Complexité des algorithmes : nombres_instructions élémentaires[1]

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

[1] Version préliminaire du cours. Tout retour sur la forme comme sur le fond est le bienvenu.

Objectifs

But d'une analyse de complexité "théorique"

- Pour un même problème, comparer plusieurs algorithmes avant de choisir celui qui sera implémenté
- identifier les cas favorables, moyens et surtout défavorables
- Analyse asymptotique :
  - Déterminer la faisabilité d'un algorithme en estimant une "borne" supérieur du temps qui sera nécessaire à un algorithme.

Objectifs

But d'une analyse de complexité "théorique"

- Pour un même problème, comparer plusieurs algorithmes avant de choisir celui qui sera implémenté
- identifier les cas favorables, moyens et surtout défavorables
- Analyse asymptotique :
  - Déterminer la faisabilité d'un algorithme en estimant une "borne" supérieur du temps qui sera nécessaire à un algorithme.
  - Cette estimation consiste à établir une relation le temps d'exécution et la taille des données en entrée.

Objectifs

But d'une analyse de complexité "théorique"

- Pour un même problème, comparer plusieurs algorithmes avant de choisir celui qui sera implémenté
- identifier les cas favorables, moyens et surtout défavorables
- Analyse asymptotique :
  - Déterminer la faisabilité d'un algorithme en estimant une "borne" supérieur du temps qui sera nécessaire à un algorithme.
  - Cette estimation consiste à établir une relation le temps d'exécution et la taille des données en entrée.
  - Cette estimation doit-être indépendante du contexte (langage programmation, machine etc.).

Calcul de la complexité en trois étapes

Etape 1

Définir l'unité de mesure : la notion d'instructions élémentaires

Calcul de la complexité en trois étapes

Etape 1

Définir l'unité de mesure : la notion d'instructions élémentaires

Etape 2

Calculer le nombre d'instructions élémentaires $T(n)$ en fonction de la taille des données n

Calcul de la complexité en trois étapes

Etape 1

Définir l'unité de mesure : la notion d'instructions élémentaires

Etape 2

Calculer le nombre d'instructions élémentaires $T(n)$ en fonction de la taille des données n

Etape 3

Calculer une estimation d'une borne supérieure $f(n)$ de $T(n)$.

A chaque mesure
son unité

Unité de mesure

Que peut-être une unité de mesure ?

- Une unité de mesure représente une instruction élémentaire.
- La question est qu'est-ce qui peut-être considérée comme instruction élémentaire :
  - Une addition?
  - une soustraction ?
  - Une multiplication ?
- L'idéal : une instruction *ie* est considérée comme élémentaire si le temps d'exécution de toute instruction *i* est multiple du temps d'exécution de *ie* (indépendamment de tout contexte machines, compilateurs etc.)
- En pratique, il est quasiment impossible de définir une telle instruction élémentaire.

Unité de mesure

Que peut-être une unité de mesure ?

- On peut-être tenté de considérer que ”le produit de deux entiers” par exemple est plus coûteux que ”l’addition de deux entiers”.
- Le problème est qu’une définition trop précise de l’instruction élémentaire rendrait le calcul du nombre d’instructions élémentaires, exécutés par un algorithme, fastidieux.

Unité de mesure

- En pratique, on considérera qu'il n'y a pas de différence entre les 3 opérations suivantes (avec a,b,c des entiers) :
  - a =b;
  - a=b*c;
  - a=a+b*c;
- On verra dans la suite, lorsque l'on s'intéresse à l'analyse asymptotique des algorithmes, la différence entre les trois opération est négligeable.

Unité de mesure

Se débarasser du contexte

Une unité de mesure représente une instruction élémentaire.
Une unité de mesure peut-être :

- Une addition
- une soustraction
- Une multiplication, une division, Mod (%), Div
- Une opération arithmétique simple (sans appel de fonctions)
- Une comparaison, les opérations booléennes (et,ou,non)
- une affectation
- Des opérations de lectures et écritures simples

Calculer le nombre d'instructions élémentaires

Notations

- Le temps de calcul ou bien le nombre d'instructions nécessaires pour l'exécution d'un algorithme sera noté : *T, f, g, h, etc.*.
- Ces temps de calculs dépendent des paramètres notés n,m,p,x, a, b, etc
- Certaines fonctions admettent un seul paramètre d'autres admettent plusieurs paramètres, par exemple :
  - On peut noter $T(a)$ le temps nécessaire pour le calcul de la fonction "Estpremiernaïf". a est le nombre à tester la primalité.
  - On peut noter $T(N, M)$ le temps nécessaire pour le calcul de la variante de l'algorithme "jeu pommes, pêches, poires, abricots". N étant le nombre de joueurs et M représente le nombre éliminatoire.

Calculer le nombre d'instructions élémentaires

Remarque

Dans le reste de ce cours, les fonctions utilisées donnent toujours des valeurs positives quelque soit la valeur de l'entrée (des paramètres), puisqu'elles représentent des temps de calculs.

Calculer le nombre d'instructions élémentaires

Séquence

La première étape est d'identifier les séquences dans un algorithme.
Si votre algorithme est composée des séquences :

$$I_1;$$
$$I_2;$$
$$.$$
$$.$$
$$I_n;$$

Alors :

$$T(n) = T_{I_1}(n) + T_{I_2}(n) + ... + T_{I_n}(n),$$

où:

- $T(n)$ représente le nombre total d'instructions
- $T_{I_j}(n)$ représente le nombre d'instructions dans I_j.

