Le Mélange de Tableaux

(Array Shuffling)

Joseph AZAR
IUT Nord Franche-Comté
joseph.azar@univ-fcomte.fr

💡 Astuce : Appuyez sur `0` ou `Échap` pour la vue d'ensemble

🎯 Pourquoi mélanger un tableau ?

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

- 🎴 Jeux de cartes (poker, solitaire...)

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

- 🎴 Jeux de cartes (poker, solitaire...)
- 🎰 Loteries et tirages au sort

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

- 🎴 Jeux de cartes (poker, solitaire...)
- 🎰 Loteries et tirages au sort
- 🎵 Lecture aléatoire de playlists

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

- 🎴 Jeux de cartes (poker, solitaire...)
- 🎰 Loteries et tirages au sort
- 🎵 Lecture aléatoire de playlists
- 🎮 Génération de niveaux aléatoires dans les jeux

🎯 Pourquoi mélanger un tableau ?

> **Définition :** Mélanger (shuffle) = réorganiser les éléments d'un tableau de manière aléatoire

Applications réelles :

- 🎴 Jeux de cartes (poker, solitaire...)
- 🎰 Loteries et tirages au sort
- 🎵 Lecture aléatoire de playlists
- 🎮 Génération de niveaux aléatoires dans les jeux
- 📊 Tests statistiques et échantillonnage

🎰 Exemple : Loterie à 10 numéros

Tableau initial : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1	2	3	4	5	6	7	8	9	10

🎰 Exemple : Loterie à 10 numéros

Tableau initial : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1	2	3	4	5	6	7	8	9	10

🎲 **Après mélange :**

7	2	9	1	5	10	3	6	4	8

🎰 Exemple : Loterie à 10 numéros

Tableau initial : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1	2	3	4	5	6	7	8	9	10

🎲 **Après mélange :**

7	2	9	1	5	10	3	6	4	8

⚠️ **Important :** Chaque élément doit avoir la même probabilité d'être à n'importe quelle position !

Approche 1 : Méthode avec tableau auxiliaire

(La méthode "intuitive" mais coûteuse)

💡 L'idée de base

Étapes :

💡 L'idée de base

Étapes :

1. 1️⃣ Créer un tableau auxiliaire vide

💡 L'idée de base

Étapes :

1. 1️⃣ Créer un tableau auxiliaire vide
2. 2️⃣ Tant que le tableau original n'est pas vide :
   - Choisir un index aléatoire
   - Copier l'élément dans le tableau auxiliaire
   - Supprimer l'élément du tableau original

💡 L'idée de base

Étapes :

1. 1️⃣ Créer un tableau auxiliaire vide
2. 2️⃣ Tant que le tableau original n'est pas vide :
   - Choisir un index aléatoire
   - Copier l'élément dans le tableau auxiliaire
   - Supprimer l'élément du tableau original
3. 3️⃣ Retourner le tableau auxiliaire

💡 L'idée de base

Étapes :

1. 1️⃣ Créer un tableau auxiliaire vide
2. 2️⃣ Tant que le tableau original n'est pas vide :
   - Choisir un index aléatoire
   - Copier l'élément dans le tableau auxiliaire
   - Supprimer l'élément du tableau original
3. 3️⃣ Retourner le tableau auxiliaire

Avantage :
✅ Facile à comprendre
✅ Produit un mélange équitable

Inconvénient :
❌ Nécessite un tableau supplémentaire
❌ Suppression = opération coûteuse !

⚠️ Mais... comment supprimer un élément ?

Problème : En Java, un tableau a une taille **fixe** !
On ne peut pas vraiment "supprimer" un élément, on doit créer un **nouveau tableau plus petit**.

⚠️ Mais... comment supprimer un élément ?

> **Problème :** En Java, un tableau a une taille **fixe** !
> On ne peut pas vraiment "supprimer" un élément, on doit créer un **nouveau tableau plus petit**.

📋 Étapes pour supprimer l'élément à l'index 2 :

⚠️ Mais... comment supprimer un élément ?

