Algorithmique et complexité

Polytech Paris-Saclay, PEIP 2, Informatique 3

Thibaut Benjamin

7 Novembre 2025

Amphi 1

Équipe d'enseignement

- Amphi
  Thibaut Benjamin thibaut.benjamin@universite-paris-saclay.fr

- TD/TP
  Thibaut Benjamin
  Melissa Larbi
  Ali Sahili
  Julien Rauch

Nous sommes là pour vous, n'hésitez pas à venir vers nous lorsque vous avez des questions.

Objectif du cours

- Renforcer les compétences en programmation C++
- Comprendre les bases théoriques le la complexité des algorithmes
- Développer une connaissance des algorithmes classique
- Apprendre à analyser la complexité d'un algorithme

Organisation du cours

- 4 Cours en Amphis
- 4 Séances de TD
- 4 Séances de TP

Évaluations

- Examen de 2h Vendredi 5 Décembre
  Tous documents papiers autorisés

- 1 Séance de mini-contrôle noté en TD

Ressource en ligne

Une adresse à ajouter à vos favoris ! (cours, TD, TP, etc...)

`https://thibautbenjamin.github.io/cours-polytech-peip2-algo`

Ressource en ligne

Une adresse à ajouter à vos favoris ! (cours, TD, TP, etc...)

`https://thibautbenjamin.github.io/cours-polytech-peip2-algo`

Si vous le souhaitez, contribuez au document de cours !

Venez me voir si vous ne savez pas comment vous y prendre.

Séance du jour

- Comprendre ce que la complexité d'un algorithme représente
- Réviser les bases de la programmation en C++
- Aborder la notion de tableau en C++

La complexité des algorithmes

Qu'est-ce qu'un algorithme ?

Definition
Un algorithme est une séquence d'intstructions élémentaires permettant de réaliser une tache.

Pour ce cours, un algorithme est un programme en C++

Un exemple : Les algorithme de tri

- On dispose d'un paquet de données que l'on peut comparer entre elles pour savoir si laquelle est la plus grande.
- On souhaite trier ces données par ordre croissant
- De nombreux algorithmes pour faire cela (voir le prochain amphi)
- Une visualisation utile : `https://mszula.github.io/visual-sorting/`

Visualisation des algorithmes de tri

- rendez-vous sur : `https://mszula.github.io/visual-sorting/`

- regardez les tris suivants
  - Tri par sélection (selection sort)
  - Tri par insertion (insertion sort)
  - Tri à bulles (bubble sort)
  - Tri fusion (merge sort)
  - Tri rapide (quicksort)

Qu'est-ce que la complexité d'un algorithme ?

Definition
La complexité d'un algorithme est le nombre d'opérations que cet algorithme effectue en fonction de la taille de ses entrées.

Temps d'exécution : De manière générale, plus un algorithme a une complexité élevée, et plus il mettra de temps à s'exécuter.

Complexité asymptotique

- Sur des entrées de petites tailles, avec un ordinateur moderne, les algorithmes s'exécuteront tous très rapidement, peu importe leur complexité.

- C'est pourquoi on regarde la complexité quand les entrées deviennent très grandes[1]. On parle de complexité asymptotique.

- On cherchera en général à calculer une approximation de la complexité.
  Si les entrées doublent de taille, est-ce qu'on va devoir attendre quelques secondes de plus, ou quelques années de plus ?

1. En mathématiques, on dit que leur taille tend vers l'infini

Rappels de C++

Les fichiers C++

```
#include <iostream>      //Ces deux lignes seront omises
using namespace std;     //dans la suite

int main (){
  cout << "Bonjour tout le monde!" ;
  return 0;
}
```

Les fichiers C++

```
#include <iostream>      //Ces deux lignes seront omises
using namespace std;     //dans la suite

int main (){
  cout << "Bonjour tout le monde!" ;
  return 0;
}
```

```
Bonjour tout le monde!
```

