récursifs)
- les performances

Ce cours ne présente pas toutes les méthodes de tri, entre autres :

- le *shellsort*
- le *heapsort* (ou tri par tas)
- le *radixsort*
- etc.

INSA ROUEN NORMANDIE

Procédures ou fonctions comme paramètre 1 / 3

Objectifs

- Généraliser des algorithmiques (par exemple les tris)
- Séparer logique métier et IHM (*callback*)

- Créer un type procédure ou fonction
- Déclarer des fonctions ou procédures comme paramètres formels de procédure (passage de paramètre en entrée) ou fonction
- Utiliser des identifiant de procédures ou fonctions comme paramètres effectifs

INSA ROUEN NORMANDIE

Procédures ou fonctions comme paramètre 2 / 3

Exemple

Type ComparerEntier = **fonction**(a,b : **Entier**) : **Booleen**

procédure triABulle **(E/S** t :**Tableau[**1..MAX**] d'Entier**, **E** nb :**Naturel**, dansLOrdre : ComparerEntier**)**

```
    Déclaration   estTrie : Booleen
                  i : Naturel
debut
    repeter
        estTrie ← VRAI
        pour i ←1 à nb-1 faire
            si non dansLOrdre(t[i],t[i+1]) alors
                echanger(t[i],t[i+1])
                estTrie ← FAUX
            finsi
        finpour
    jusqu'a ce que estTrie
fin
```

Procédures ou fonctions comme paramètre 3 / 3

Exemple

fonction plusPetit **(**a,b : **Entier) : Booleen**
debut
    retourner a$\leq$b
fin
fonction plusGrand **(**a,b : **Entier) : Booleen**
debut
    retourner a$\geq$b
fin

Conclusion

Important

Avant de se mettre devant la machine, **on doit écrire les algorithmes !!!**

Méthode

1. Analyser le problème
2. Définir un énoncé
3. Faire une analyse descendante
4. À chaque niveau de l'analyse définir les signatures des procédures et/ou fonctions
5. Écrire les algorithmes de chaque procédure et fonction de niveau n de l'analyse en considérant que l'on possède de nouvelles instructions qui résolvent les problèmes du niveau $n-1$