Les tableaux [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.

Les tableaux

Motivations

Exercice

Ecrire un programme C qui :

- lit dix entiers relatifs et
- affiche ces entiers dans l'ordre inverse de la lecture.

Exemple

- si les nombres lus sont :

  12 16 -3 8 9 4 3 4 2 11

- alors le programme affichera :

  11 2 4 3 4 9 8 -3 16 12

Motivations

Voici donc une solution naïve et fastidieuse (pas de soucis elle marche!).

```
#include <stdio.h>
int main (void)
{
    int a0, a1, a2,a3,a4,a5,a6,a7,a8,a9;
    printf("\nMerci d'introduire 10 nombres relatifs : ");
    scanf("%d%d%d%d%d%d%d%d%d%d",
         &a0,&a1,&a2,&a3,&a4,&a5,&a6,&a7,&a8,&a9);
    printf("L'affichage des nombres dans l'ordre inverse de lecture est :
         %d%d%d%d%d%d%d%d%d%d \n", a9,a8,a7,a6,a5,a4,a3,a2,a1,a0);
    return(0);
}
```

Motivations

Remarques

- L'écriture de ce programme est facile
- Cependant, il faut déclarer autant de variables que de valeurs à lire (ici 10)!
- Ces déclarations sont nécessaires pour mémoriser les données lues pour un traitement futur (dans notre cas un affichage dans l'ordre inverse de l'introduction des données).
- Ce problème aurait été impossible à traiter, même pour un programmeur particulièrement motivé, si le nombre de variables est égal à 1000 (déclarer 1000 variables n'est pas concevable).

Motivations

Les tableaux informellement ...

- Un nouveau type de données, appelé tableau.
  - J'insiste : un nouveau type de données
- Les tableaux permettent de stocker un ensemble de données du même type.

Motivations

Les tableaux informellement ...

- Un nouveau type de données, appelé tableau.
- Les tableaux permettent de stocker un ensemble de données du même type.
- Dans un de nos exemples, la déclaration :

```
#include <stdio.h>
int main (void)
{
    int a0, a1, a2,a3,a4,a5,a6,a7,a8,a9;
}
```

sera remplacée par :

Motivations

Les tableaux informellement ...

- Un nouveau type de données, appelé tableau.
- Les tableaux permettent de stocker un ensemble de données du même type.
- Dans un de nos exemples, la déclaration :

```
#include <stdio.h>
int main (void)
{
    int a0, a1, a2,a3,a4,a5,a6,a7,a8,a9;
}
```

sera remplacée par :

```
#include <stdio.h>
int main (void)
{
    int a[10];
}
```

Motivations

Les tableaux informellement ...

Cette déclaration :

```
#include <stdio.h>
int main (void)
{
    int a[10];
}
```

indique que :

- La variable ”a” est de type de tableau de 10 éléments.
- Chaque élément est de type ”int”

Question ?

Question

Question

Quelle est la différence entre les deux déclarations suivantes :

```
int i;
```

et

```
const int i;
```

Question

Variable et constante

Quelle est la différence entre les deux déclarations suivantes :

```
int i=2;
```

et

```
const int i=2;
```

Réponse

- Dans le premier cas, le contenu de la variable *i* peut être modifié alors que
- dans le deuxième cas, la variable *i* gardera la même valeur d'initialisation (c'est-à-dire 2) tout au long de la vie du programme.

Tableaux : propriétés importantes

Les principales caractéristiques à retenir pour les tableaux

- Un tableau est une structure de données qui permet de travailler avec plusieures données de même type.
- Le nombre maximal de données à stocker est connu avant l'exécution du programme.
  - Même si de plus en plus de compilateurs déclarent des tableaux avec des tailles variables.

Les principales caractéristiques à retenir pour les tableaux

- Un tableau est une variable constante (donc sa valeur, plutôt son adresse, ne peut-être modifiée).
- Une fois la déclaration est faite, un tableau est une variable qui se comporte comme une variable de type pointeur.
  - J'insiste : il se comporte et non il devient ...
- C'est-à-dire après la déclaration la variable tableau contient une adresse.

Tableaux

- Les éléments d'un tableau peuvent être de :
  - un type simple
  - un type complexe (enregistrement par exemple)
- Chaque élément d'un tableau est représenté par un indice indiquant sa position
- La taille d'un tableau est fixe.
- L'espace mémoire est réservé au niveau de la compilation.

Tableaux à une entrée

- Déclaration :

  type Nom_du_tableau [C];

- Les indices varient entre 0 et C-1.

- Chaque élément du tableau est une variable indicée, identifiée par le nom du tableau suivi de l'indice entre crochets.

Tableaux à une entrée

- Déclaration :

  type Nom_du_tableau [C];

- ATTENTION : C est une constante (sa valeur est connu avant l'exécution).
- On peut initialiser un tableau :
  - *int T[5]={1,3,-3,7,8};*
  - voire *int T[]={1,3,-3,7,8};*

Tableaux à une entrée

- Supposons que nous avons la déclaration suivante :

  int T [10];

- Alors :
  - Les éléments du tableau sont désignés par : T[0], T[1], ..., T[9].
  - Ou bien par :
    - *T, *(T+1), *(T+2), ..., *(T+9)
    - puisqu'un tableau se comporte comme un pointeur.
  - De même (T+i) donne l'adresse de l'élément T[i].
    - Dis autrement, (T+i) est égal à (&T[i]).

Pointeurs vs tableaux

Question

Quelle différence entre les deux déclarations :

int T [10];

et

*int *p =malloc(10*sizeof(int);*

Pointeurs vs tableaux

Question

Quelle différence entre les deux déclarations :

int T [10];

et

*int *p =malloc(10*sizeof(int);*

D'abord les points communs

- Après la déclaration p et T contiendront des adresses.
- Utilisent les mêmes méthodes pour accéder aux éléments :
  - . *(p+i) ou
  - p[i].
- Les éléments se trouvent dans des espaces mémoires sont contingents.

Pointeurs vs tableaux

Question

Quelle différence entre les deux déclarations :

int T [10];

et

*int *p =malloc(10*sizeof(int);*

Les différences

- L'espace réservé pour T est obtenu au niveau de la compilation (et non au niveau de l'exécution).
- p est une variable alors que T est une constante.
- De ce fait, l'instruction p=q est permise alors T=q ne l'est pas.
- De même l'instruction p++ est autorisée alors que T++ ne l'est pas.

Remarque importante : ne confondons pas

Un tableau est un tableau

et

Un pointeur est un pointeur

Tableaux vs pointeurs

- A la question ”un tableau est-il un pointeur ?”. La réponse est :
  - une variable simple est une variable simple,
  - un tableau est un tableau,
  - un pointeur est un pointeur, et
  - une vache est une vache!
- Mais une fois la déclaration est faite, la variable tableau se comporte comme (ou est transformé en ou est désintégrée en) en une variable de type pointeur constant.
- Et si vous vous allez loin, il y aura un exercice facultatif qui sera donnée dans un des TD et/ou TP.

Paramètres de fonctions

Les tableaux : paramètres de fonctions

Il existe trois différentes façons de déclarer

un tableau comme paramètre d'une fonction.

Les tableaux : paramètres de fonctions

Il existe trois différentes façons de déclarer un tableau comme paramètre d'une fonction.

- Dans cette première déclaration la taille est précisée (déclaration standard) :

  typefonction nomfonction (typetab tableau[taille], ...)

- "..." fait références à d'autres arguments de la fonction (et il ne signifie pas un nombre variable d'arguments).

Les tableaux : paramètres de fonctions

- Attention : le nombre d'éléments doit-être spécifié (soit via #define ou via l'ajout d'un paramètre).

Les tableaux : paramètres de fonctions

Il existe trois différentes façons de déclarer un tableau comme paramètre d'une fonction.

- Dans cette deuxième déclaration la taille n'est pas précisée (déclaration standard) :

  typefonction nomfonction (typetab tableau[], ...)

- La deuxième déclaration est intéressante car elle permet d'utiliser la fonction avec des tableaux de tailles différentes.

Les tableaux : paramètres de fonctions

- Il existe trois différentes façons de déclarer un tableau comme paramètre d'une fonction.
- Dans la troisième déclaration on utilise un pointeur :
  *typefonction nomfonction (typetab * tableau, ...)*
- En réalité c'est ce que l'on fait avec les deux premières déclarations!

Les tableaux : paramètres de fonctions

Remarque 1

Dans l'appel aux fonctions, Il ne faut pas mettre "&" lorsque vous passez les tableaux comme paramètres de fonctions.

Exemples

Exercice: Tableau

Exercice facile

Ecrire une fonction booléenne qui vérifie si un tableau d'entiers est trié de manière décroissante.

Vérification si un tableau est trié

```
2  int est_trie (int A[], int N)
3  {
4   int i=0;
5   while (i<(N-1))
6   {
7    if (A[i]<A[i+1]) return 0;
8    i++;
9   }
0   return 1;
   }
```

Vérification si un tableau est trié : une autre écriture

```
2  int est_trie (int A[5], int N)
3  {
4   int i=0;
5   while (i<(N-1))
6   {
7    if (A[i]<A[i+1]) return 0;
8    i++;
9   }
0   return 1;
   }
```

Vérification si un tableau est trié : encore une autre écriture

```
2  int est_trie (int *A, int N)
3  {
4   int i=0;
5   while (i<(N-1))
6   {
7    if (A[i]<A[i+1]) return 0;
8    i++;
9   }
0   return 1;
   }
```

Vérification si un tableau est trié : appel à la fonction

```
#include <stdio.h>
int est_trie (int A[], int N)
{
 int i=0;
 while (i<(N-1))
 {
  if (A[i]<A[i+1]) return 0;
  i++;
 }
 return 1;
}


int main (void)
{
 int b[3]={2,1,5};
 if (est_trie(b, 3)==0)
  printf ("Aie ... Le tableau n'est pas trié \\");
 else
  printf ("Oh yes ... Le tableau est trié \\");
 return 0;
}
```