Déclaration et affectation des variables

```
int main (){
  int x;                  // Declaration
  x = 8;                  // Affectation
  int y = 15;             // Declaration et affectation
  x = y;                  // Affectation
  cout << " x = " << x << " et y = " << y;
  return 0;
}
```

Déclaration et affectation des variables

```
int main (){
  int x;                  // Declaration
  x = 8;                  // Affectation
  int y = 15;             // Declaration et affectation
  x = y;                  // Affectation
  cout << " x = " << x << " et y = " << y;
  return 0;
}
```

```
x = 15 et y = 15
```

Exemple : échange de la valeur

```
int main (){
  int x = 0, y = 1;
  // On veut echanger la valeur de x et y
  int z = x; x = y; y = z;
  cout << "x = " << x << " et y = " << y;
  return 0;
}
```

Exemple : échange de la valeur

```
int main (){
  int x = 0, y = 1;
  // On veut echanger la valeur de x et y
  int z = x; x = y; y = z;
  cout << "x = " << x << " et y = " << y;
  return 0;
}
```

```
x = 1 et y = 0
```

Les types de données

- `int` : Les nombres entiers positifs ou négatifs [2]
- `char` : Les caractères
- `bool` : Les booléens, c'est à dire `true` ou `false`.

2. Techniquement, compris entre deux bornes

Définir des fonctions

```
int fahren (int x){
  return 1.8 * x + 32;
}

int main (){
  int x = 30;
  cout << x << " Celsius = " << fahren(x) << " Fahrenheit";
  return 0;
}
```

Définir des fonctions

```
int fahren (int x){
  return 1.8 * x + 32;
}

int main (){
  int x = 30;
  cout << x << " Celsius = " << fahren(x) << " Fahrenheit";
  return 0;
}
```

```
30 Celsius = 86 Fahrenheit
```

Passage par valeurs

```
void echanger (int x, int y){
  int z = x; x = y; y = z;
}

int main (){
  int x = 0, y = 1;
  echanger(x,y);
  cout <<  "x = " << x << " et y = " << y;
  return 0;
}
```

Passage par valeurs

```
void echanger (int x, int y){
  int z = x; x = y; y = z;
}

int main (){
  int x = 0, y = 1;
  echanger(x,y);
  cout <<  "x = " << x << " et y = " << y;
  return 0;
}
```

```
x = 0 et y = 1
```

Passage par valeurs : Explication

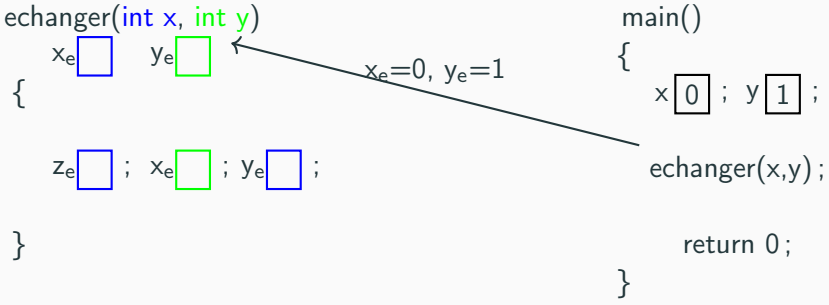
```
echanger(int x, int y)
    xe□    ye□
{

    ze□ ;  xe□ ; ye□ ;

}
```

Passage par valeurs : Explication

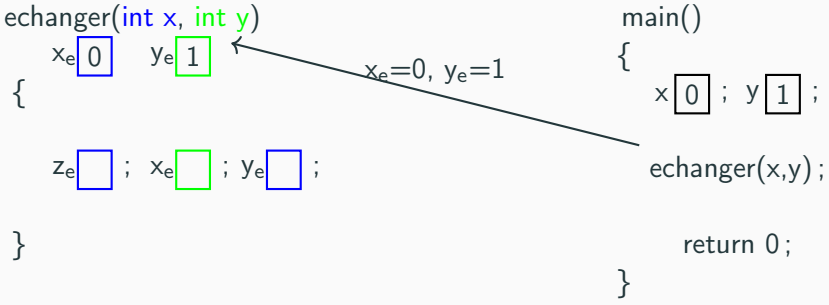
```
echanger(int x, int y)
    xe      ye
{

    ze  ;  xe  ;  ye  ;

}
```

```
main()
{
   x 0 ;  y 1 ;

   echanger(x,y) ;

      return 0 ;
}
```

