

Tri par sélection

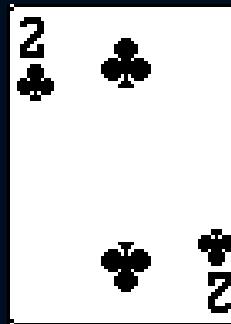
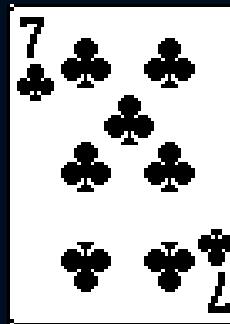
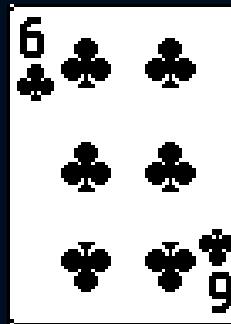
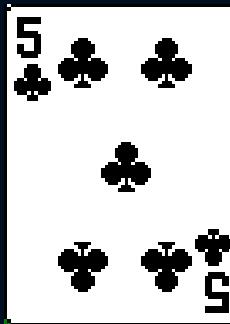
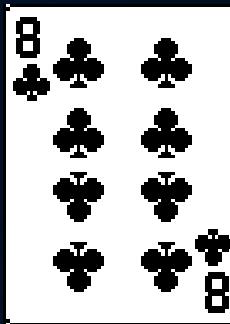
DEMERVILLE ERWAN

Activité préliminaire

- Prendre 5 cartes aléatoirement.
- Placer les cartes les unes à la suite des autres :
 - Les cartes sont numérotées de **1 à 5**.
 - Les positions des cartes sont numérotées de **1 à 5**.
- Trier **instinctivement** les cartes en **notant chaque action effectuée**.
(Exemples : « Comparer valeurs carte 2 et carte 1 », « Décaler carte 4 d'un cran à droite », « placer carte 5 en position 2 », etc.)

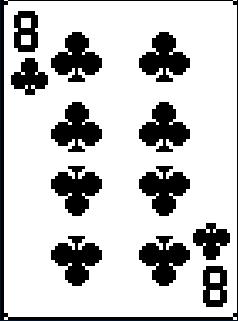
Activité préliminaire

- Voici une liste de 5 cartes non triées :



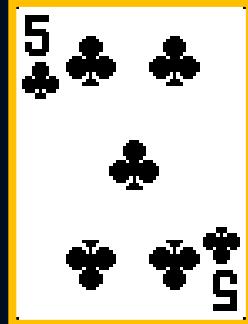
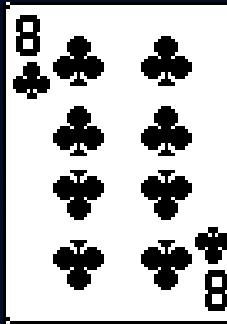
- On veut trier ces cartes en commençant par la première carte à gauche et en ajoutant chaque carte au fur et à mesure.

Activité préliminaire



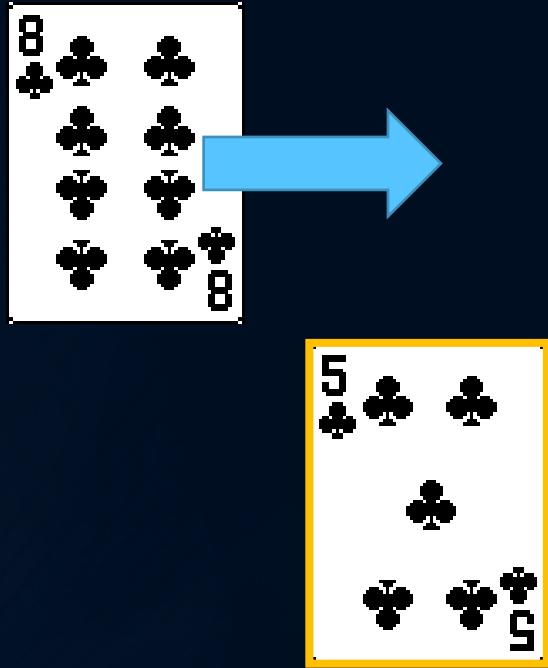
- On commence par la première carte, de valeur **8**.
- Une seule carte : La liste est donc **triée**.

Activité préliminaire



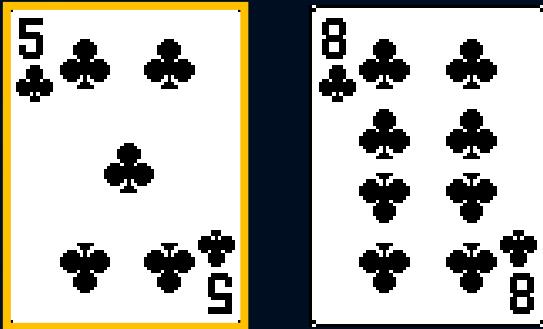
- On ajoute une carte de valeur 5.
- On compare la valeur de la nouvelle carte avec la valeur de la carte précédente.
- Ici, $5 < 8$, donc :

Activité préliminaire



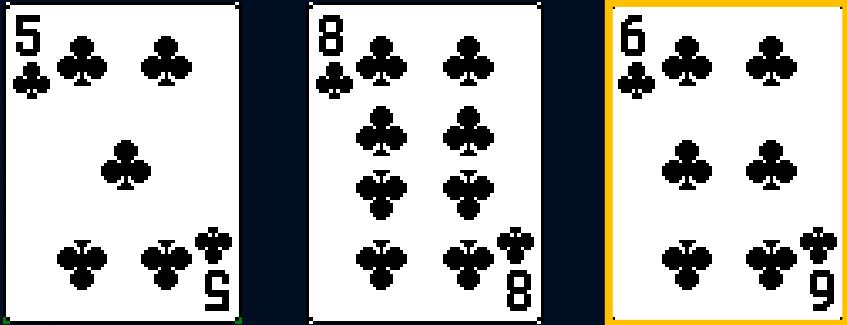
- Ici, $5 < 8$, donc :
 - On décale la carte 8 d'un cran à droite et on met la carte 5 de côté (= en mémoire).