Un exercice
à faire en TD

Pomme, pêche, poire, abricot

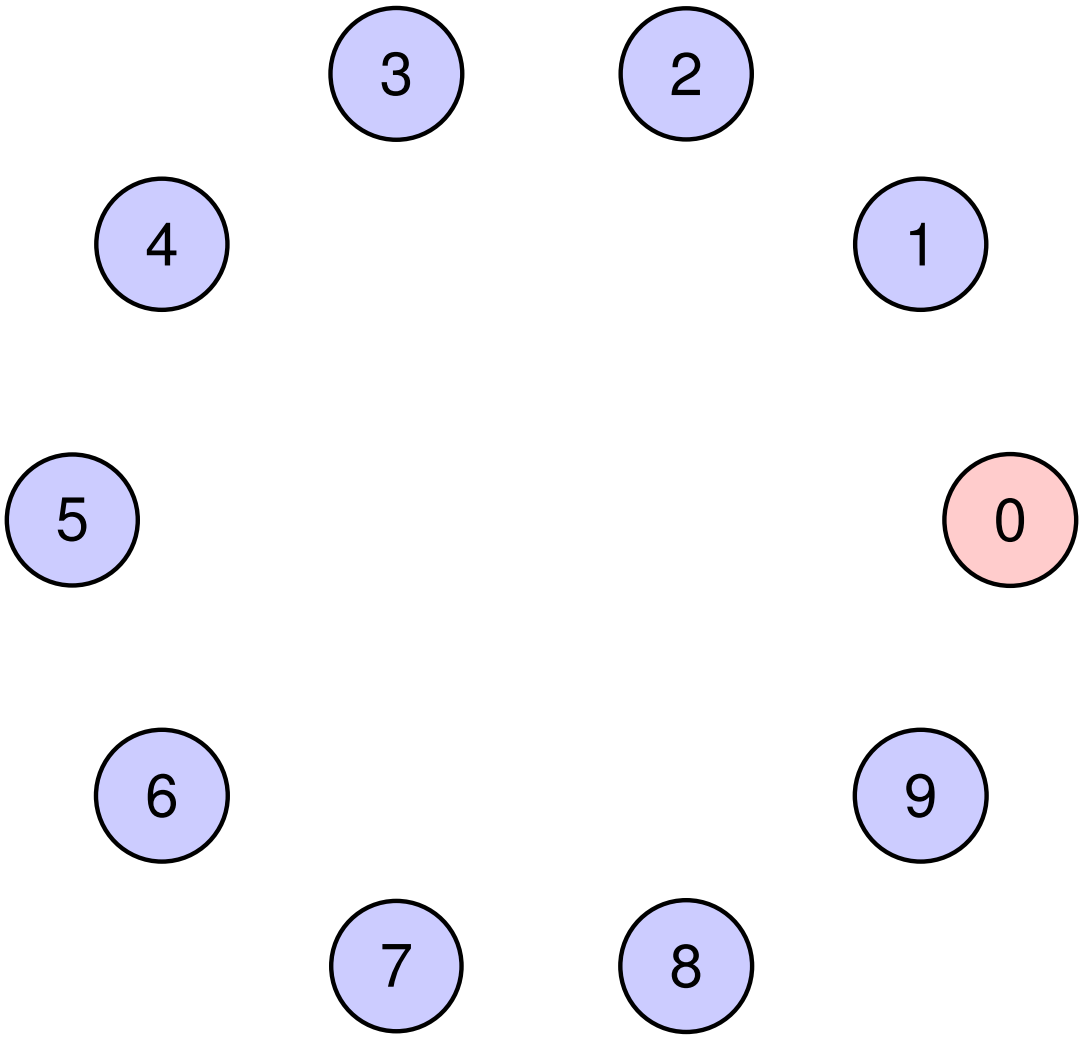
Pomme, pêche, poire, abricot
Y'en a une, y'en a une
Pomme, pêche, poire, abricot
Y'en a une qui est en trop.

Exercice : jeu

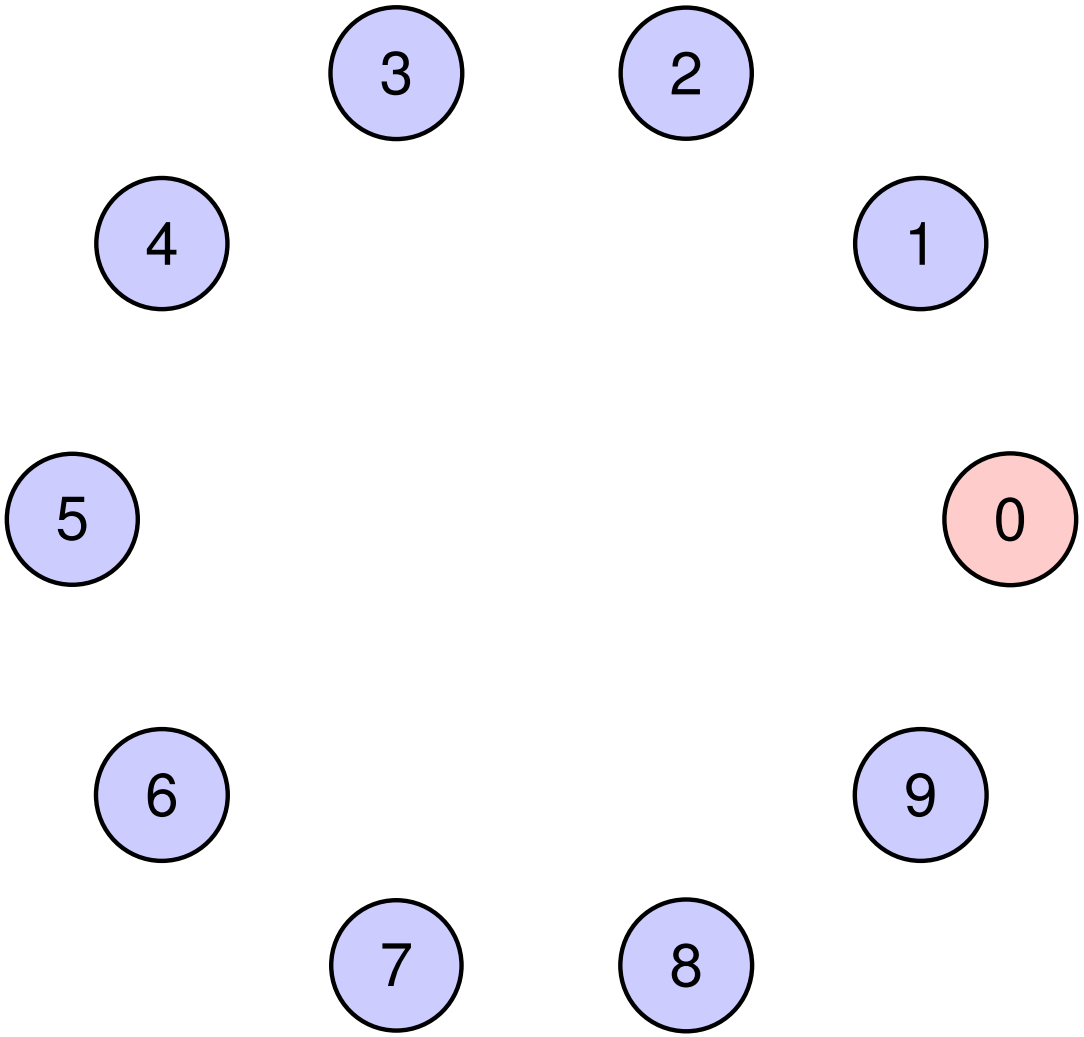
Ecrire une fonction qui code le jeu suivant :

- Paramètres :
  - On se donne N joueurs (numérotés de 0 à N-1). N est une constante définie avec #define
  - On se donne (en paramètre) un nombre M
- Règles du jeu :
  - On compte de 1 à M de manière répétitive.
  - Le $M^{ième}$ joueur est éliminé
  - Le jeu s'arrête lorsqu'il ne reste qu'un seul joueur (le gagnant).