> **Problème :** En Java, un tableau a une taille **fixe** !
> On ne peut pas vraiment "supprimer" un élément, on doit créer un **nouveau tableau plus petit**.

📋 Étapes pour supprimer l'élément à l'index 2 :

1. Tableau initial de taille N = 5

↑ On veut supprimer "C" à l'index 2

⚠️ Mais... comment supprimer un élément ?

Problème : En Java, un tableau a une taille **fixe** !
On ne peut pas vraiment "supprimer" un élément, on doit créer un **nouveau tableau plus petit**.

📋 Étapes pour supprimer l'élément à l'index 2 :

1. Tableau initial de taille N = 5

↑ On veut supprimer "C" à l'index 2

2. Créer un nouveau tableau de taille N-1 = 4

⚠️ Mais... comment supprimer un élément ?

Problème : En Java, un tableau a une taille **fixe** !
On ne peut pas vraiment "supprimer" un élément, on doit créer un **nouveau tableau plus petit**.

📋 Étapes pour supprimer l'élément à l'index 2 :

1. Tableau initial de taille N = 5

↑ On veut supprimer "C" à l'index 2

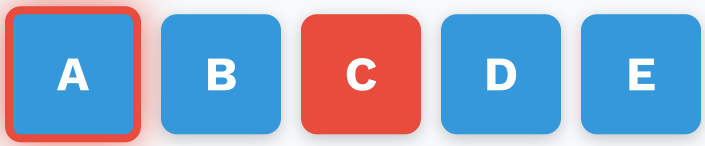
2. Créer un nouveau tableau de taille N-1 = 4

3. Copier tous les éléments SAUF celui à supprimer


A
B
D
E

🎬 Animation : Suppression d'un élément

Tableau original (taille = 5) :



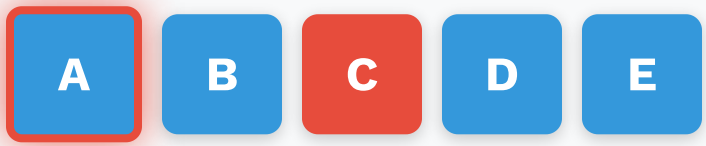
Nouveau tableau (taille = 4) :

▶️ Étape de copie 🔄 Recommencer

🎬 Prêt à copier les éléments (sauf "C" à l'index 2)
Cliquez sur le bouton pour copier élément par élément

🎬 Animation : Suppression d'un élément

Tableau original (taille = 5) :



Nouveau tableau (taille = 4) :

▶️ Étape de copie 🔄 Recommencer

🎬 Prêt à copier les éléments (sauf "C" à l'index 2)
Cliquez sur le bouton pour copier élément par élément

⚠️ **Complexité :** Copier N éléments = **O(N)**

Algorithme : Supprimer un élément

```
ALGORITHME supprimerElement(arr, N, indexASupprimer)
DEBUT
    SI indexASupprimer < 0 OU indexASupprimer >= N ALORS RETOURNER arr FIN SI
    nouveauTableau ← créer tableau de taille (N - 1)
    k ← 0
    POUR i DE 0 À N-1 FAIRE
        SI i ≠ indexASupprimer ALORS
            nouveauTableau[k] ← arr[i]
            k ← k + 1
        FIN SI
    FIN POUR
    RETOURNER nouveauTableau
FIN
```

Algorithme : Supprimer un élément

```
ALGORITHME supprimerElement(arr, N, indexASupprimer)
DEBUT
    SI indexASupprimer < 0 OU indexASupprimer >= N ALORS RETOURNER arr FIN SI
    nouveauTableau ← créer tableau de taille (N - 1)
    k ← 0
    POUR i DE 0 À N-1 FAIRE
        SI i ≠ indexASupprimer ALORS
            nouveauTableau[k] ← arr[i]
            k ← k + 1
        FIN SI
    FIN POUR
    RETOURNER nouveauTableau
FIN
```

Complexité : O(N)