Exemple : séquence

Dans la fonction ”Somme”, nous pouvons identifier trois séquences I1, I2 et I3.

```
double Somme(double A[], int N) {
   double s;   int i;

Sequence I1:       s=0;

Sequence I2:
   for (i=0; i< N; i++)
      s = s + A[i];

Sequence I3:   return s;}
```

Exemple : séquence

```
double Somme(double A[], int N) {
   double s;   int i;

Sequence I1:       s=0;

Sequence I2:
   for (i=0; i< N; i++)
      s = s + A[i];

Sequence I3:   return s;}
```

Séquence

La fonction *Somme* contient 3 séquences. Le coût de la fonction *Somme* est de :

$$T(Somme) = T(I1) + T(I2) + T(I3).$$

Les séquences *I*1 et *I*3 contiennent chacune une seule instruction élémentaires. Cependant, I2 est une séquence complexe. De ce fait :

$$T(Somme) = 2 + T(I2).$$

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel simple de la forme :

Si Condition
Alors *I*;

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel simple de la forme :

Si Condition
Alors I;

Réponse : cas favorable :

$$T(n) = T_{condition}(n)$$

- $T(n)$ représente le nombre total d'instructions
- $T_{condition}(n)$ représente le nombre d'instructions nécessaire pour tester la condition (qui peut-être 1 s'il s'agit par exemple d'une simple comparaison entre deux expressions arithmétiques).
- Le cas favorable correspond à la situation où la condition est fausse (et de ce fait la suite d'instructions I n'est pas effectuée).

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel simple de la forme :

Si Condition
Alors *I*;

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel simple de la forme :

Si Condition
Alors I;

Réponse

Cas défavorable :

$$T(n) = T_{condition}(n) + T_I(n),$$

où:

- $T(n)$ représente le nombre total d'instructions
- $T_I(n)$ représente le nombre d'instructions dans I.

Exemple : Conditionnel

```
//Condition     : C
    if (i == 0)
```

```
//Corps de la condition : I
    {
      debut = (joueur *)malloc(sizeof(joueur));
      p = debut;
    }
```

- Le coût du conditionnel est égal au coût de la condition C plus le coût du corps du conditionnel I.
- La condition est composée d'une instruction élémentaire, donc : T(C)=1
- Le corps du conditionnel est composé de deux instructions élémentaires, donc : T(I)=2.
- Le coût final du conditionnel est : T (conditionnel)=3.

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel de la forme :

Si Condition
Alors I_1;
Sinon I_2;

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel de la forme :

Si Condition
Alors I_1;
Sinon I_2;

Réponse

- **Cas favorable**

$$T(n) = T_{condition}(n) + min(T_{I_1}(n), T_{I_2}(n)).$$

Calculer le nombre d'instructions élémentaires

Conditionnel

Comment définir le nombre d'instructions associé à un conditionnel de la forme :

Si Condition
Alors I_1;
Sinon I_2;

Réponse

- **Cas favorable**

$$T(n) = T_{condition}(n) + min(T_{I_1}(n), T_{I_2}(n)).$$

- **Cas défavorable**

$$T(n) = T_{condition}(n) + max(T_{I_1}(n), T_{I_2}(n)).$$

Exemple : Conditionnel

```
//Condition     : C
    if (i == 0)
```

```
//Corps si la conditions est vraie : I1
    {
      debut = (joueur *)malloc(sizeof(joueur));
      p = debut;
    }
```

```
//Corps si la conditioncC est fausse : I2
    else {
      p->suivant = (joueur *) malloc(sizeof(joueur));
      p = p->suivant;    }
```

- Nous avons vu que : T(C)=1 et T(I1)=2
- Le corps du conditionnel I2, si la condition est fausse, est composé de deux instructions élémentaires, donc : T(I2)=2.
- Le coût final du conditionnel est : T (conditionnel)=1+max(2,2)=3.

Calculer le nombre d'instructions élémentaires

Boucle Pour

Comment définir le nombre d'instructions associé à des boucles "Pour" de la forme :

```
Pour j=Début jusqu'à Fin
Faire
    I(j);
Finfaire
```

Remarques

- I(j) : est la suite d'instructions à exécuter en fonction de j.

Calculer le nombre d'instructions élémentaires

Calculer le nombre d'instructions élémentaires

Réponse

- Pour calculer le nombre d'instruction, on notera $T_j(I)$ le nombre d'instructions dans I pour la valeur j.
- L'entête de la boucle "Pour" contient :
  - Une initialisation j=Début (réalisée qu'une seule fois)
  - Chaque étape de la boucle :
    - Une instruction pour incrémenter j
    - Une instruction pour comparer j à la valeur d'arrêt Fin
- Au final, le nombre d'instructions associées à la boucle est :

$$T(n) = 1 + \sum_{j=1,..,n} (2 + T_j(I)).$$

Exemple : tri par sélection

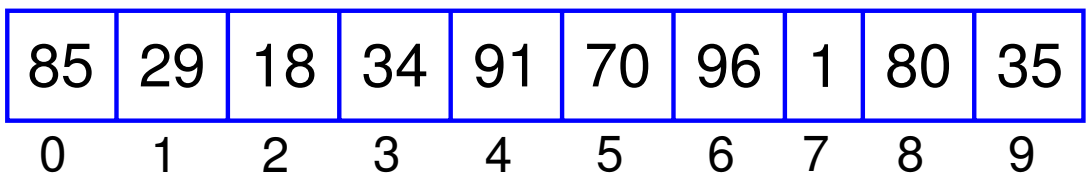
Tri par sélection

- On se donne un tableau T d'entiers, de Taille n, à trier par ordre croissant.
- L'idée est de parcourir le tableau de 0 à $n-2$, et à chaque étape i :
  - On cherche $T[j]$ la valeur minimal entre $T[i]$,..., $T[n-1]$
  - On échange $T[j]$ et $T[i]$.