Déroulement du Jeu : Il reste 10 joueurs

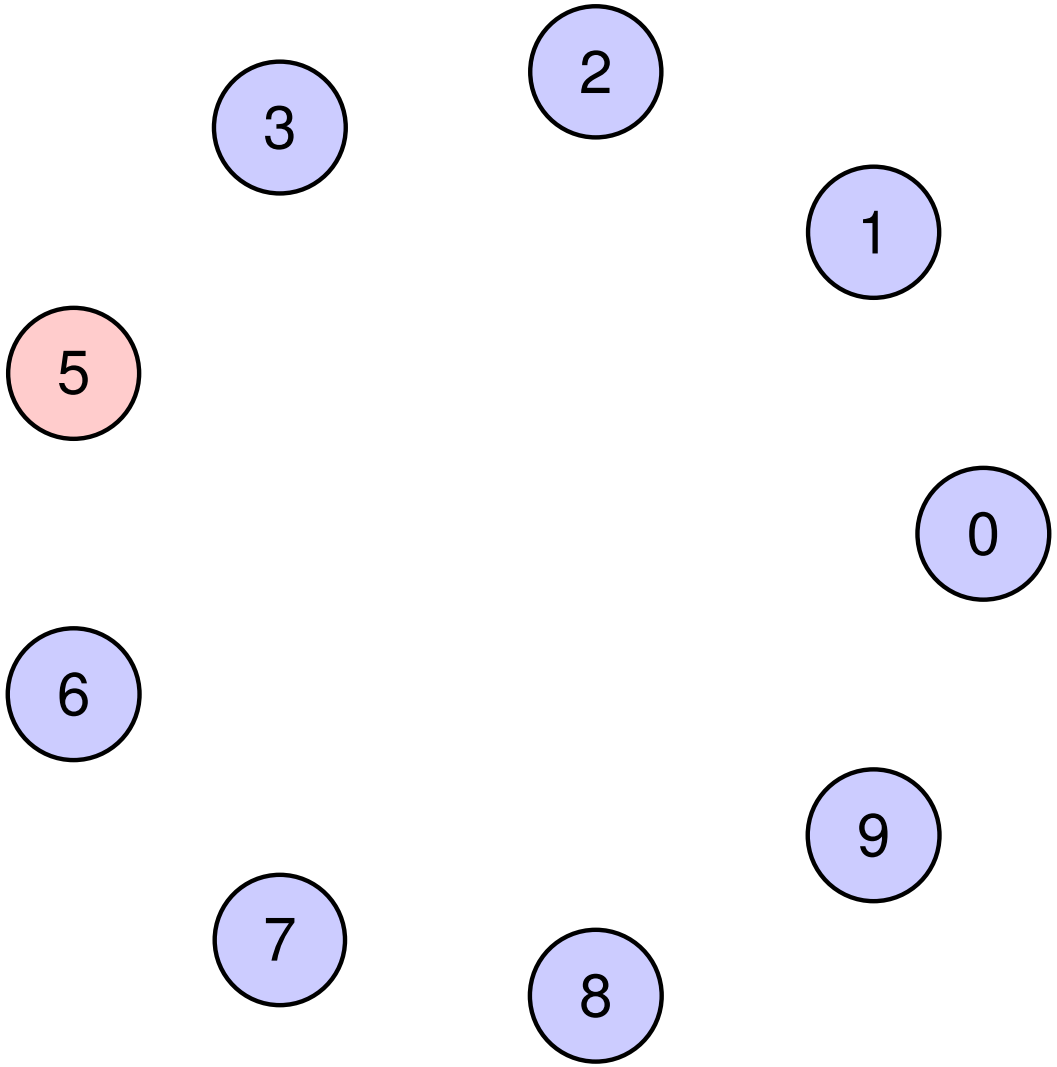


Déroulement du Jeu : Il reste 10 joueurs



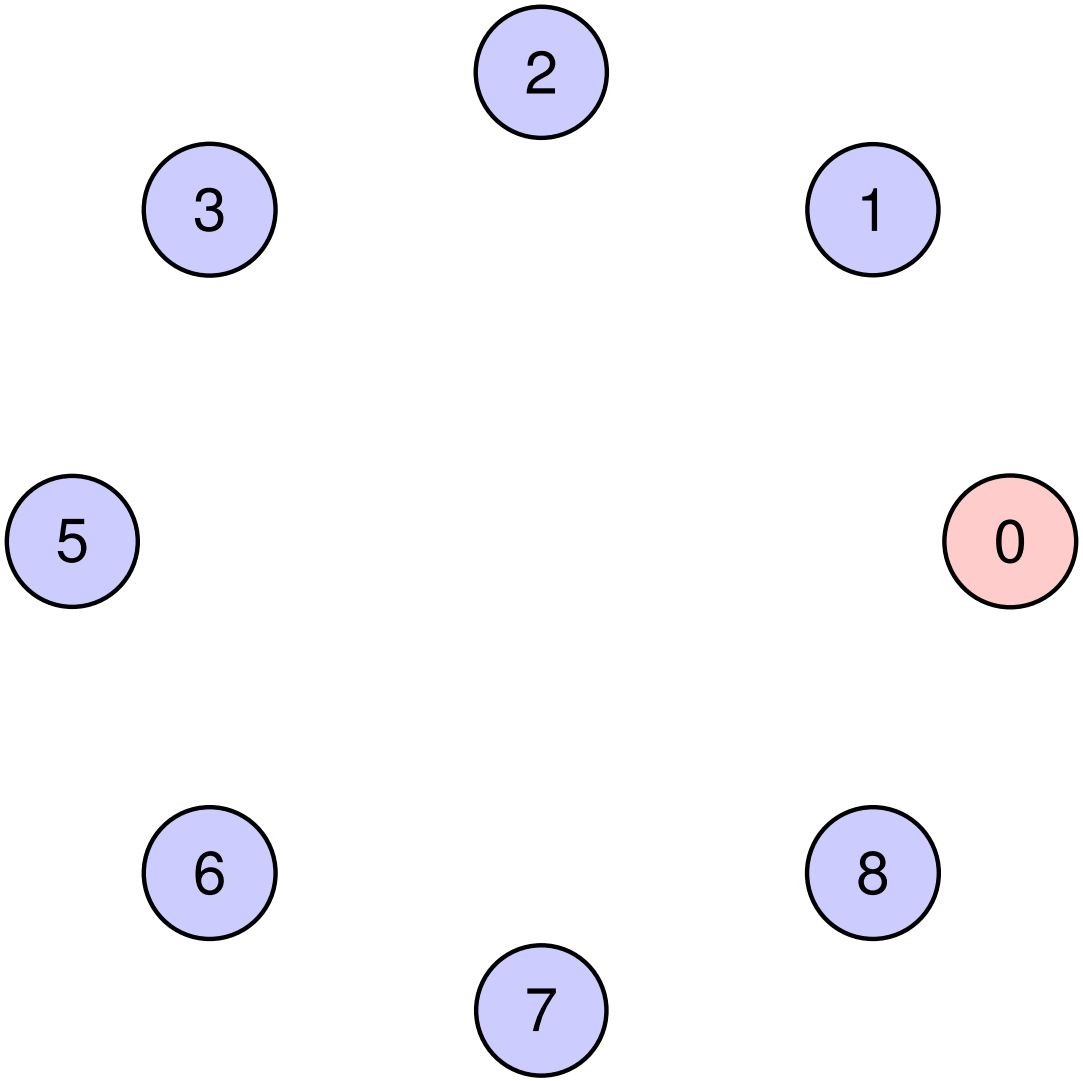
Le joueur 4 sera éliminé.

Déroulement du Jeu : Il reste 9 joueurs



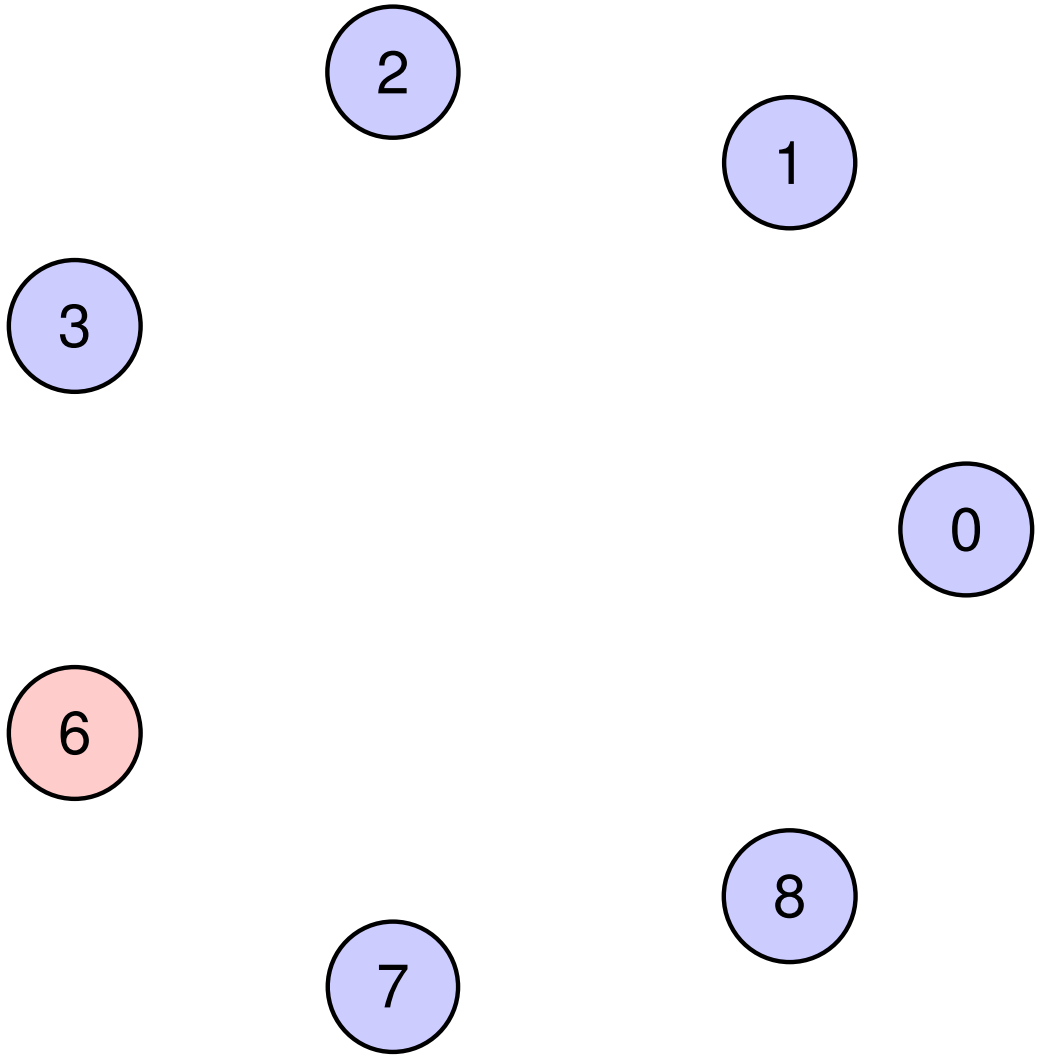
Le joueur 9 sera éliminé.

Déroulement du Jeu : Il reste 8 joueurs



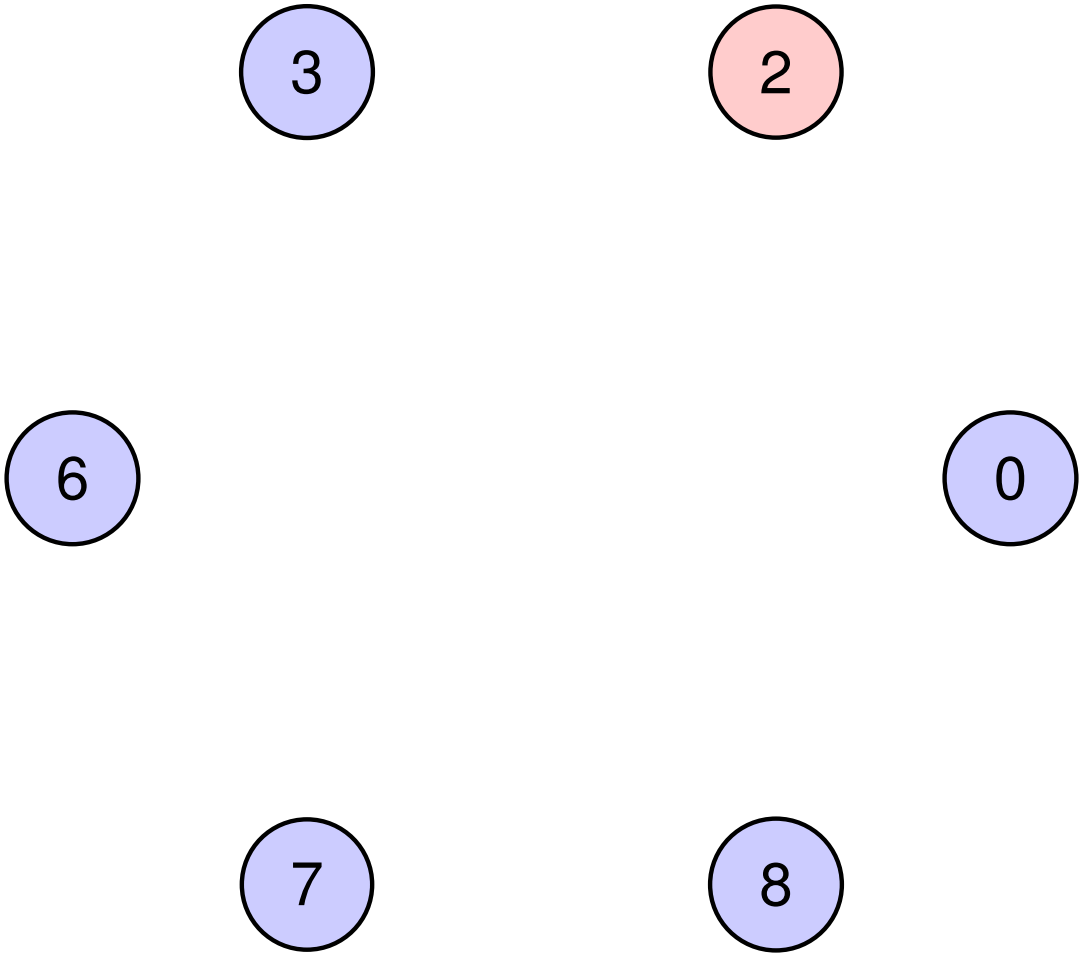
Le joueur 5 sera éliminé.

