Exercices

1. Ecrire une fonction récursive qui permet de calculer la suite suivante:

$$U_1 = 1$$
$$U_n = U_{n-1} + N \quad \text{tel que } N \geq 1$$

2. Ecrire une fonction booléenne rend une valeur vrai si un nombre "b" est diviseur d'un nombre "a" ou faux dans le cas contraire.

3. Ecrire une fonction récursive permettant de calculer la puissance pour x réel et n entier en utilisant la définition récursive suivante:

$X^n = 1$ pour $n = 0$,

$X^n = (X^{n/2})^2$ pour n pair,

$X^n = X^{n-1} . X$ pour n impair.

Exercices : Solution

1.

```
Fonction Suite-R (N : Entier) : entier
 Debut
   Si  N = 1 Alors
            Retourne(1)
    Sinon  retourne(Suite-R (N-1) + N)
     FinSi
Fin
```

2.

```
 Fonction divisible (a, b : entier) : booleen
     Debut
        Si a ≤ b alors
                 Retourne(a = b)
       Sinon
                  retourne(divisible (a –b, b) )
        FinSi
     Fin
```

Exercices : Solution

3.

```
Fonction Cal (X : Reel, n : entier) : entier
Debut
    Si  n =0   Alors
            retourne(1)
     Sinon
          Si n Mod 2 = 0 Alors
     Retourne(Cal (Cal (X, n Div 2), 2)  )
          Sinon
                    retourne(Cal (X, n -1) * X )
          FinSi
      Finsi
 Fin
```

Analyse descendante

- Pourquoi cet analyse

  Il est souvent difficile d'écrire la version définitive du premier coup, on risque de perdre dans des détails et de ne pas voir l'essentiel.

  On procède par étapes en définissant des actions très générales: cette méthode est appelée méthode par raffinages successifs ou l'analyse descendante.

- L'analyse descendante est un mode de pensée visant à construire des algorithmes en partant d'un niveau très général et en détaillant peu à peu chaque traitement, jusqu'à arriver au niveau de description le plus bas.

- Diviser un problème en sous problèmes pour régner, construire des modules ou des sous programmes.

Analyse descendante

L'analyse descendante consiste en trois étapes où se mêlent rigueur et savoir-faire:

1. Diviser le problème en sous problèmes. C'est une partie importante qui conditionne le reste et qui n'est basée que sur l'expérience.
2. Résoudre chacun des sous problèmes:

- Soit en appliquant à nouveau la première étape (Diviser les sous problèmes en à résoudre en sous sous problèmes.
- Soit en résolvant directement le sous problème qui est un problème simple et connu.

3. Combiner les résultats des sous problèmes. C'est la partie la plus rigoureuse dans laquelle on doit faire attention à ce que donnent les sous résultats et à ce qu'il est nécessaire de faire pour obtenir le résultat final.

Analyse descendante

Etude d'un exemple 1:

Le problème: Ecrire un programme qui lit une suite d'entiers et détermine si chacun d'eux est pair et qui , dans le cas contraire, détermine si l'entier est premier.

Première étape: mettre en place la structure générale de la solution (niveau 1) sans s'occuper des problèmes « comment déterminer si N est paire et N est premier ».

```
// Niveau 1: Programme qui lit et étudie une liste d'entiers
Début
  Lire  un entier N
   Tant que N est différent de 0 faire
     Déterminer si N est pair et éventuellement s'il est premier
      Lire l'entier N suivant
    Fin Tq
Fin
```

Analyse descendante

Deuxième étape: il s'agit de raffiner un peu plus le sous problème càd Déterminer si N est pair et éventuellement s'il est premier

Troisième étape: raffiner les deux modules N est pair et N est premier

```
// Niveau 2: Calculer s'il est pair et premier
Début
  si N est pair alors
       Ecrire(« N pair »)
   sinon
       Ecrire(«  N est impair »)
       calculer et afficher si N est premier
    fin
Fin
```

```
// Niveau 3: Si N est pair
si (N mod 2)=0 alors
```

```
// Niveau 3: calculer et afficher N est premier
Div <- 2
Tant que ((N mod div) !=0 et (div*div <= à N) faire
   incrémenter div
Fin tq
 si N mod div!= 0 alors
      Ecrire (« N est premier »)
 sinon
   Ecrire (« N n'est pas premier »)
```

Analyse descendante

```
// Dernière étape combiner les résultats des sous programmes
Var N:entier
Début
  Lire(N)
   Tant que N !=0 faire
      si (N mod 2)=0 alors
        Ecrire(« N pair »)
       sinon
         Ecrire(«  N est impair »)
    Div <- 2
         Tant que ((N mod div) !=0 et (div*div <= à N) faire
           div<-div+1
         Fin tq
          si N mod div!= 0 alors Ecrire (« N est premier »)
          sinon Ecrire (« N n'est pas premier »)
          Fin si
          N<-N+1
       fin si
    Fin Tq
Fin
```

Analyse descendante

Etude d'un exemple 2:

Le problème: Ecrire un algorithme qui demande à l'utilisateur un entier et qui indique si cet entier est un nombre de Armstrong.