Exemple : Tri par sélection

Exemple : tri par sélection

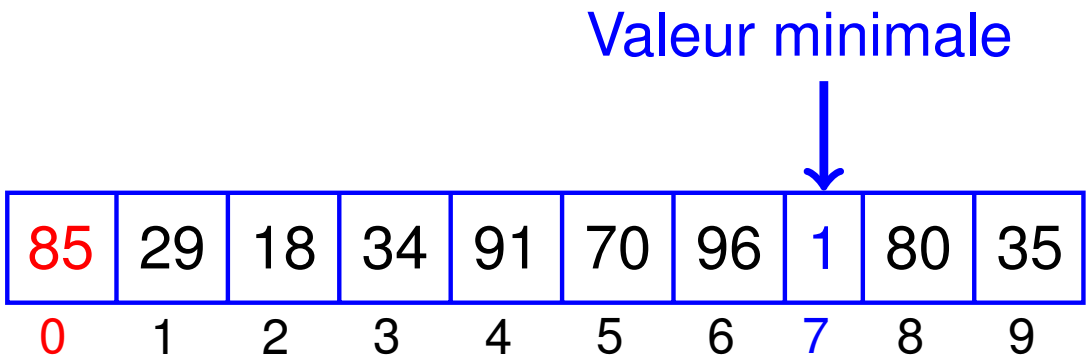
Voici le tableau initial à trier :



Exemple : tri par sélection

Traitement de la case 0 :

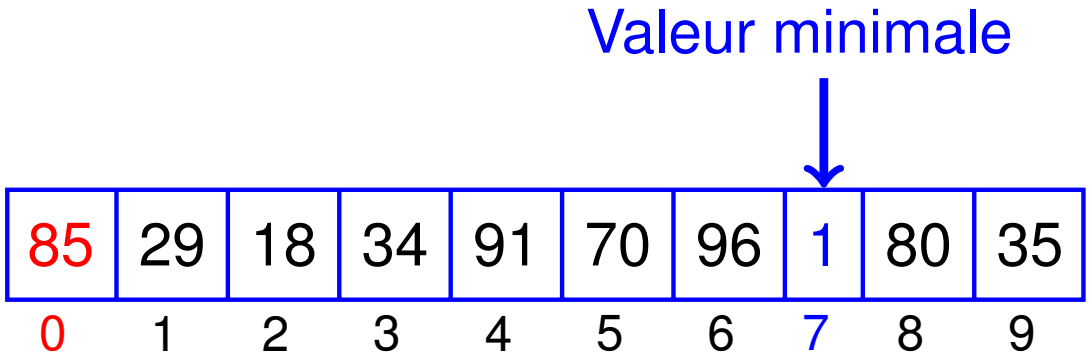
La valeur minimale se trouve à la case 7 .



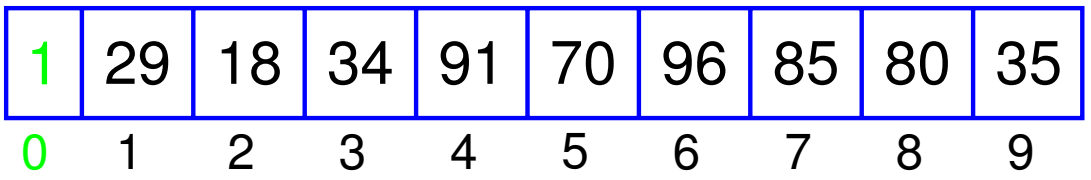
Exemple : tri par sélection

Traitement de la case 0 :

La valeur minimale se trouve à la case 7 .



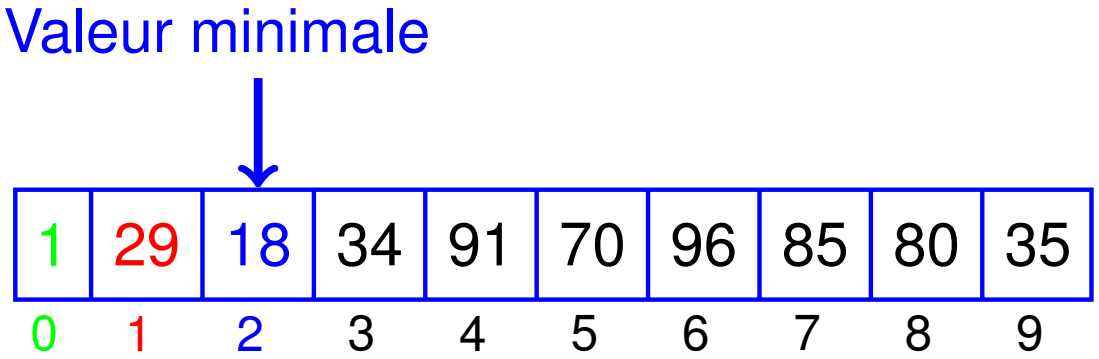
L'échange entre Tab[0]=85 et Tab[7]=1 donne :



Exemple : tri par sélection

Traitement de la case 1 :

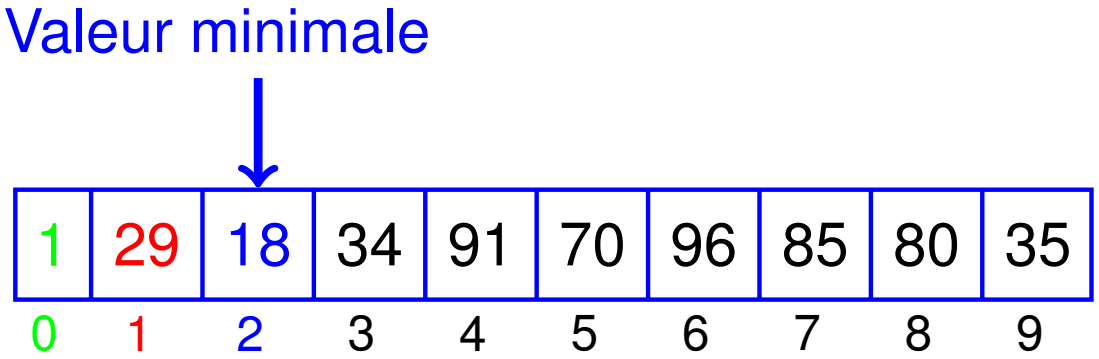
La valeur minimale se trouve à la case 2 .