Déroulement du Jeu : Il reste 7 joueurs



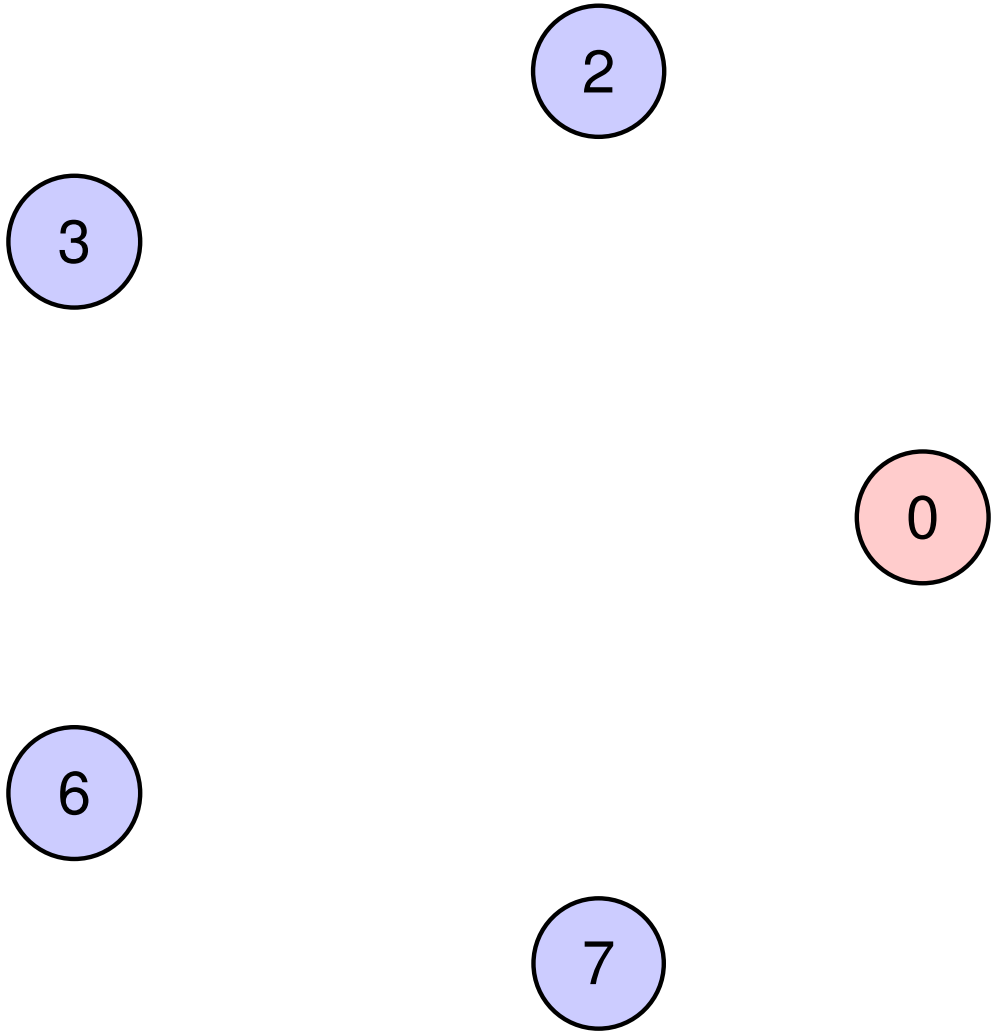
Le joueur 1 sera éliminé.

Déroulement du Jeu : Il reste 6 joueurs



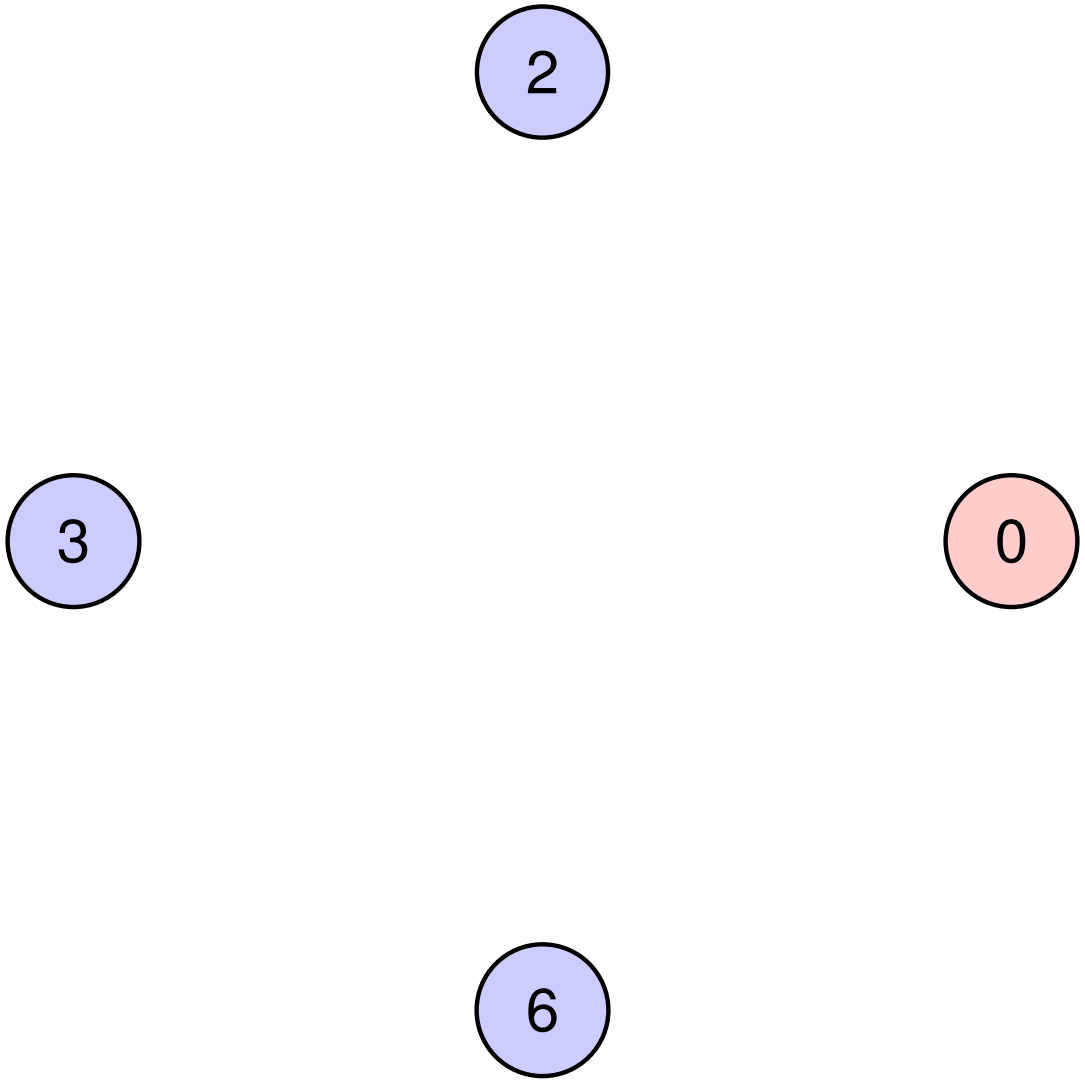
Le joueur 8 sera éliminé.

Déroulement du Jeu : Il reste 5 joueurs



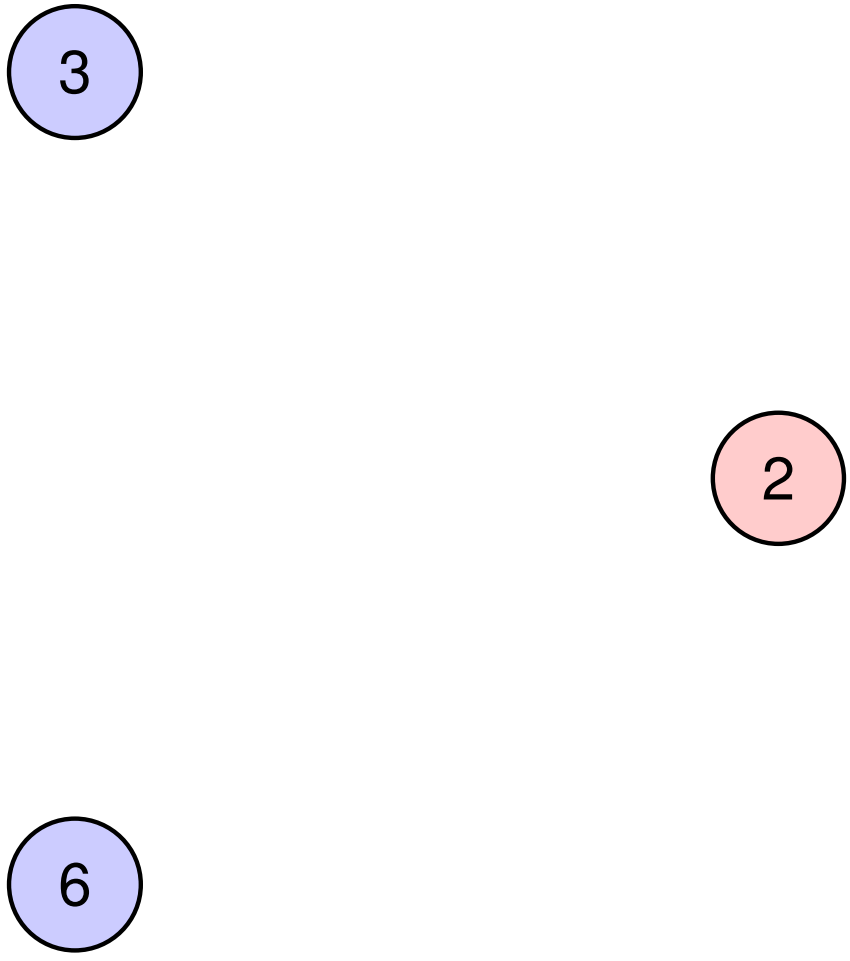
Le joueur 7 sera éliminé.

Déroulement du Jeu : Il reste 4 joueurs



Le joueur 0 sera éliminé.

Déroulement du Jeu : Il reste 3 joueurs



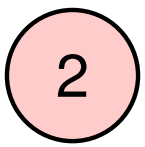
Le joueur 3 sera éliminé.

Déroulement du Jeu : Il reste 2 joueurs

Le joueur 6 sera éliminé.

Déroulement du Jeu : Il reste un et un seul joueur

Le gagnant est :



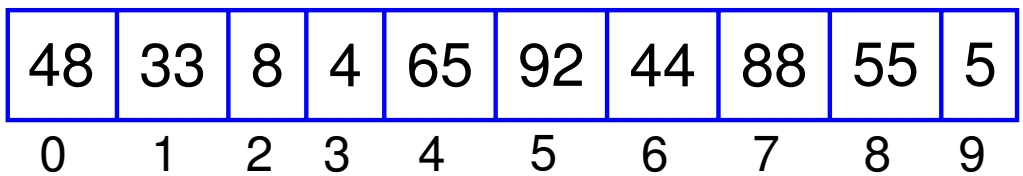
exercice

Exercice

- Déclarer une constante *k* (par exemple égale à 10) grâce à la directive #define.
- Écrire une fonction, appelée echange, qui échange deux entiers.
- Ecrire une fonction, tri_bulles, qui
  - prend en entrée un tableau d'entiers et une variable constante notée *taille*.
  - Cette fonction réalise le tri par bulles du tableau de *taille* entiers.
- Tester votre fonction depuis le programme principal main.

Exemple : tri par bulles

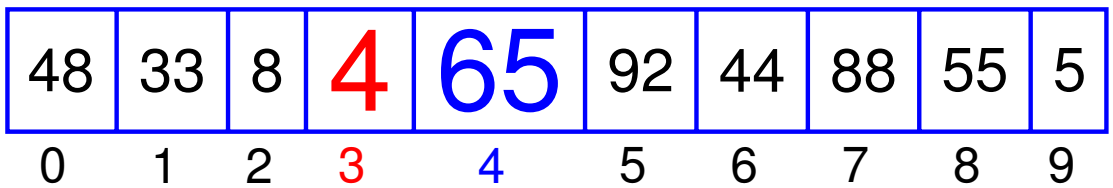
Voici le tableau initial à trier :



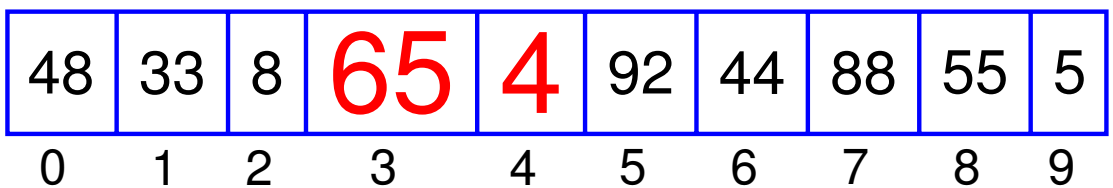
Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[3]=4 et Tab[4]=65 ne sont pas ordonnées.