📝 Algorithme : Supprimer un élément

```
ALGORITHME supprimerElement(arr, N, indexASupprimer)
DEBUT
    SI indexASupprimer < 0 OU indexASupprimer >= N ALORS RETOURNER arr FIN SI
    nouveauTableau ← créer tableau de taille (N - 1)
    k ← 0
    POUR i DE 0 À N-1 FAIRE
        SI i ≠ indexASupprimer ALORS
            nouveauTableau[k] ← arr[i]
            k ← k + 1
        FIN SI
    FIN POUR
    RETOURNER nouveauTableau
FIN
```

Complexité : O(N)

⚠️ **Attention :** Cette opération nécessite de **parcourir tout le tableau** (boucle FOR) → coûteux !

🎬 Retour au shuffle : Démonstration complète

💡 Maintenant qu'on sait comment supprimer un élément, voyons l'algorithme complet en action !
Regardez bien : le tableau original **diminue de taille** à chaque étape.

Tableau original :

Taille actuelle : 6

1	2	3	4	5	6

Tableau mélangé :

Taille actuelle : 0

?	?	?	?	?	?

▶️ Étape suivante

🔄 Recommencer

📊 Étape 0/6 - **Le tableau original a maintenant 6 élément(s)**

Algorithme : Shuffle avec tableau auxiliaire

```
ALGORITHME shuffleAvecAuxiliaire(arr, N)
DEBUT
    tableauMelange ← créer tableau de taille N
    tailleActuelle ← N
    indexMelange ← 0
    TANT QUE tailleActuelle ≠ 0 FAIRE
        indexAleatoire ← ALEATOIRE(0, tailleActuelle - 1)
        tableauMelange[indexMelange] ← arr[indexAleatoire]
        arr ← supprimerElement(arr, tailleActuelle, indexAleatoire)
        tailleActuelle ← tailleActuelle - 1
        indexMelange ← indexMelange + 1
    FIN TANT QUE
    RETOURNER tableauMelange
FIN
```

 Algorithme : Shuffle avec tableau auxiliaire

```
ALGORITHME shuffleAvecAuxiliaire(arr, N)
DEBUT
    tableauMelange ← créer tableau de taille N
    tailleActuelle ← N
    indexMelange ← 0
    TANT QUE tailleActuelle ≠ 0 FAIRE
        indexAleatoire ← ALEATOIRE(0, tailleActuelle - 1)
        tableauMelange[indexMelange] ← arr[indexAleatoire]
        arr ← supprimerElement(arr, tailleActuelle, indexAleatoire)
        tailleActuelle ← tailleActuelle - 1
        indexMelange ← indexMelange + 1
    FIN TANT QUE
    RETOURNER tableauMelange
FIN
```

Complexité : O(n²) 😢

🔍 Traçage des variables : pas à pas

💡 Suivons l'évolution des variables à chaque itération pour comprendre l'algorithme !

Variables actuelles :

tailleActuelle = **4** | indexMelange = **0** | indexAleatoire = **?**

Tableau original (taille = 4) :

1	2	3	4

Tableau mélangé :

?	?	?	?

▶️ Étape suivante 🔄 Recommencer

🎬 **État initial**

TANT QUE tailleActuelle (4) ≠ 0 → On continue !

Pourquoi O(n²) ?

📊 Pourquoi O(n²) ?

Boucle principale : n itérations

📊 Pourquoi O(n²) ?

Boucle principale : n itérations

À chaque itération :

- Choisir un index aléatoire : O(1)
- Copier l'élément : O(1)
- Supprimer l'élément : O(n) ⚠️

📊 Pourquoi O(n²) ?

Boucle principale : n itérations

À chaque itération :

- Choisir un index aléatoire : O(1)
- Copier l'élément : O(1)
- Supprimer l'élément : O(n) ⚠️

Total : n × O(n) = $O(n^2)$

📊 Pourquoi O(n²) ?

Boucle principale : n itérations

À chaque itération :

- Choisir un index aléatoire : O(1)
- Copier l'élément : O(1)
- Supprimer l'élément : O(n) ⚠️

Total : n × O(n) = $O(n^2)$

⏱️ Pour un tableau de 1000 éléments : **~1 million d'opérations !**

🎯 Approche 2 : Fisher-Yates

(L'algorithme optimal !)