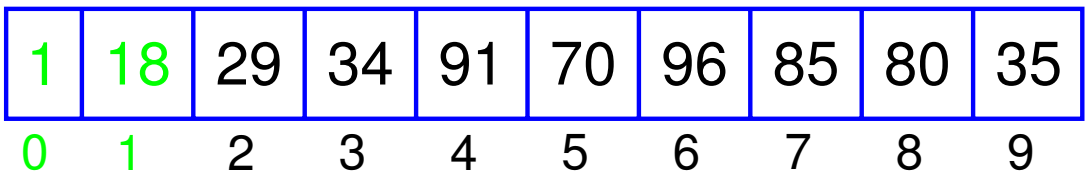
Exemple : tri par sélection

Traitement de la case 1 :

La valeur minimale se trouve à la case 2 .

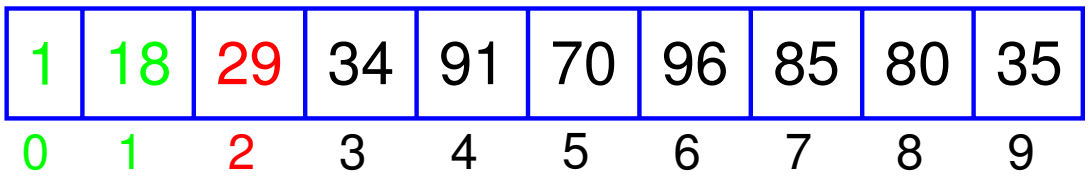


L'échange entre Tab[1]=29 et Tab[2]=18 donne :



Exemple : tri par sélection

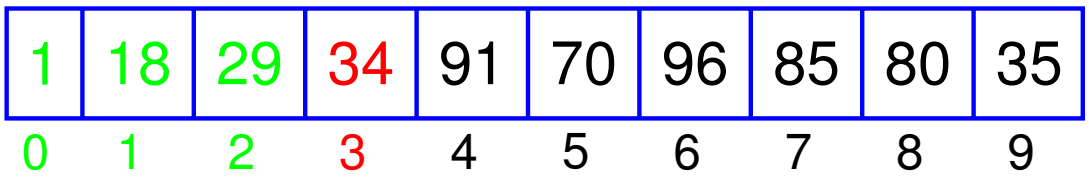
Traitement de la case 2 :



- L'élément T[2] est à sa bonne place
- Aucun échange n'est nécessaire

Exemple : tri par sélection

Traitement de la case 3 :

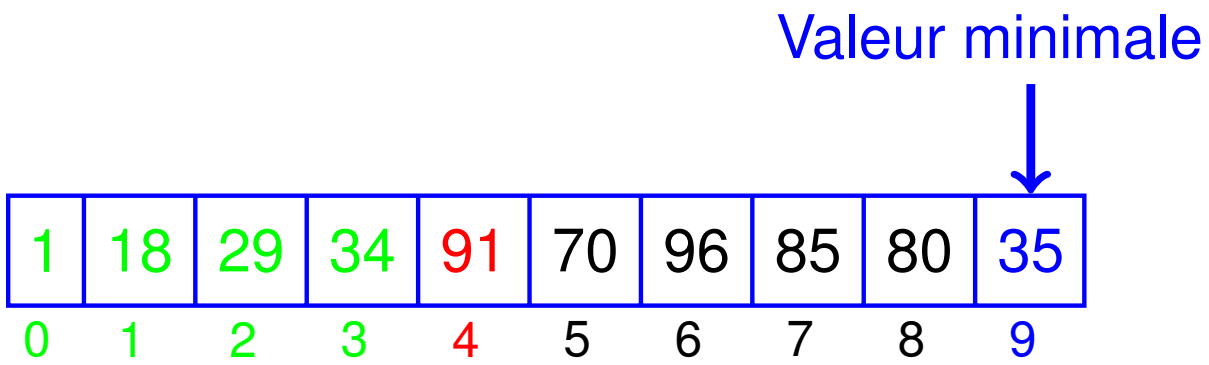


- L'élément T[3] est à sa bonne place
- Aucun échange n'est nécessaire

Exemple : tri par sélection

Traitement de la case 4 :

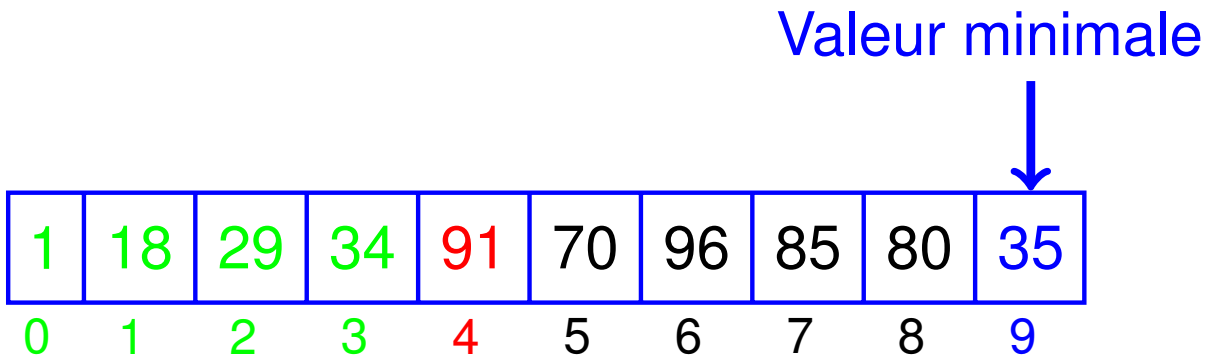
La valeur minimale se trouve à la case 9 .



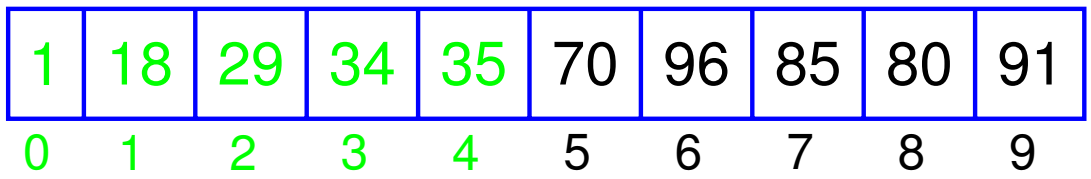
Exemple : tri par sélection

Traitement de la case 4 :

La valeur minimale se trouve à la case 9 .