Décomposition de premier niveau :

1. Saisir un entier *nb*
2. Décomposer *nb*
3. Conclure s'il s'agit d'un nombre de Armstrong

Décomposition de second niveau. Les étapes 2 et 3 peuvent se décomposer en sous étapes plus simples :

2.1. Extraire le chiffre des centaines *c*
2.2. Extraire le chiffre des dizaines *d*
2.3. Extraire le chiffre des unités *u*

3.1. Calculer $s = c^3 + d^3 + u^3$
3.2. Comparer *s* et *nb*

Analyse descendante

Etude d'un exemple 2:

Le problème: Ecrire un algorithme qui demande à l'utilisateur un entier et qui indique si cet entier est un nombre de Armstrong.

Suite à cette analyse descendante, l'algorithme vient aisément.

```
ALGORITHME Armstrong
VARIABLES nb, c, d, u, s, temp : entiers
DEBUT
        saisir nb

        c ← nb div 100

        temp ← nb mod 100
        d ← temp div 10

        u ← temp mod 10

        s ← c³ + d³ + u³

        SI s = nb ALORS
                afficher nb, "est un nombre de Armstrong"
        SINON
                afficher nb, "n'est pas un nombre de Armstrong"
        FINSI
FIN
```

Les algorithmes de tris

- Le tri consiste à ordonner les éléments du tableau dans l'ordre croissant ou décroissant.
- Un tableau T est dit « trié en ordre croissant » si tous les éléments consécutifs du tableau vérifient: T[i-1]<= T[i]
- D'où la définition:
  - Un tableau vide (n=0) est ordonné (trié),
  - Un tableau contenant un seul élément n=1 est ordonné,
  - Un Tableau T[1…..n], n>1 est ordonné si i appartient à [2….n], T[i-1]<= T[i]
- Il existe plusieurs algorithmes connus pour trier les éléments 'un tableau
  - Le tri par sélection et permutation(tri par minimun successif)
  - Le tri par sélection
  - Le trie à bulles
  - Le trie par insertion

Tri par sélection

- Principe:

Utiliser un vecteur VT (vecteur trié) comme vecteur résultat.

Celui-ci contiendra les éléments du vecteur initial dans l'ordre croissant.

1. Chercher le plus grand élément dans le vecteur initial V
2. Sélectionner le plus petit élément dans V
3. Le mettre dans son ordre dans le vecteur VT Le remplacer par le plus grand élément dans le vecteur initial (pour qu'il ne sera plus le minimun)
4. Si le nombre d'éléments dans le vecteur résultat n'est pas identique à celui dans le vecteur initial retourner à l'étape 1 sinon on s'arrete.

Les algorithmes de tris itératifs: Tri par sélection

- Algorithme:

Tri_selection(A,n)

```
Const Bi=1
      Bs=10
Var V,VT: tableau[Bi…Bs] réel
N,i,j,indmin: entier
min, max: réel
Début
// remplissage u vecteur V
//Recherche du maximum
max<-V[1]
pour i := 2 à N faire
   si max<= V[i] alors
      max<-V[i]
   Fin si
Fin pour
```

```
pour i:=1 à N-1 faire
//Recherche du minimum
min<-V[1]
Indmin<-1
pour i:=2 à N faire
      si min > V[j] alors
       min<-V[j]
       indmin:=j
      fsi
Fpour
// Mettre le minimum trouvé à sa place dans
le vecteur VT
VT[i]:= min[indmin]
min[indmin]:= max
Fin pour
VT[N] := max
fin
```

Tri par minimum successif

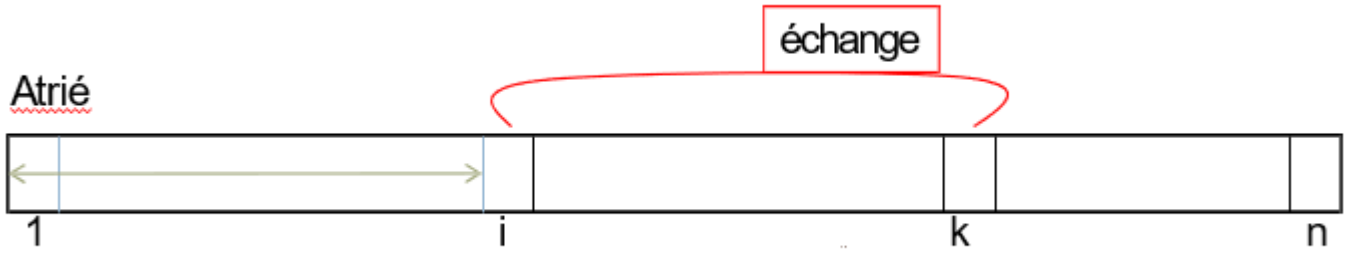
Algorithme:

pour i=1 à n-1 faire

-chercher le 1ème minimum, A_k , de $\{A_{i+1},\ldots,A_n\}$ (K est l'indice de $\min\{A_{i+1},\ldots,A_n\}$ dans le tableau A)