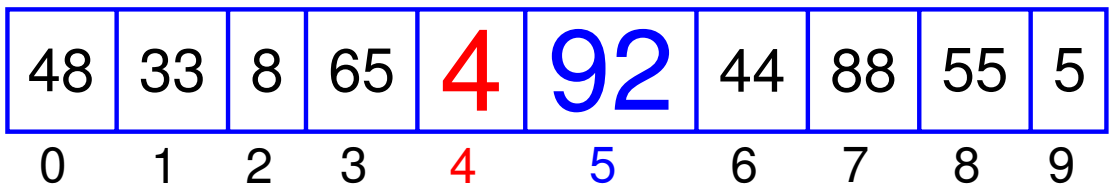
L'échange entre Tab[3]=4 et Tab[4]=65 donne :



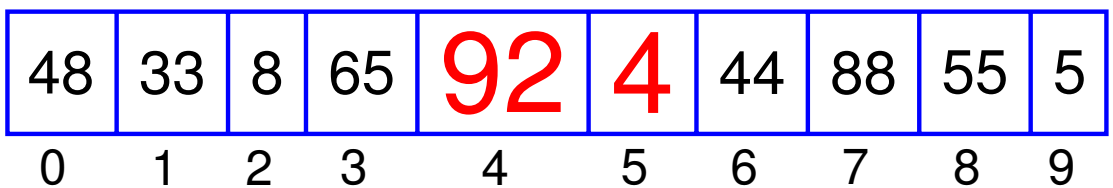
Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[4]=4 et Tab[5]=92 ne sont pas ordonnées.



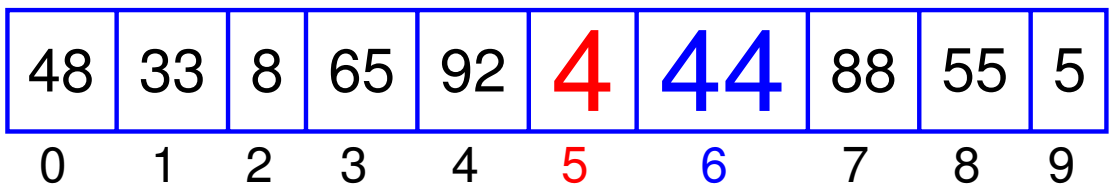
L'échange entre Tab[4]=4 et Tab[5]=92 donne :



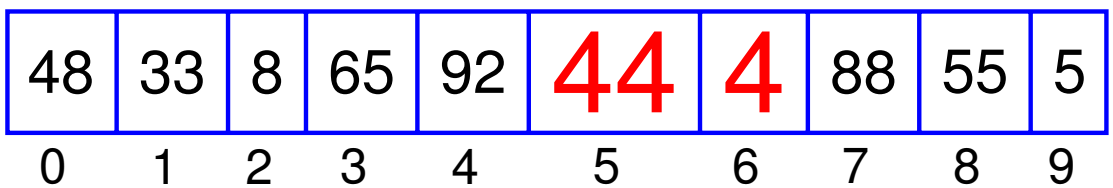
Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[5]=4 et Tab[6]=44 ne sont pas ordonnées.



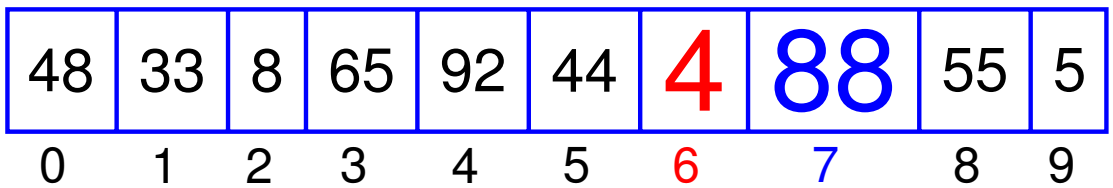
L'échange entre Tab[5]=4 et Tab[6]=44 donne :



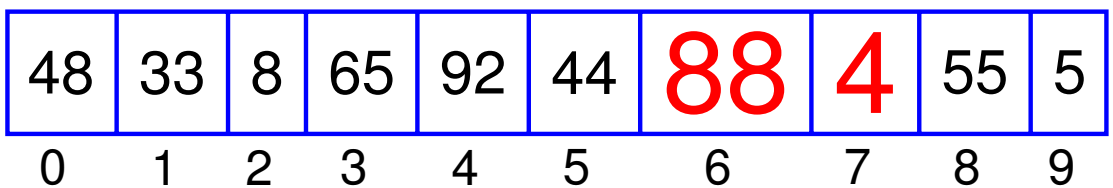
Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[6]=4 et Tab[7]=88 ne sont pas ordonnées.



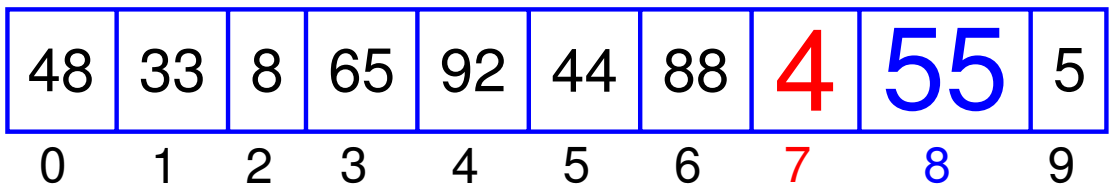
L'échange entre Tab[6]=4 et Tab[7]=88 donne :



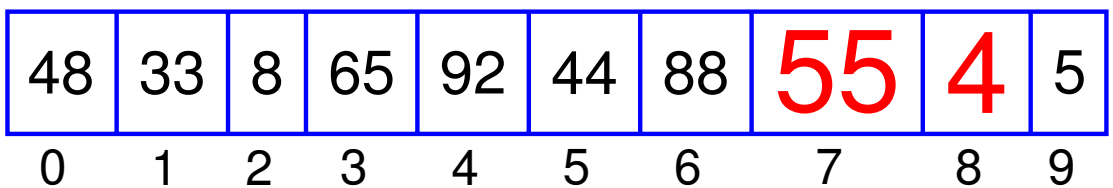
Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[7]=4 et Tab[8]=55 ne sont pas ordonnées.



L'échange entre Tab[7]=4 et Tab[8]=55 donne :



Exemple : tri par bulles

Traitement de l'iteration 1 :

Les deux cases sucessives Tab[8]=4 et Tab[9]=5 ne sont pas ordonnées.

48	33	8	65	92	44	88	55	4	5
0	1	2	3	4	5	6	7	8	9

L'échange entre Tab[8]=4 et Tab[9]=5 donne :

48	33	8	65	92	44	88	55	5	4
0	1	2	3	4	5	6	7	8	9

Exemple : tri par bulles

Fin de l'itération 1 :

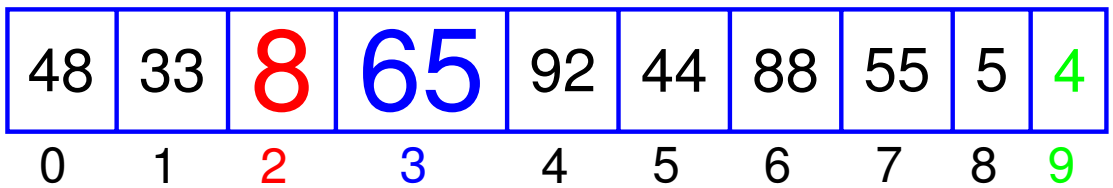
Après l'itération 1, l'élément Tab[9]=4 est à sa bonne place

48	33	8	65	92	44	88	55	5	4
0	1	2	3	4	5	6	7	8	9

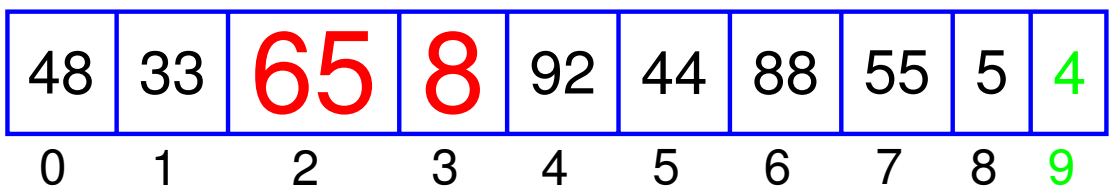
Exemple : tri par bulles

Traitement de l'iteration 2 :

Les deux cases sucessives Tab[2]=8 et Tab[3]=65 ne sont pas ordonnées.



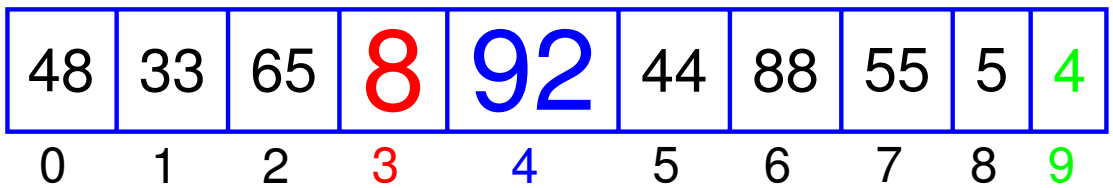
L'échange entre Tab[2]=8 et Tab[3]=65 donne :



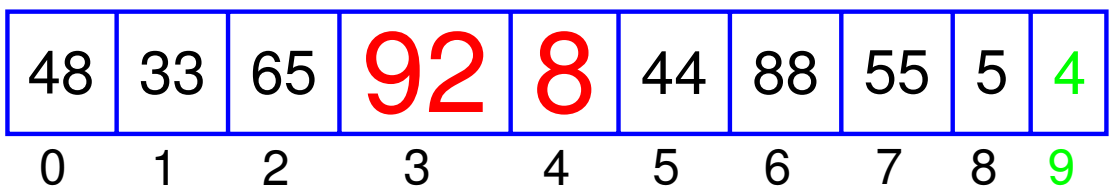
Exemple : tri par bulles

Traitement de l'iteration 2 :

Les deux cases sucessives Tab[3]=8 et Tab[4]=92 ne sont pas ordonnées.



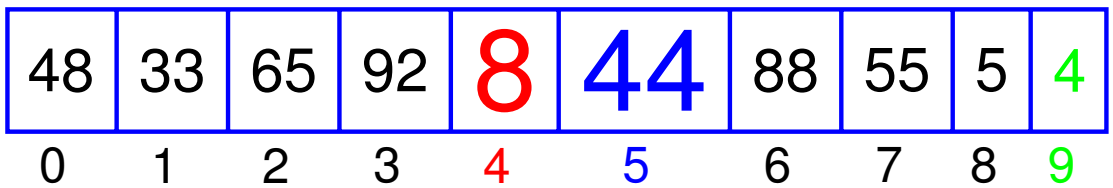
L'échange entre Tab[3]=8 et Tab[4]=92 donne :



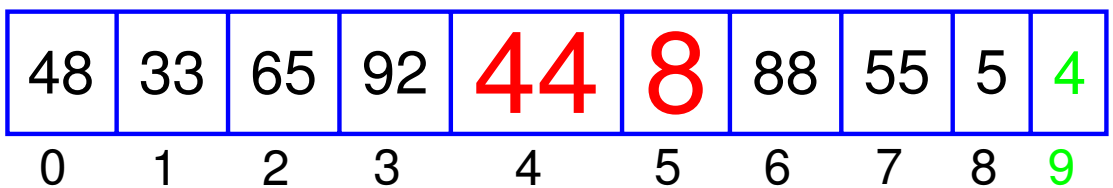
Exemple : tri par bulles

Traitement de l'iteration 2 :

Les deux cases sucessives Tab[4]=8 et Tab[5]=44 ne sont pas ordonnées.