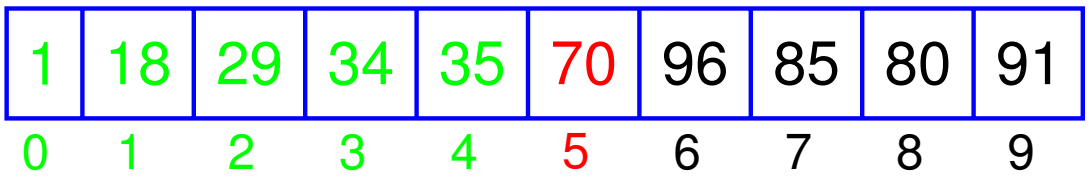


L'échange entre Tab[4]=91 et Tab[9]=35 donne :



Exemple : tri par sélection

Traitement de la case 5 :

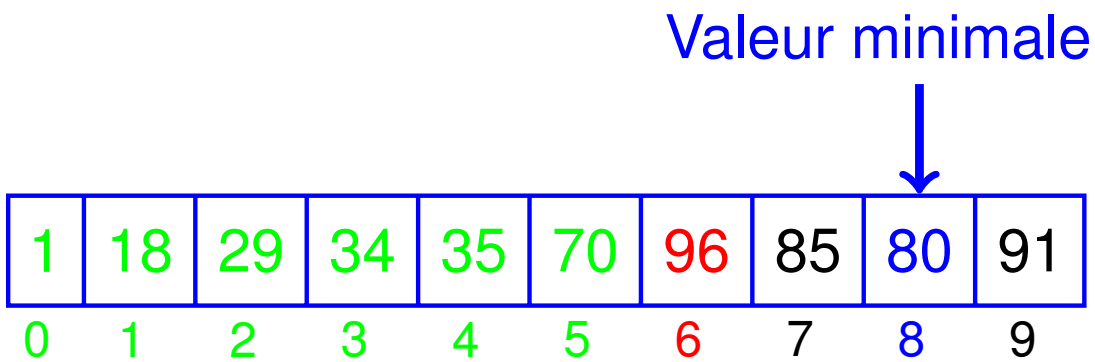


- L'élément T[5] est à sa bonne place
- Aucun échange n'est nécessaire

Exemple : tri par sélection

Traitement de la case 6 :

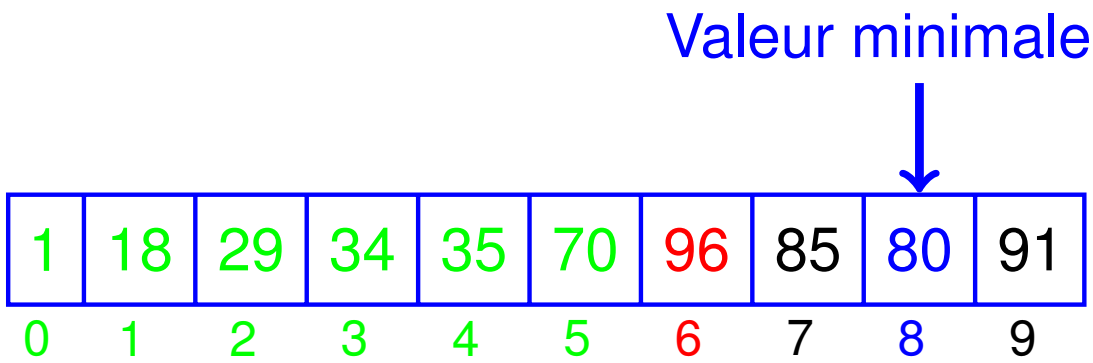
La valeur minimale se trouve à la case 8 .



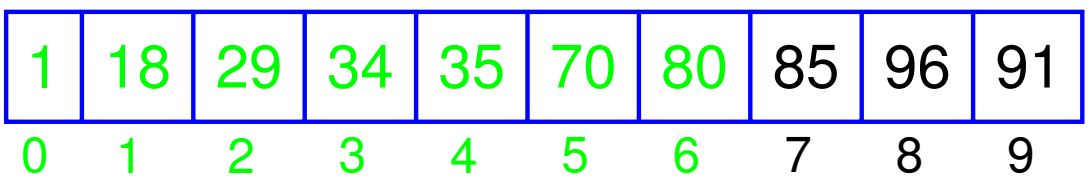
Exemple : tri par sélection

Traitement de la case 6 :

La valeur minimale se trouve à la case 8 .

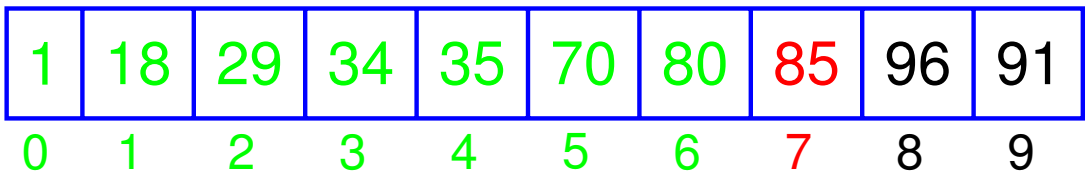


L'échange entre Tab[6]=96 et Tab[8]=80 donne :



Exemple : tri par sélection

Traitement de la case 7 :