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

💡 **L'idée géniale :**

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

💡 **L'idée géniale :**

- Pas besoin de tableau auxiliaire !

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

💡 **L'idée géniale :**

- Pas besoin de tableau auxiliaire !
- Pas besoin de suppression !

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

💡 **L'idée géniale :**

- Pas besoin de tableau auxiliaire !
- Pas besoin de suppression !
- On mélange **sur place** (in-place)

✨ L'algorithme de Fisher-Yates

> **Inventé en 1938** par Ronald Fisher et Frank Yates
> Aussi appelé **Knuth shuffle** (popularisé par Donald Knuth en 1964)

💡 **L'idée géniale :**

- Pas besoin de tableau auxiliaire !
- Pas besoin de suppression !
- On mélange **sur place** (in-place)
- On parcourt le tableau **une seule fois**

🔄 Comment ça marche ?

Algorithme :

🔄 Comment ça marche ?

Algorithme :

1. Parcourir le tableau de la **fin vers le début**

🔄 Comment ça marche ?

Algorithme :

1. Parcourir le tableau de la **fin vers le début**
2. Pour chaque position `i` :
   - Choisir un index aléatoire `j` entre `0` et `i`
   - Échanger `arr[i]` avec `arr[j]`

🔄 Comment ça marche ?

Algorithme :

1. Parcourir le tableau de la **fin vers le début**
2. Pour chaque position `i` :
   - Choisir un index aléatoire `j` entre `0` et `i`
   - Échanger `arr[i]` avec `arr[j]`

✅ **Avantages :**

- Pas de mémoire supplémentaire
- Une seule passe
- Mélange équitable garanti
- Simple à implémenter

Complexité : O(n) 🎉

🎬 Fisher-Yates : Animation pas à pas

Tableau en cours de mélange :

1	2	3	4	5	6	7	8

▶️ Étape suivante 🔄 Recommencer

Index i = 7

Cliquez pour choisir un index aléatoire j entre 0 et 7

🎬 Fisher-Yates : Animation pas à pas

Tableau en cours de mélange :

1	2	3	4	5	6	7	8

▶️ Étape suivante 🔄 Recommencer

Index i = 7
Cliquez pour choisir un index aléatoire j entre 0 et 7

🔑 **Observation :** La partie droite (déjà mélangée) ne change plus !

Algorithme : Fisher-Yates

```
ALGORITHME FisherYates(arr, N)
DEBUT
    POUR i DE (N - 1) À 1 FAIRE
        j ← ALEATOIRE(0, i)
        temp ← arr[i]
        arr[i] ← arr[j]
        arr[j] ← temp
    FIN POUR
    RETOURNER arr
FIN
```

 Algorithme : Fisher-Yates

```
ALGORITHME FisherYates(arr, N)
DEBUT
    POUR i DE (N - 1) À 1 FAIRE
        j ← ALEATOIRE(0, i)
        temp ← arr[i]
        arr[i] ← arr[j]
        arr[j] ← temp
    FIN POUR
    RETOURNER arr
FIN
```

✨ **Simple, élégant, et optimal !**

📝 Algorithme : Fisher-Yates

```
ALGORITHME FisherYates(arr, N)
DEBUT
    POUR i DE (N - 1) À 1 FAIRE
        j ← ALEATOIRE(0, i)
        temp ← arr[i]
        arr[i] ← arr[j]
        arr[j] ← temp
    FIN POUR
    RETOURNER arr
FIN
```

✨ **Simple, élégant, et optimal !**

🔑 **Points clés :**

- Une seule boucle (pas de boucle imbriquée)
- Échange direct (pas de suppression)
- Pas de tableau auxiliaire (mélange sur place)

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates
Complexité temps	$O(n^2)$	$O(n)$

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates
Complexité temps	$O(n^2)$	$O(n)$
Complexité mémoire	$O(n)$	$O(1)$

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates
Complexité temps	$O(n^2)$	$O(n)$
Complexité mémoire	$O(n)$	$O(1)$
Facilité de compréhension	Facile	Moyenne

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates
Complexité temps	$O(n^2)$	O(n)
Complexité mémoire	O(n)	O(1)
Facilité de compréhension	Facile	Moyenne
Utilisation en production	Non	Oui !