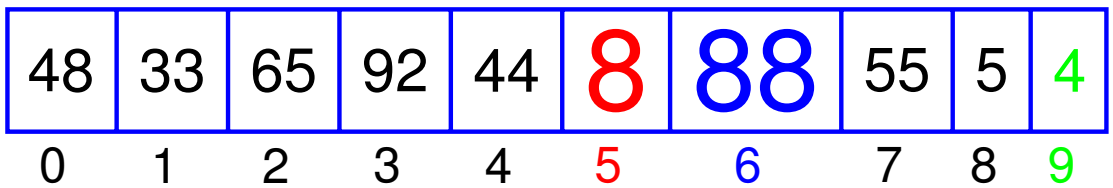
L'échange entre Tab[4]=8 et Tab[5]=44 donne :



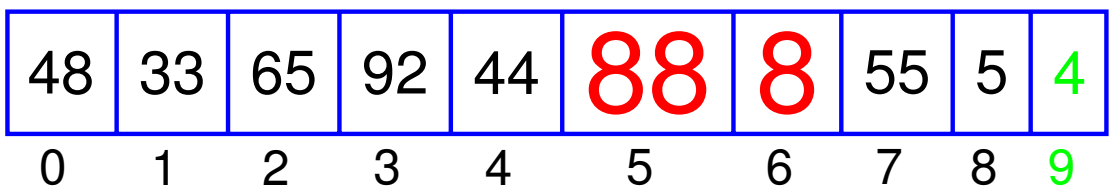
Exemple : tri par bulles

Traitement de l'iteration 2 :

Les deux cases sucessives Tab[5]=8 et Tab[6]=88 ne sont pas ordonnées.



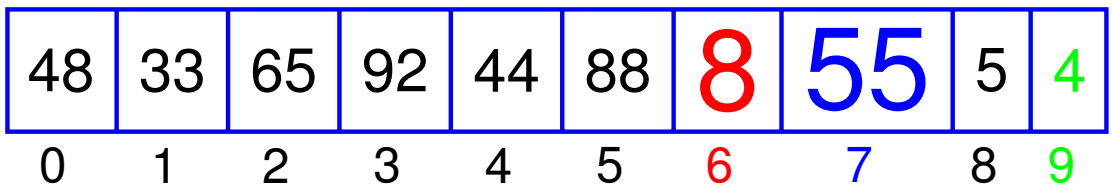
L'échange entre Tab[5]=8 et Tab[6]=88 donne :



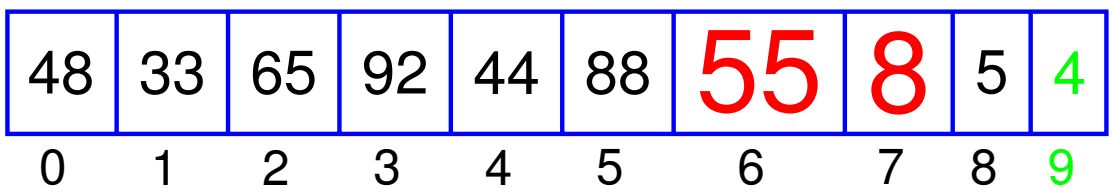
Exemple : tri par bulles

Traitement de l'iteration 2 :

Les deux cases sucessives Tab[6]=8 et Tab[7]=55 ne sont pas ordonnées.



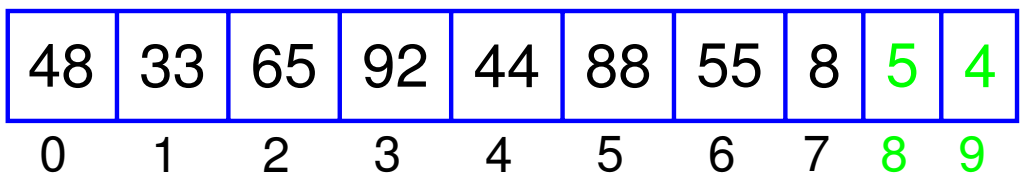
L'échange entre Tab[6]=8 et Tab[7]=55 donne :



Exemple : tri par bulles

Fin de l'itération 2 :

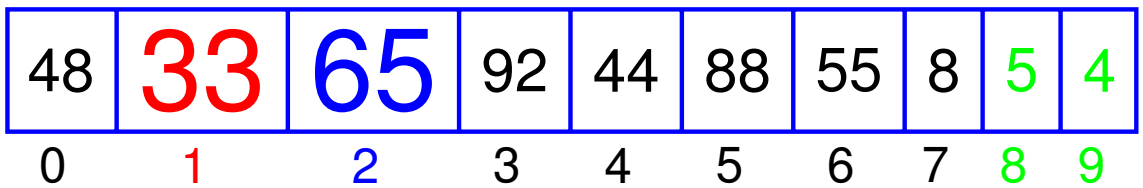
Après l'itération 2, l'élément Tab[8]=5 est à sa bonne place



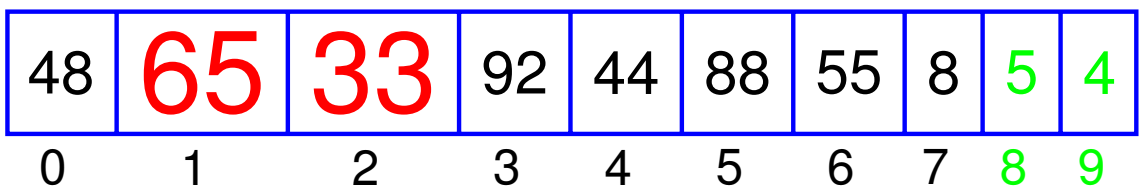
Exemple : tri par bulles

Traitement de l'iteration 3 :

Les deux cases sucessives Tab[1]=33 et Tab[2]=65 ne sont pas ordonnées.



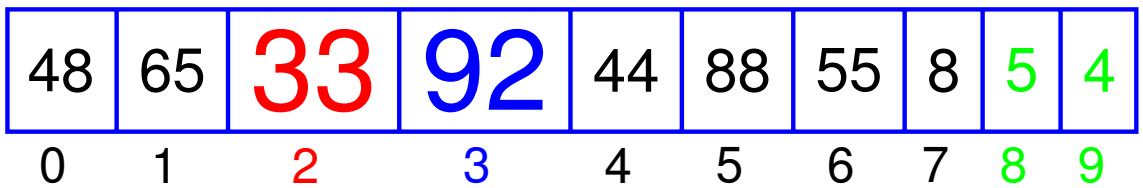
L'échange entre Tab[1]=33 et Tab[2]=65 donne :



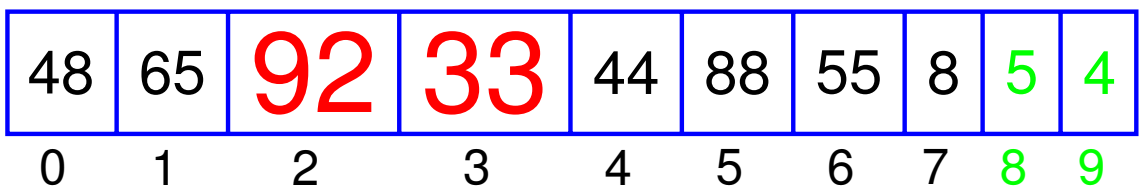
Exemple : tri par bulles

Traitement de l'iteration 3 :

Les deux cases sucessives Tab[2]=33 et Tab[3]=92 ne sont pas ordonnées.



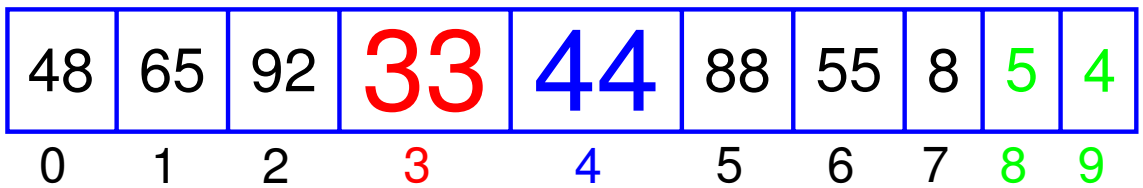
L'échange entre Tab[2]=33 et Tab[3]=92 donne :



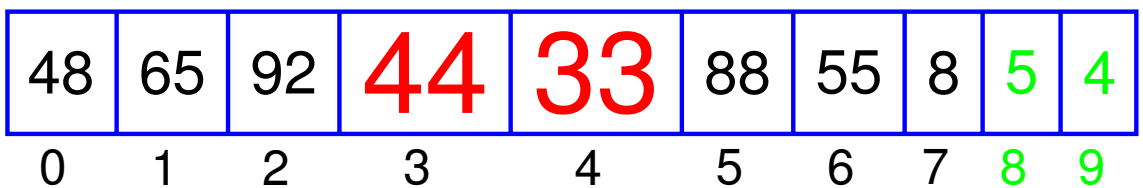
Exemple : tri par bulles

Traitement de l'iteration 3 :

Les deux cases sucessives Tab[3]=33 et Tab[4]=44 ne sont pas ordonnées.



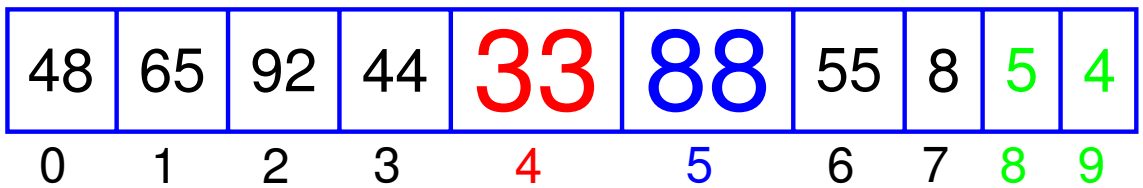
L'échange entre Tab[3]=33 et Tab[4]=44 donne :



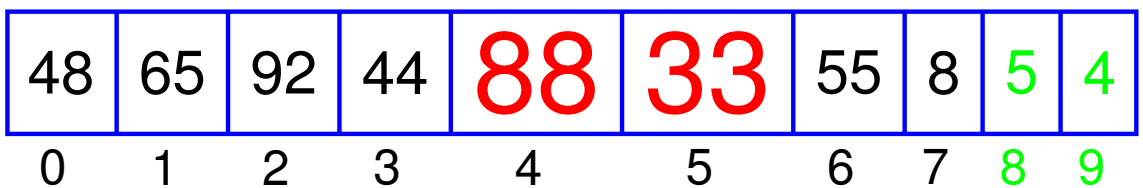
Exemple : tri par bulles

Traitement de l'iteration 3 :

Les deux cases sucessives Tab[4]=33 et Tab[5]=88 ne sont pas ordonnées.