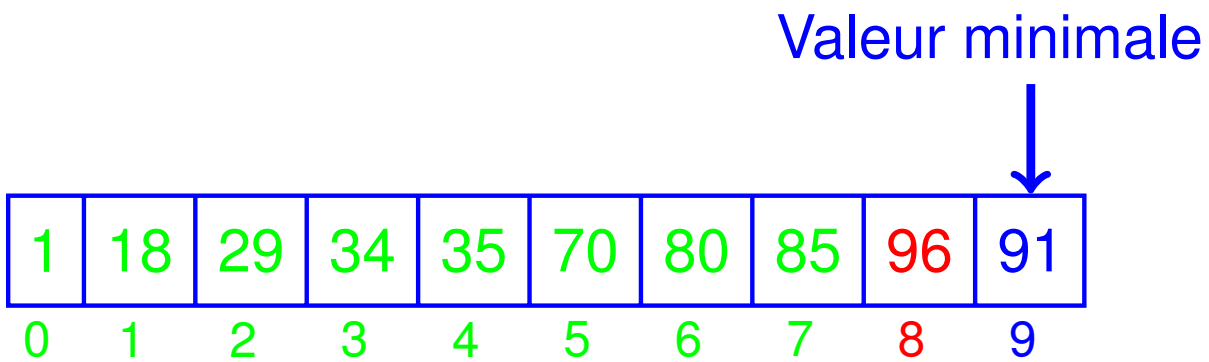


- L'élément T[7] est à sa bonne place
- Aucun échange n'est nécessaire

Exemple : tri par sélection

Traitement de la case 8 :

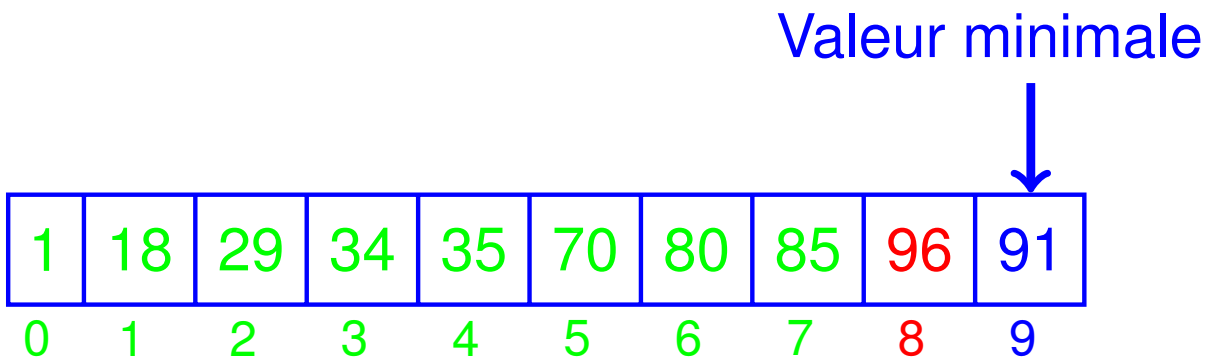
La valeur minimale se trouve à la case 9 .



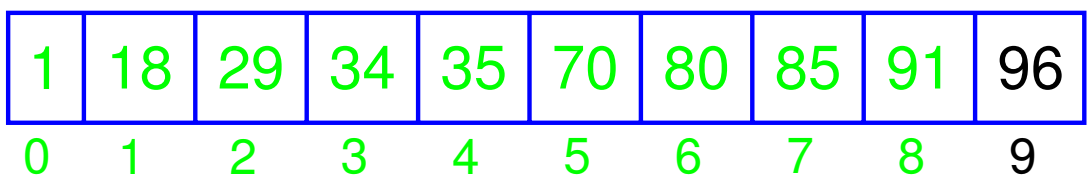
Exemple : tri par sélection

Traitement de la case 8 :

La valeur minimale se trouve à la case 9 .



L'échange entre Tab[8]=96 et Tab[9]=91 donne :



Code source de la fonction Tri par sélection

```
void tri_selection(const int n, int Tab[])
{
    // n = Taille du tableau
  int i,j, indice_min, echange;
  for(i = 0; i < (n-1); i++)
  {
    // Recherche de la valeur minimale entre i et n
  indice_min=i;
  for (j=i+1; j<n; j++)
    if (Tab[j] < Tab[indice_min]) indice_min=j;
            // Echange entre i et indice_min
  echange = Tab[i];
  Tab[i]=Tab[indice_min];
  Tab[indice_min]=echange;
  }
 }
```

Analyse de la complexité : tri par sélection

- Dans la boucle principale, nous exécutons d'abord l'instruction élémentaire d'initialisation : i=0. Cette étape coûte une (1) unité.
- Pour chaque itération *i* de la boucle principale, nous réalisons :
  - deux (2) opérations élémentaires :
    une comparaison $i < (n-1)$; et une incrémentation $i++$;
  - quatre (4) opérations élémentaires :

```
1   indice_min=i;
2   echange = Tab[i];
3   Tab[i]=Tab[indice_min];
4   Tab[indice_min]=echange;
```

  - la boucle intérieure :

```
1   for (j=i+1; j<n; j++)
2     if (Tab[j] < Tab[indice_min]) indice_min=j;
```

Analyse de la complexité : tri par sélection

- Notons *T(n)* le nombre d'instructions nécessaires pour trier un tableau d'entiers de taille *n* en utilisant notre fonction "Tri_sélection".
- Pour un entier *i* compris entre 0 et *n-2*, notons $T_{BI}(i)$ le nombre d'instructions réalisées par la boucle intérieure :

```
1   for (j=i+1; j<n; j++)
2     if (Tab[j] < Tab[indice_min]) indice_min=j;
```

- Nous avons alors :

$$T(n) = 1 + [(6 + T_{BI}(0)) + (6 + T_{BI}(1)) + ... + (6 + T_{BI}\ (n\text{-}2))]$$

$$= 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

- Il nous reste maintenant à calculer $T_{BI}(i)$ pour *i* compris entre 0 et *(n-2)*.

Analyse de la complexité : tri par sélection (cas défavorable)

- La boucle intérieure :

```
1   for (j=i+1; j<n; j++)
2     if (Tab[j] < Tab[indice_min]) indice_min=j;
```

contient une instruction élémentaire d'initialisation : *j=i+1*. Cette étape coûte une (1) unité.

- Pour chaque itération j (de $i+1$ jusqu'à*(n-2)*), nous réalisons :
  - Deux (2) opérations élémentaires :
    une comparaison $j < n$; et une opération d'incrémentation $j++$;.
  - Une (1) opération élémentaire de comparaison :
    Tab[j] $<$ *Tab[indice_min]*
  - Si la condition est vraie, une (1) autre opération élémentaire d'affectation *indice_min=j;* est réalisée.