⚖️ Comparaison des deux méthodes

Critère	Tableau auxiliaire	Fisher-Yates
Complexité temps	$O(n^2)$	O(n)
Complexité mémoire	O(n)	O(1)
Facilité de compréhension	Facile	Moyenne
Utilisation en production	Non	Oui !

🏆 **Fisher-Yates est utilisé dans toutes les bibliothèques standard**

(Java Collections.shuffle(), Python random.shuffle(), etc.)

🎰 Application pratique : Simulateur de loterie

Simulation d'une loterie à 42 numéros (comme l'exemple du TP)

🎲 Tirer 7 numéros

🔄 Nouvelle grille

Cliquez sur le bouton pour tirer vos numéros !

🎰 Application pratique : Simulateur de loterie

Simulation d'une loterie à 42 numéros (comme l'exemple du TP)

🎲 Tirer 7 numéros

🔄 Nouvelle grille

Cliquez sur le bouton pour tirer vos numéros !

💡 **Astuce :** Il suffit de mélanger le tableau puis de prendre les 7 premiers éléments !

Algorithme complet : Simulation de loterie

```
ALGORITHME PRINCIPAL SimulationLoterie
DEBUT
    N ← 42
    loterie ← créer tableau de taille N
    POUR i DE 0 À N-1 FAIRE
        loterie[i] ← i + 1
    FIN POUR
    loterie ← FisherYates(loterie, N)
    mesNumeros ← créer tableau de taille 7
    POUR i DE 0 À 6 FAIRE
        mesNumeros[i] ← loterie[i]
    FIN POUR
    AFFICHER(mesNumeros)
FIN
```

📝 Algorithme complet : Simulation de loterie

```
ALGORITHME PRINCIPAL SimulationLoterie
DEBUT
    N ← 42
    loterie ← créer tableau de taille N
    POUR i DE 0 À N-1 FAIRE
        loterie[i] ← i + 1
    FIN POUR
    loterie ← FisherYates(loterie, N)
    mesNumeros ← créer tableau de taille 7
    POUR i DE 0 À 6 FAIRE
        mesNumeros[i] ← loterie[i]
    FIN POUR
    AFFICHER(mesNumeros)
FIN
```

💡 **Stratégie gagnante :**

1️⃣ Initialiser : mettre tous les numéros possibles

2️⃣ Mélanger : utiliser Fisher-Yates (O(n))

3️⃣ Extraire : prendre les k premiers éléments

🎯 Points clés à retenir

🎯 Points clés à retenir

1️⃣ **Fisher-Yates est LA solution optimale**

- O(n) en temps, O(1) en mémoire
- Mélange équitable garanti

🎯 Points clés à retenir

1️⃣ **Fisher-Yates est LA solution optimale**

- O(n) en temps, O(1) en mémoire
- Mélange équitable garanti

2️⃣ **L'astuce : échanger au lieu de supprimer**

- Évite les opérations coûteuses
- Permet le mélange in-place

🎯 Points clés à retenir

1️⃣ **Fisher-Yates est LA solution optimale**
- O(n) en temps, O(1) en mémoire
- Mélange équitable garanti

2️⃣ **L'astuce : échanger au lieu de supprimer**
- Évite les opérations coûteuses
- Permet le mélange in-place

3️⃣ **Attention au ALEATOIRE(0, i) !**
- Important pour inclure l'index i lui-même
- ALEATOIRE(0, i) signifie : un nombre entre 0 et i **inclus**
- Garantit l'équité du mélange

Fin du cours

Questions ?

Joseph AZAR
joseph.azar@univ-fcomte.fr
IUT Nord Franche-Comté

📚 Ressources : Wikipedia - Fisher-Yates