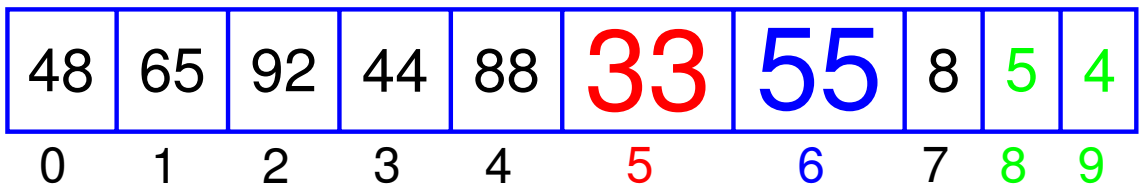
L'échange entre Tab[4]=33 et Tab[5]=88 donne :



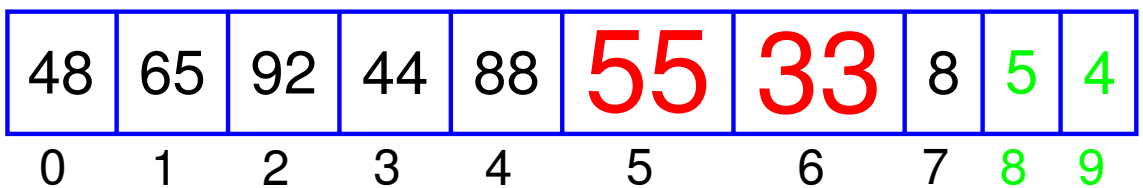
Exemple : tri par bulles

Traitement de l'iteration 3 :

Les deux cases sucessives Tab[5]=33 et Tab[6]=55 ne sont pas ordonnées.



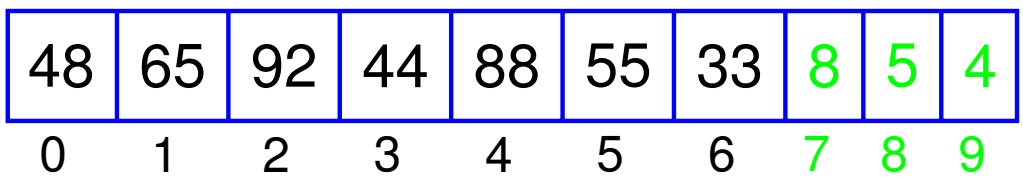
L'échange entre Tab[5]=33 et Tab[6]=55 donne :



Exemple : tri par bulles

Fin de l'itération 3 :

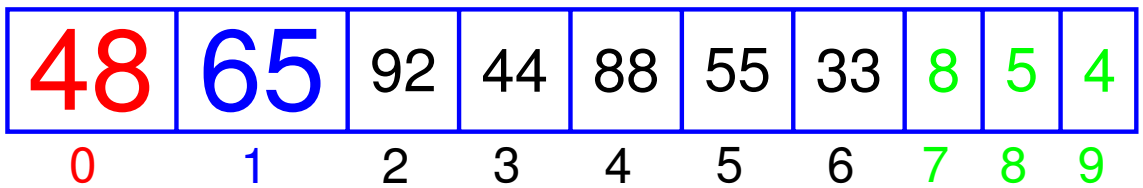
Après l'itération 3, l'élément Tab[7]=8 est à sa bonne place



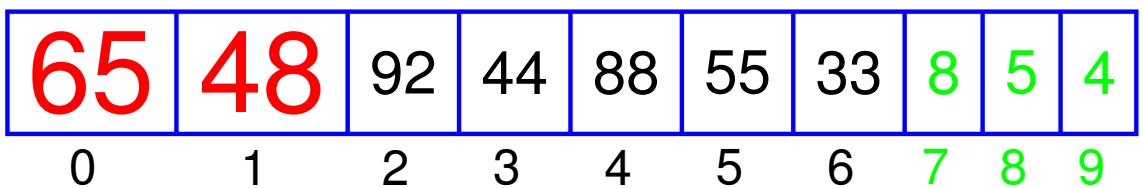
Exemple : tri par bulles

Traitement de l'iteration 4 :

Les deux cases sucessives Tab[0]=48 et Tab[1]=65 ne sont pas ordonnées.



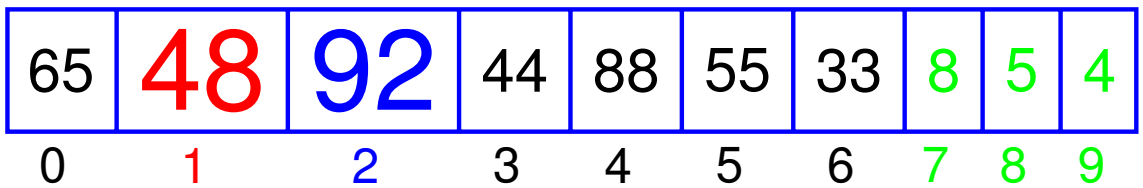
L'échange entre Tab[0]=48 et Tab[1]=65 donne :



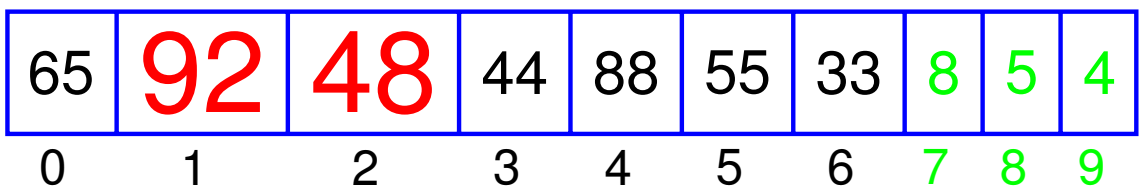
Exemple : tri par bulles

Traitement de l'iteration 4 :

Les deux cases sucessives Tab[1]=48 et Tab[2]=92 ne sont pas ordonnées.



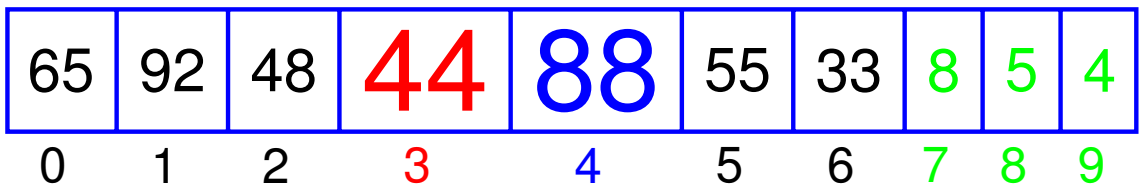
L'échange entre Tab[1]=48 et Tab[2]=92 donne :



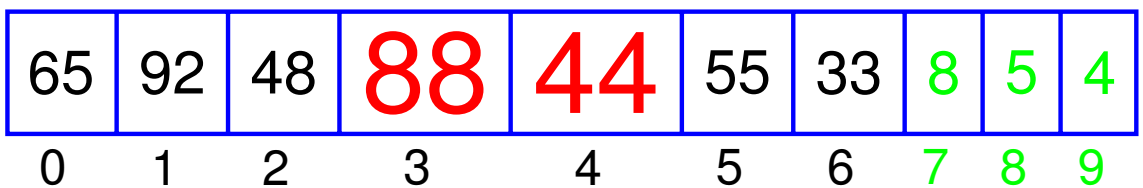
Exemple : tri par bulles

Traitement de l'iteration 4 :

Les deux cases sucessives Tab[3]=44 et Tab[4]=88 ne sont pas ordonnées.



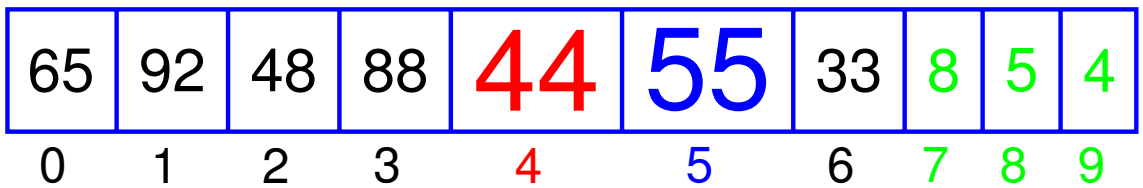
L'échange entre Tab[3]=44 et Tab[4]=88 donne :



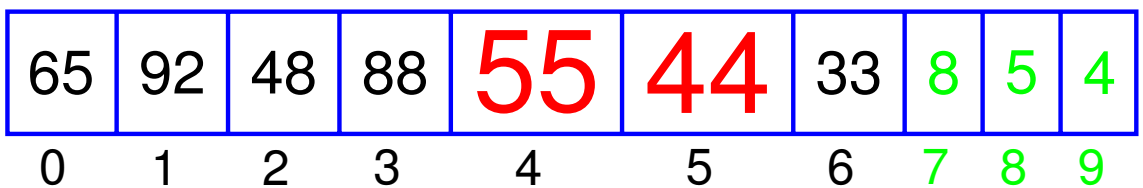
Exemple : tri par bulles

Traitement de l'iteration 4 :

Les deux cases sucessives Tab[4]=44 et Tab[5]=55 ne sont pas ordonnées.



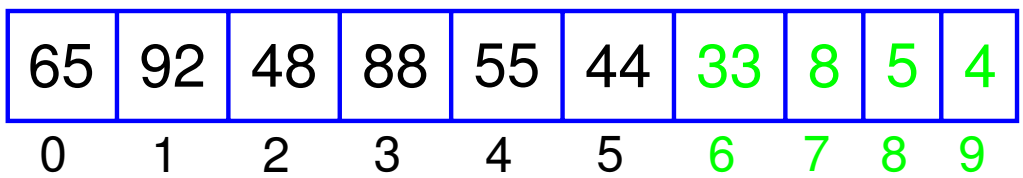
L'échange entre Tab[4]=44 et Tab[5]=55 donne :



Exemple : tri par bulles

Fin de l'itération 4 :

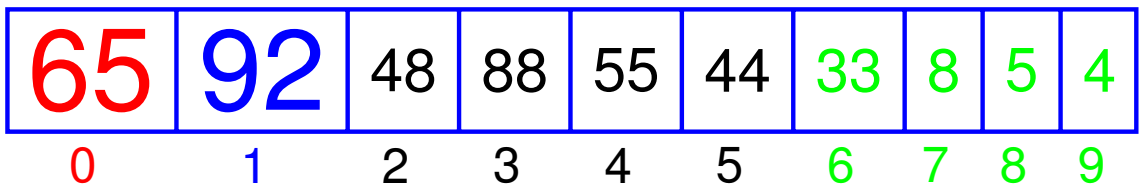
Après l'itération 4, l'élément Tab[6]=33 est à sa bonne place



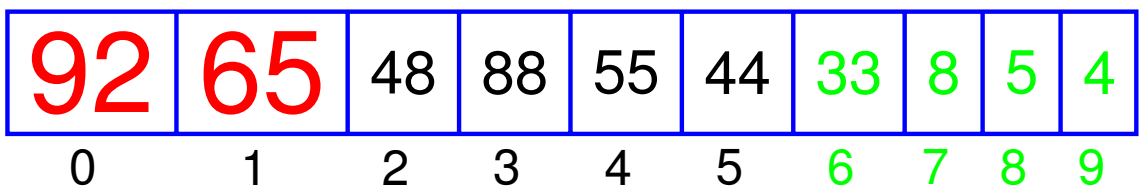
Exemple : tri par bulles

Traitement de l'iteration 5 :

Les deux cases sucessives Tab[0]=65 et Tab[1]=92 ne sont pas ordonnées.