- Dans le pire des cas, nous obtenons :

$$T_{BI}(i) = 1 + 4 * (n - 1 - i).$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+4*(n-1-0)+1+4*(n-1-1)+...+1+4*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+4*(n-1-0)+1+4*(n-1-1)+...+1+4*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (4*(n-1-0)+4*(n-1-1)+...+4*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+4*(n-1-0)+1+4*(n-1-1)+...+1+4*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (4*(n-1-0)+4*(n-1-1)+...+4*(n-1-(n-2))\ ]$$

$$= 7n-6+[\ (4*(n-1-0)+4*(n-1-1)+...+4*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [\, T_{BI}(0) + T_{BI}(1) + \ldots + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+4*(n-1-0)+1+4*(n-1-1)+\ldots+1+4*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (4*(n-1-0)+4*(n-1-1)+\ldots+4*(n-1-(n-2))\ ]$$

$$= 7n-6+[\ (4*(n-1-0)+4*(n-1-1)+\ldots+4*(n-1-(n-2))\ ]$$

$$= 7n-6+4*[\ (n-1)+(n-2)+\ldots+1\ ]$$

$$= 7n-6+4* \frac{(n-1)*n}{2}$$

Analyse de la complexité : tri par sélection (cas défavorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$
\begin{aligned}
T(n) &= 1 + 6 * (n - 1) + [\, T_{BI}(0) + T_{BI}(1) + \ldots + T_{BI}\ (n\text{-}2)] \\
&= 1 + 6*(n-1)+[\ (1+4*(n-1-0)+1+4*(n-1-1)+\ldots+1+4*(n-1-(n-2))\ ] \\
&= 1 + 6*(n-1)+(n-1)+[\ (4*(n-1-0)+4*(n-1-1)+\ldots+4*(n-1-(n-2))\ ] \\
&= 7n-6+[\ (4*(n-1-0)+4*(n-1-1)+\ldots+4*(n-1-(n-2))\ ] \\
&= 7n-6+4*[\ (n-1)+(n-2)+\ldots+1\ ] \\
&= 7n-6+4*\ \frac{(n-1)*n}{2}
\end{aligned}
$$

Calcul final

$$T(n) = 2n^2 + 5n - 6$$

Analyse de la complexité : tri par sélection (cas favorable)

- La boucle intérieure :

```
1   for (j=i+1; j<n; j++)
2     if (Tab[j] < Tab[indice_min]) indice_min=j;
```

contient une instruction élémentaire d'initialisation : *j=i+1*. Cette étape coûte une (1) unité.

- Pour chaque itération j (de $i+1$ jusqu'à*(n-2)*), nous réalisons :
  - Deux (2) opérations élémentaires :
    une comparaison $j < n$; et une opération d'incrémentation $j++$;.
  - Une (1) opération élémentaire de comparaison :
    Tab[j] $<$ *Tab[indice_min]*
  - Dans le cas favorable, l'autre opération élémentaire d'affectation *indice_min=j;* n'est pas réalisée.

- Dans le cas favorable, nous obtenons :

$$T_{BI}(i) = 1 + 3 * (n - 1 - i).$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+3*(n-1-0)+1+3*(n-1-1)+...+1+3*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+3*(n-1-0)+1+3*(n-1-1)+...+1+3*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (3*(n-1-0)+3*(n-1-1)+...+3*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+3*(n-1-0)+1+3*(n-1-1)+...+1+3*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (3*(n-1-0)+3*(n-1-1)+...+3*(n-1-(n-2))\ ]$$

$$= 7n-6+[\ (3*(n-1-0)+3*(n-1-1)+...+3*(n-1-(n-2))\ ]$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n-1)+[\ (1+3*(n-1-0)+1+3*(n-1-1)+...+1+3*(n-1-(n-2))\ ]$$

$$= 1 + 6*(n-1)+(n-1)+[\ (3*(n-1-0)+3*(n-1-1)+...+3*(n-1-(n-2))\ ]$$

$$= 7n-6+[\ (3*(n-1-0)+3*(n-1-1)+...+3*(n-1-(n-2))\ ]$$

$$= 7n-6+3*[\ (n-1)+(n-2)+...+1\ ]$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n\text{-}1)+[\ (1+3*(n\text{-}1\text{-}0)+1+3*(n\text{-}1\text{-}1)+...+1+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

$$= 1 + 6*(n\text{-}1)+(n\text{-}1)+[\ (3*(n\text{-}1\text{-}0)+3*(n\text{-}1\text{-}1)+...+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

$$= 7n\text{-}6+[\ (3*(n\text{-}1\text{-}0)+3*(n\text{-}1\text{-}1)+...+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

$$= 7n\text{-}6+3*[\ (n\text{-}1)+(n\text{-}2)+...+1\ ]$$

$$= 7n\text{-}6+3* \frac{(n-1)*n}{2}$$

Analyse de la complexité : tri par sélection (cas favorable)

Reprenons le calcul de la complexité de la fonction Trisélection est de :

$$T(n) = 1 + 6 * (n - 1) + [T_{BI}(0) + T_{BI}(1) + ... + T_{BI}\ (n\text{-}2)]$$

$$= 1 + 6*(n\text{-}1)+[\ (1+3*(n\text{-}1\text{-}0)+1+3*(n\text{-}1\text{-}1)+...+1+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

$$= 1 + 6*(n\text{-}1)+(n\text{-}1)+[\ (3*(n\text{-}1\text{-}0)+3*(n\text{-}1\text{-}1)+...+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

$$= 7n\text{-}6+[\ (3*(n\text{-}1\text{-}0)+3*(n\text{-}1\text{-}1)+...+3*(n\text{-}1\text{-}(n\text{-}2))\ ]$$