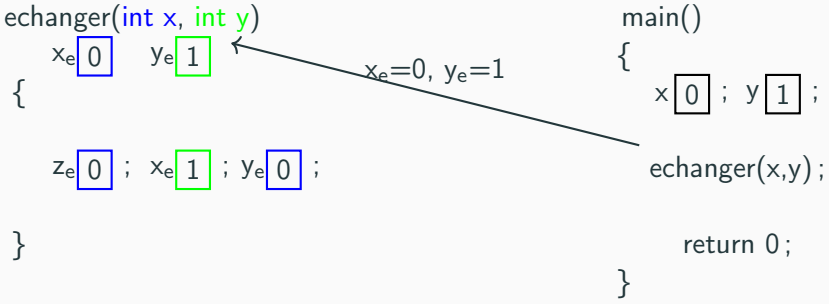
Passage par valeurs : Explication



Passage par valeurs : Explication



Passage par valeurs : Explication



Passage par référence

```
void echanger (int &x, int &y){
  int z = x; x = y; y = z;
}

int main (){
  int x = 0, y = 1;
  echanger(x,y);
  cout <<  "x = " << x << " et y = " << y;
  return 0;
}
```

Passage par référence

```
void echanger (int &x, int &y){
  int z = x; x = y; y = z;
}

int main (){
  int x = 0, y = 1;
  echanger(x,y);
  cout <<  "x = " << x << " et y = " << y;
  return 0;
}
```

```
x = 1 et y = 0
```

Les conditions

```
int main (){
  bool condition = false;
  if (condition){
    cout <<  "la condition est vraie";
  } else {
    cout << "la condition est fausse";
  }
  return 0;
}
```

Les conditions

```
int main (){
  bool condition = false;
  if (condition){
    cout <<  "la condition est vraie";
  } else {
    cout << "la condition est fausse";
  }
  return 0;
}
```

```
la condition est fausse
```

Les boucles

```
int main (){
  for (int i = 0; i < 5; i++){
    cout <<  "boucle " << i << ", ";
  }
  return 0;
}
```

Les boucles

```
int main (){
  for (int i = 0; i < 5; i++){
    cout <<  "boucle " << i << ", ";
  }
  return 0;
}
```

```
boucle 0, boucle 1, boucle 2, boucle 3, boucle 4,
```

Les tableaux

Les tableaux : une structure de donnée

- Les structures de données permettent de stocker des données de manière organisée. Il est important de choisir une structure de donnée adaptée aux opérations que l'on va avoir besoin de réaliser.

- Les tableaux permettent de stocker une collection ordonnée d'objets de même type.

- **Exemple** : un tableau d'`int` :

0	1	2	3	4
10	-7	42	0	10

  Attention : On commence à compter les positions à partir de 0.

Avec les tableaux on doit

- Garder la même taille tout au long du programme.
- Déclarer la taille du tableau au moment de l'initialisation.
- A la place de changer de taille, il faut déclarer un nouveau tableau avec la nouvelle taille et copier toutes le valeur de l'ancien tableau dans le nouveau.
  Déconseillé !

Avec les tableaux on peut

- Accéder à un élément stocké à n'importe quelle position dans le tableau.
- Changer la valeur contenue à n'importe quelle position dans le tableau.
- **Attention !** Dans le tableau signifie dans les bornes du tableau. Si un tableau est de longueur n, cela veut dire n'importe quelle position entre 0 et $n - 1$.