- échanger A_i et A_k

L'algorithme fonctionne selon le schéma suivant:



Tri par minimum successif

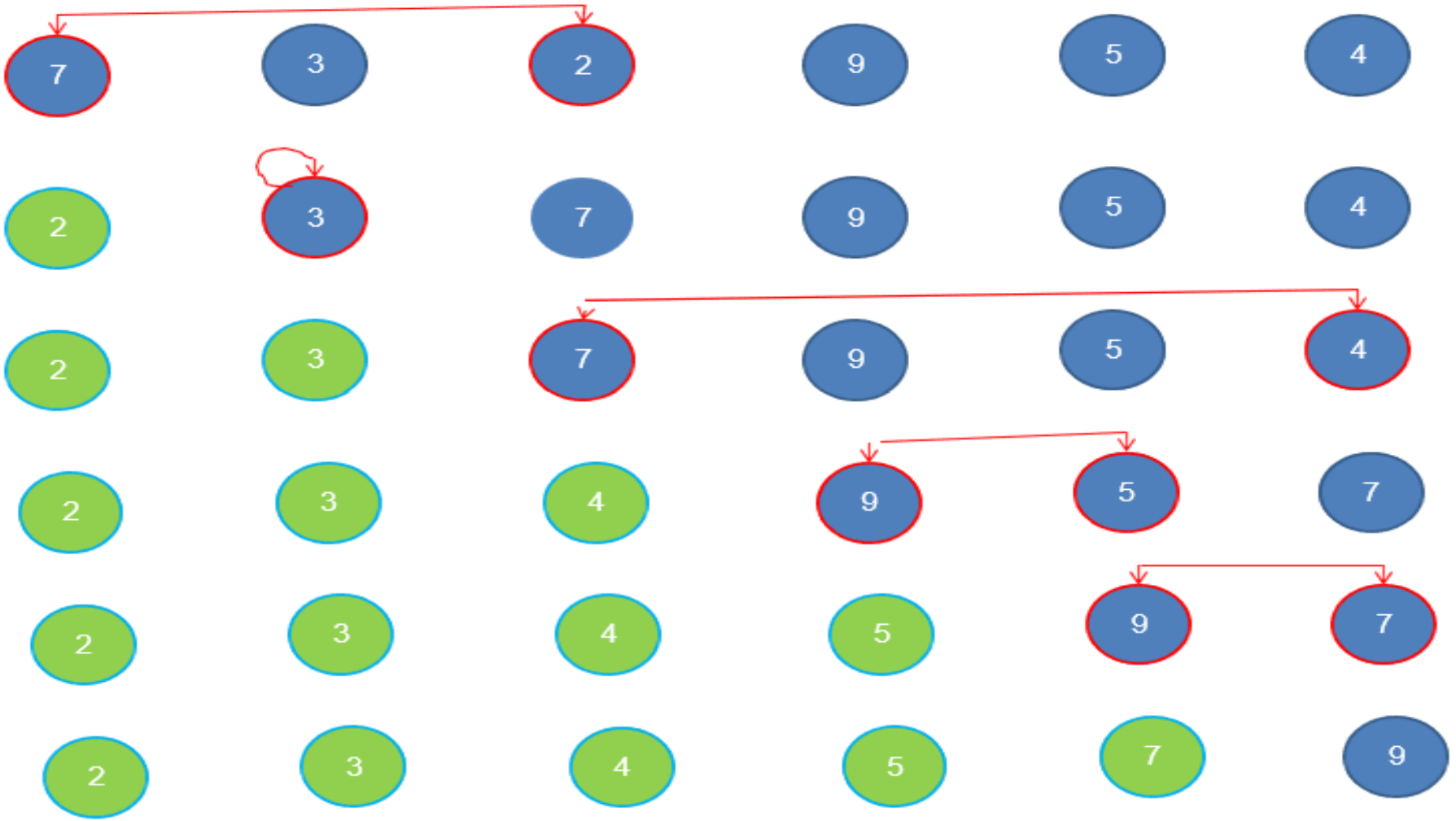
- Principe:

Le tri par minimum successif nommé tri par sélection et permutation: pour une place donné, on sélectionne l'élément qui doit y être positionné.

Si on parcourt le tableau de gauche à droite, on positionne à chaque fois le plus petit élément qui se trouve dans le sous tableau droite.

Généralement : pour trier le sous tableau t[i… nbelements] il suffit de positionner au rang i le plus petit élément de ce sous tableau et de trier le sous tableau t[i+1…... nbelements]

Tri par sélection: minimum successif



Tri par minimum successif

- Algorithme:

Tri_selection(A,n)

début

pour *i := 1* ***à*** *n-1* ***faire***

//Recherche de min{A_i, …, A_n} = A_k

k := i;

pour *j:=i+1* ***à*** *n* ***faire***

si *A[j] < A[k]* ***alors***

k:=j;

fsi;

fpour

// échange de A_k et A_i

temp := A[k];

A[k] := A[i];

A[i] := temp;

fpour

fin