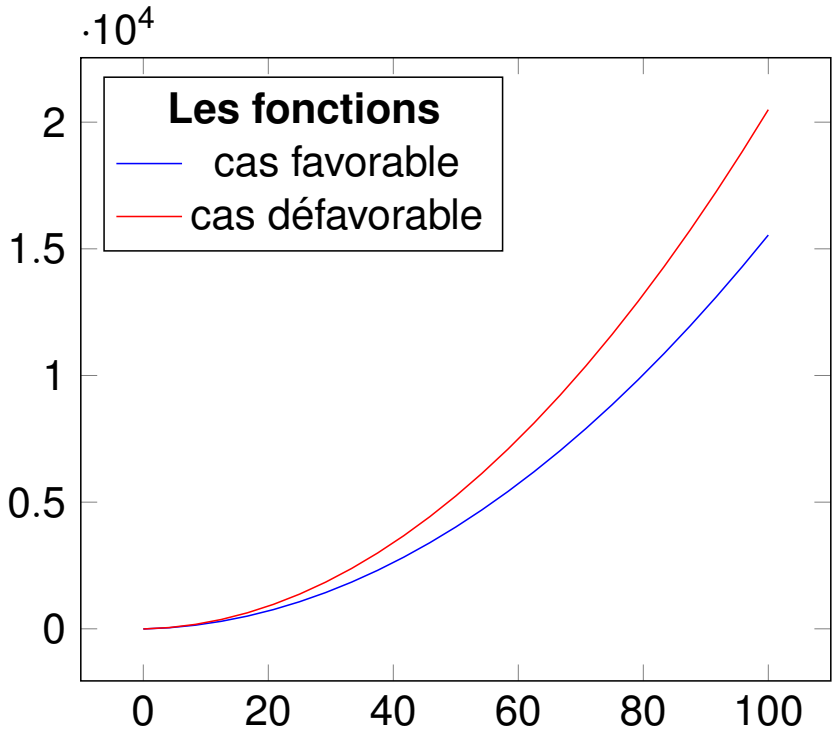
$$= 7n\text{-}6+3*[\ (n\text{-}1)+(n\text{-}2)+...+1\ ]$$

$$= 7n\text{-}6+3*\ \frac{(n-1)*n}{2}$$

Calcul final

$$T(n) = \frac{3}{2}.n^2 + \frac{11}{2}.n - 6.$$

Comparaison entre cas favorable et cas défavorable



Code source amélioré de la fonction tri par sélection

Dans la fonction du tri par sélectiion, l'échange est systématiquement effectué entre l'indice *i* et *indice_min*.
L'idée est de mettre simplement la séquence:

```
            // Echange entre i et indice_min
  echange = Tab[i];
  Tab[i]=Tab[indice_min];
  Tab[indice_min]=echange;
```

au conditionnel :

```
            // Echange entre i et indice_min si i != indice_min
  if (i != indice_min)
  { echange = Tab[i];
    Tab[i]=Tab[indice_min];
    Tab[indice_min]=echange;
  }
```

Pour les algorithmes de tri: le nombre d'échanges

- Un instruction élémentaire est souvent utilisée comme unité de mesure pour le calcul de la complexité des algorithmes.
- Pour les algorithmes de tri, le nombre de permutations (ou d'échanges ou d'inversions) entre les éléments d'un tableau est souvent utilisé comme critère de comparaison.

Calculer le nombre d'instructions élémentaires dans une boucle "Pour"

Réponse

Souvent pour calculer le nombre d'instructions dans une boucle "Pour", on calcule le nombre d'instructions $T(I)$ dans I (dans le pire des cas) et on définit :

$$T(n) = 1 + n * (2 + T(I)).$$

Calculer le nombre d'instructions élémentaires

Boucle Tantque

Comment définir le nombre d'instructions associé à des boucles "Pour" de la forme :

```
Tantque (Condition)
Faire
        I;
Finfaire
```

Calculer le nombre d'instructions élémentaires

Boucle Tantque

La réponse est plus difficile car il n'est pas facile de déterminer le nombre de fois q que le corps de la boucle "Tantque" est exécuté. Le nombre d'instructions est :

$$T(n) = q * (T(condition) + T(I)).$$

Calculer le nombre d'instructions élémentaires

Fonctions récursives

Un cours est dédié aux fonctions récursive. $T(n)$ sera définie en fonction de $T(n-1)$, $T(n-2)$,etc.

Un autre exemple : test de primalité

Exemple

Evaluons la fonction ”nombre de nombres premiers compris entre 1 et n” avec un test de la primalité qui s’arrête à la racine carrée de n.

Un autre exemple : le test de primalité

```
int Estpremierracine2(const int n)
{
 int i, j=(int) (sqrt(n)+1);
 if( n <= 1) return 0;
 else if (n==2) return 1;
 for(i = 2; i <= j;i ++)
  if(n % i == 0) return 0;
 return 1;
}
```

Analyse de la complexité : Test de primalité

- En dehors de la boucle, nous exécutons d'abord (4) instructions élémentaires pour $n > 2$ (Pour $n \leq 2$, le nombre d'instructions total est de 5).

```
1  j=(int) (sqrt(n)+1); // Une affection
2  if( n <= 1) return 0;  // Une comparaison
3  else if (n==2) return 1; // Une comparaison
4  return 1; // Un retour de valeur
```

Analyse de la complexité : Test de primalité (suite)

- Dans la boucle,

```
1  for(i = 2; i <= j;i ++)
2   if(n % i == 0) return 0;
```

nous exécutons d'abord une (1) instruction d'initialisation : i=2.

- Pour chaque itération i de la boucle (de 2 jusqu'à $j = \sqrt{n} + 1$), nous réalisons trois (3) opérations élémentaires (dans le pire des cas) :
  - une comparaison $i <= j$;
  - une incrémentation $i++$;
  - une comparison d'expressions élémentaires : $(n \,\%\, i == 0)$

Analyse de la complexité : Test de primalité

Nombre d'instructions pour le test de primalité

Le nombre d'instructions élémentaires réalisées par la fonction de test de primalité est donc:

$$T_{\text{Pr2}}(n) = \begin{cases} 4 & \text{si n } \leq 2 \\ 5 + 3 * \sqrt{n} & \text{sinon} \end{cases}$$