Déclarer un tableau en C++

```
int main (){
  int t[5]; // Declaration d'un tableau t de taille 5
  return 0;
}
```

Initialisation d'un tableau

```
int main (){
  int t[5] = {10, -7, 42, 0, 10};
  // Seulement pour l'initialisation
  return 0;
}
```

Accès aux éléments d'un tableau

```
void afficher (int n, int t[]){
 for(int i = 0; i < n; i++){
    cout << i << " : " << t[i] << ", ";
  }
}

int main (){
  int t[5] = {10, -7, 42, 0, 10};
  afficher(5, t);
  return 0;
}
```

Accès aux éléments d'un tableau

```
void afficher (int n, int t[]){
 for(int i = 0; i < n; i++){
    cout << i << " : " << t[i] << ", ";
  }
}

int main (){
  int t[5] = {10, -7, 42, 0, 10};
  afficher(5, t);
  return 0;
}
```

```
0 : 10, 1 : -7, 2 : 42, 3 : 0, 4 : 10,
```

Modification des éléments d'un tableau

```
int main (){
  int t[5] = {10, -7, 42, 0, 10};
  print(5, t);
  return 0;
}
```

Modification des éléments d'un tableau

```
int main (){
  int t[5] = {10, -7, 42, 0, 10};
  print(5, t);
  return 0;
}
```

```
0 : 10, 1 : -7, 2 : 42, 3 : 0, 4 : 10,
```

Passer un tableau en argument d'une fonction

```
void echangerPrems (int a[]){
  int z = a[0]; a[0] = a[1]; a[1] = z;
}

int main (){
  int a[4] = {0, 1, 2, 3};
  echangerPrems(a);
  afficher(4, a);
  return 0;
}
```

Passer un tableau en argument d'une fonction

```
void echangerPrems (int a[]){
  int z = a[0]; a[0] = a[1]; a[1] = z;
}

int main (){
  int a[4] = {0, 1, 2, 3};
  echangerPrems(a);
  afficher(4, a);
  return 0;
}
```

```
0 : 1, 1 : 0, 2 : 2, 3 : 3,
```

Passer un tableau en argument d'une fonction : Explications

La valeur d'un tableau est l'adresse de la zone en mémoire contenant ce tableau

```
echangerPrems(int a[])
      a_e
 {

      int z = a[0] ; a[0] = a[1] ; a[1] = z ;

 }
```

Passer un tableau en argument d'une fonction : Explications

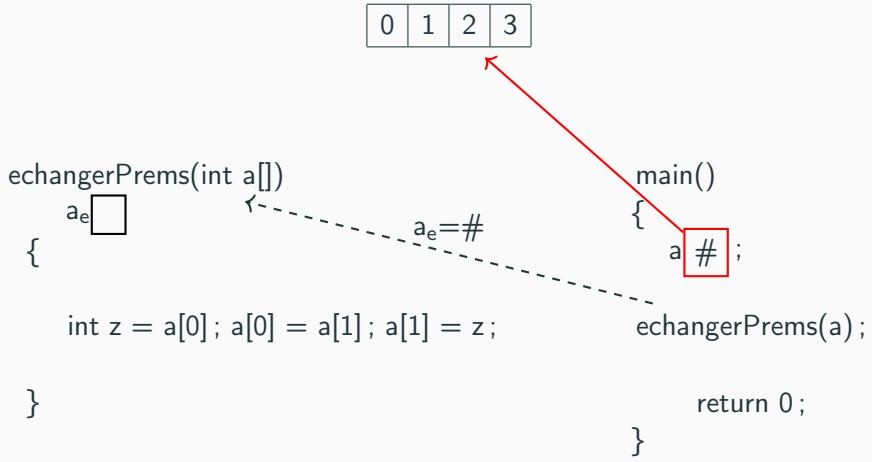
La valeur d'un tableau est l'adresse de la zone en mémoire contenant ce tableau


```
echangerPrems(int a[])
    ae
{

    int z = a[0]; a[0] = a[1]; a[1] = z;

}
```

```
main()
{
    a #;

echangerPrems(a);

    return 0;
}
```