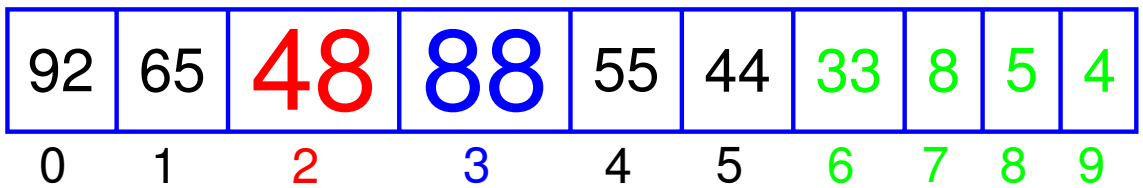
L'échange entre Tab[0]=65 et Tab[1]=92 donne :



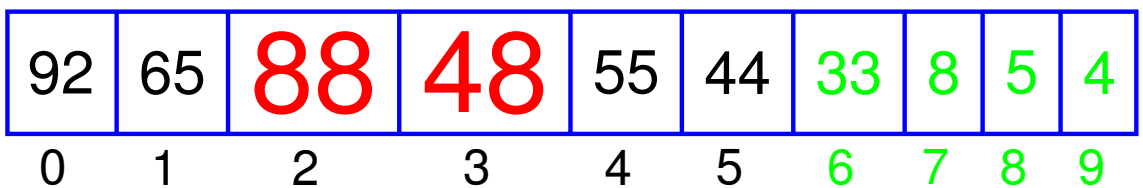
Exemple : tri par bulles

Traitement de l'iteration 5 :

Les deux cases sucessives Tab[2]=48 et Tab[3]=88 ne sont pas ordonnées.



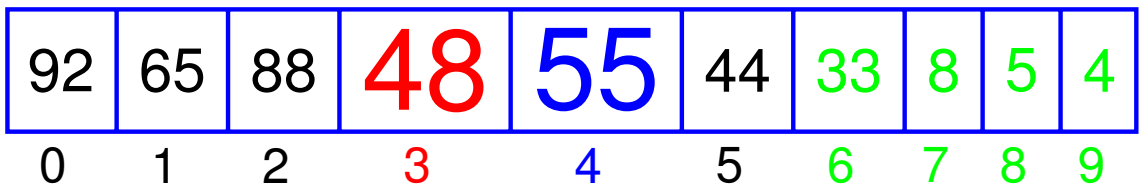
L'échange entre Tab[2]=48 et Tab[3]=88 donne :



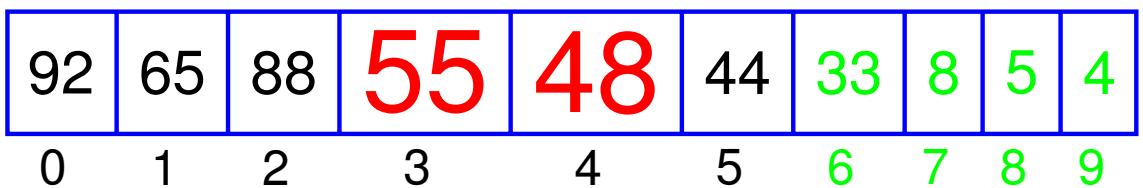
Exemple : tri par bulles

Traitement de l'iteration 5 :

Les deux cases sucessives Tab[3]=48 et Tab[4]=55 ne sont pas ordonnées.



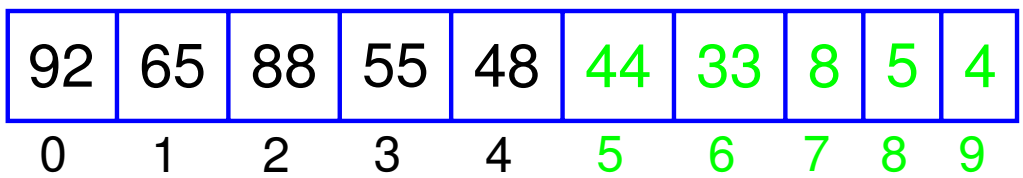
L'échange entre Tab[3]=48 et Tab[4]=55 donne :



Exemple : tri par bulles

Fin de l'itération 5 :

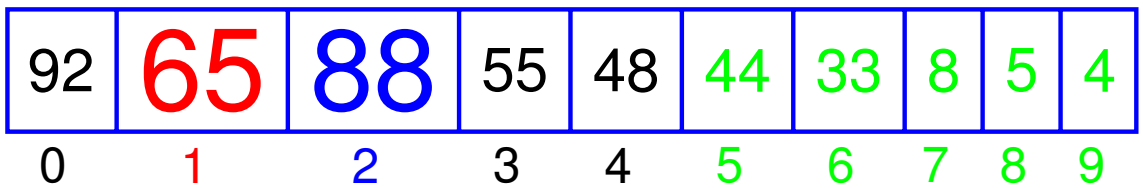
Après l'itération 5, l'élément Tab[5]=44 est à sa bonne place



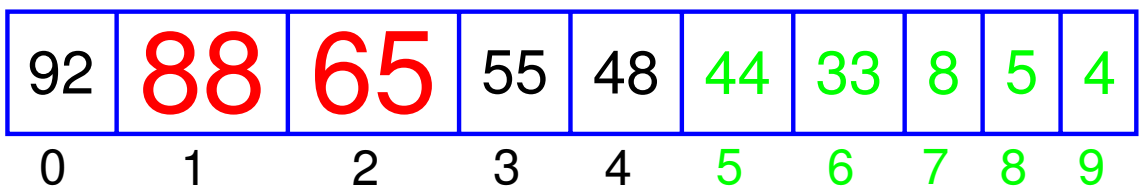
Exemple : tri par bulles

Traitement de l'iteration 6 :

Les deux cases sucessives Tab[1]=65 et Tab[2]=88 ne sont pas ordonnées.



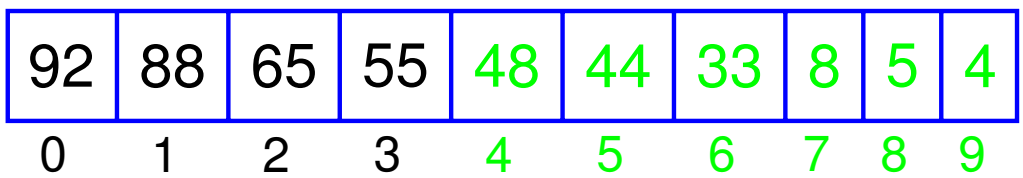
L'échange entre Tab[1]=65 et Tab[2]=88 donne :



Exemple : tri par bulles

Fin de l'itération 6 :

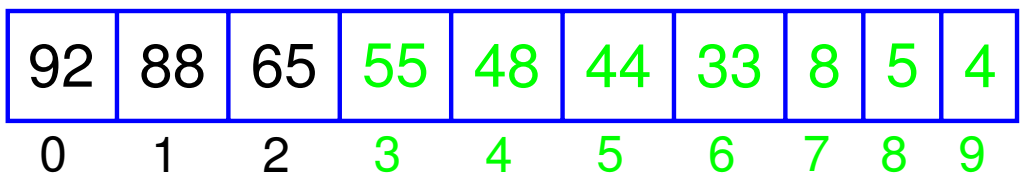
Après l'itération 6, l'élément Tab[4]=48 est à sa bonne place



Exemple : tri par bulles

Fin de l'itération 7 :

Après l'itération 7, l'élément Tab[3]=55 est à sa bonne place



Exemple : tri par bulles

Fin du déroulement :

L'application du tri par bulles a nécessité :

- 24 échanges
- 7 iterations

Après l'application de l'algorithme on obtient :

92	88	65	55	48	44	33	8	5	4
0	1	2	3	4	5	6	7	8	9

Tri à bulles

```
#include <stdio.h>
#define k 10
void echange (int *a, int *b)
{
 int c;
 if (a!=b){
 c=*a;
 *a=*b;
 *b=c;
 }
}
```

Tri à bulles

```
#include <stdio.h>
#define k 10
void echange (int *a, int *b)
{
 int c;
 if (a!=b){
 c=*a;
 *a=*b;
 *b=c;
 }
}
```

Remarque : Le ”if” n’est pas obligatoire.

Tri à bulles

```
void tri_bulles(const int Taille, int Tab[k])
{
    int i,j, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
        permutation=0;
        for(i = 0; i < n; i++)
        {
            if (Tab[i] < Tab[i+1])
            {
                echange (&Tab[i], &Tab[i+1]);
                permutation=1;
            }
        }
        n--;
    }
}
```

Tri à bulles

```
void tri_bulles(const int Taille, int Tab[k])
{
    int i,j, n=Taille-1;
    unsigned char permutation=1;
....
}
```

Remarque : Permutation jouera le rôle d'une variable booléenne.

- La valeur 1 signifie qu'un échange a eu lieu.
- La valeur 0 signifie qu'aucun échange n'a eu lieu. Le tableau est triè !

Tri à bulles

```
void tri_bulles(const int Taille, int Tab[k])
{
    int i,j, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
 ...
    }
}
```

Remarques :

- La boucle s'arrête lorsqu'un échange n'a été effectué !
- La condition (permutation==1) peut-être simplifié en (permutation) ici (pas tout le temps!)

Tri à bulles

```
void tri_bulles(const int Taille, int Tab[k])
{
    int i,j, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
        permutation=0;
        for(i = 0; i < n; i++)
        {
....
        }
        n--;
    }
}
```

Remarques : A chaque parcours on décroit la valeur de la variable *n* car le minimum est bien placée.

Tri à bulles

Une autre écriture (au niveau déclaration):

```
void tri_bulles(const int Taille, int Tab[])
{
    int i, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
        permutation=0;
        for(i = 0; i < n; i++)
        {
            if (Tab[i] < Tab[i+1])
            {
                echange (&Tab[i], &Tab[i+1]);
                permutation=1;
            }
        }
        n--;
    }
}
```

Tri à bulles

Ou encore (toujours au niveau déclaration) :

```
void tri_bulles(const int Taille, int *Tab)
{
    int i, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
        permutation=0;
        for(i = 0; i < n; i++)
        {
            if (Tab[i] < Tab[i+1])
            {
                echange (&Tab[i], &Tab[i+1]);
                permutation=1;
            }
        }
        n--;
    }
}
```

Tri à bulles

Ou encore :

```
void tri_bulles(const int Taille, int *Tab)
{
    int i, n=Taille-1;
    unsigned char permutation=1;
    while (permutation==1)
    {
        permutation=0;
        for(i = 0; i < n; i++)
        {
            if (Tab[i] < Tab[i+1])
            {
                echange (Tab+i, Tab+i+1);
                permutation=1;
            }
        }
        n--;
    }
}
```

Tri à bulles

```
void imprimer(const int Taille, int Tab[])
{
    int i;
    printf("\n");
    for(i = 0; i < Taille; i++) printf ("%d \t", Tab[i]);
    printf("\n");
}

int main (void)
{
 void echange (int *a,int *b);
     void tri_bulles(const int Taille, int Tab[]);
 int tableau [k]={48, 33, 8, 4, 65, 92, 44, 88, 55, 5};
     tri_bulles(k,tableau);
     printf ("Après tri : \n");
     imprimer(k,tableau);
     return(0);
}
```

Tri à bulles

On peut clairement améliorer l'algorithme de tri. Le but ici est d'illustrer l'utilisation des tableaux.