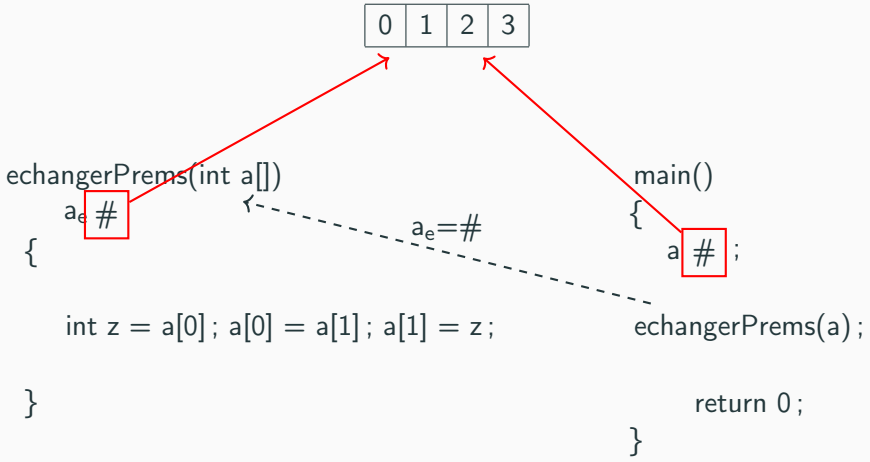
Passer un tableau en argument d'une fonction : Explications

La valeur d'un tableau est l'adresse de la zone en mémoire contenant ce tableau



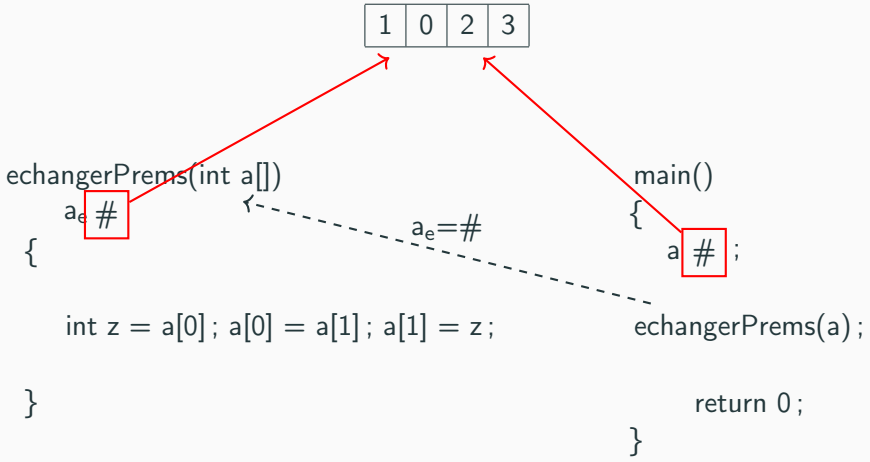
Passer un tableau en argument d'une fonction : Explications

La valeur d'un tableau est l'adresse de la zone en mémoire contenant ce tableau



Passer un tableau en argument d'une fonction : Explications

La valeur d'un tableau est l'adresse de la zone en mémoire contenant ce tableau



Quand utiliser les tableaux

- Il existe d'autres structures de données en C++pour reprérenter des collections ordonnées d'éléments.
  - Les listes
  - Les vecteurs
- Ces structures permettent de faire d'autres opérations de manière primitive. Par exemple, dans une liste il est facile d'ajouter un élément, mais accéder au élément en dernière position coute cher.
- Il faut donc choisir la bonne structure en fonction des opérations que l'on veut faire dessus. On choisira les tableaux lorsqu'on a une taille fixe et qu'on veut accéder rapidement à n'importe quelle position.

Récapitulatif

Algorithmes et complexité

Un algorithme est un programme, sa complexité est le nombre d'opération qu'il fait, plus la complexité d'un algorithme est élevée, plus il prendra de temps à s'exécuter.

- Variables, conditions et boucles
- Appels de fonction et passage par valeur

Les tableaux en C++

- Permettent de stocker des collections ordonnées de nombres.
- Modifiés en place lorsqu'ils sont passés comme arguments aux fonctions.
- Taille fixe et accès à n'importe quelle position très rapide.