Activité préliminaire



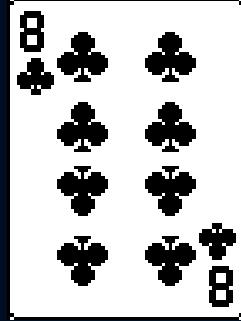
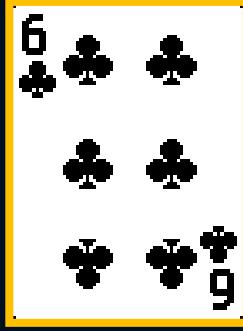
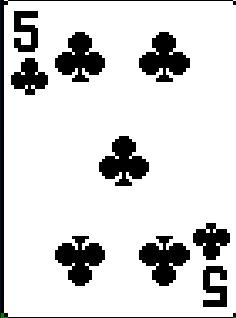
- Ici, $5 < 8$, donc :
 - On décale la carte 8 d'un cran à droite.
 - On place la carte 5 à l'ancienne position de la carte 8.

Activité préliminaire



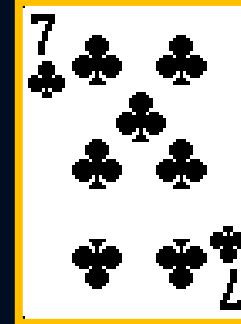
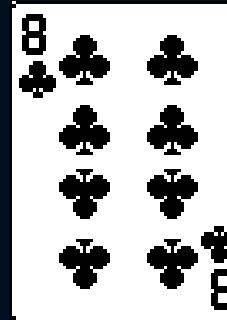
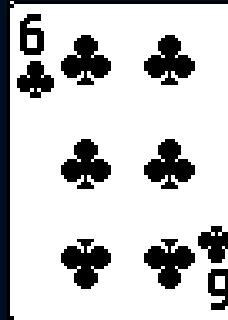
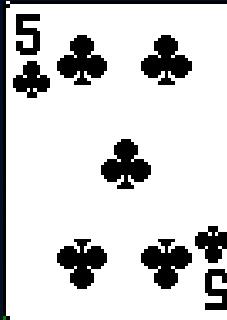
- On ajoute une carte de valeur 6.
- On compare la valeur de la nouvelle carte avec la valeur des cartes précédentes.
- $6 < 8$ et $6 > 5$, donc :
 - On inverse les cartes 8 et 6

Activité préliminaire



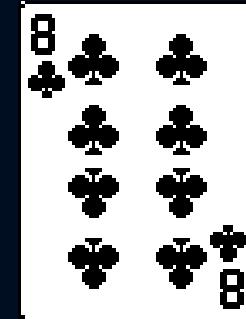
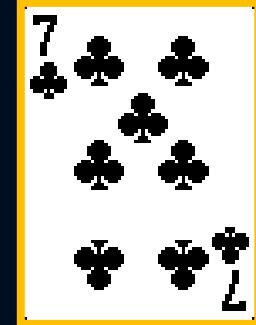
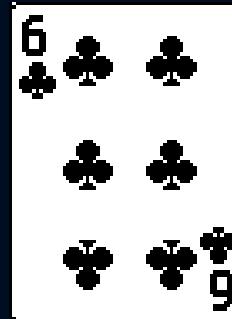
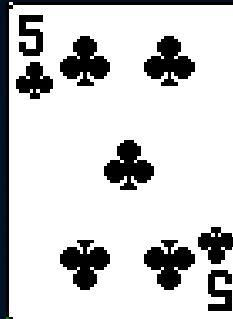
- La liste est **triée**.

Activité préliminaire



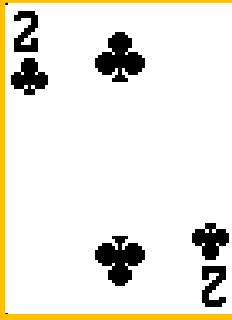
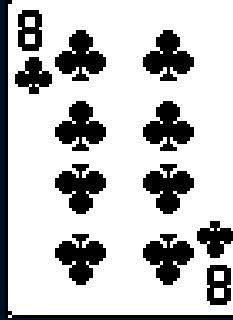
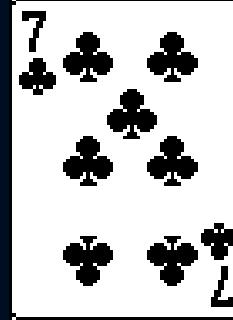
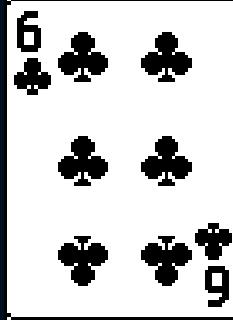
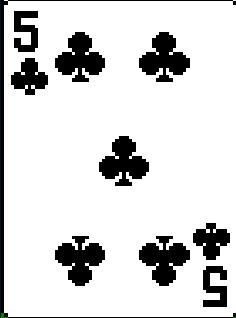
- On ajoute une carte de valeur 7.
- On compare la valeur de la nouvelle carte avec la valeur des cartes précédentes.
- Ici, $7 < 8$ et $7 > 6$, donc :
 - On inverse 7 et 8.

Activité préliminaire



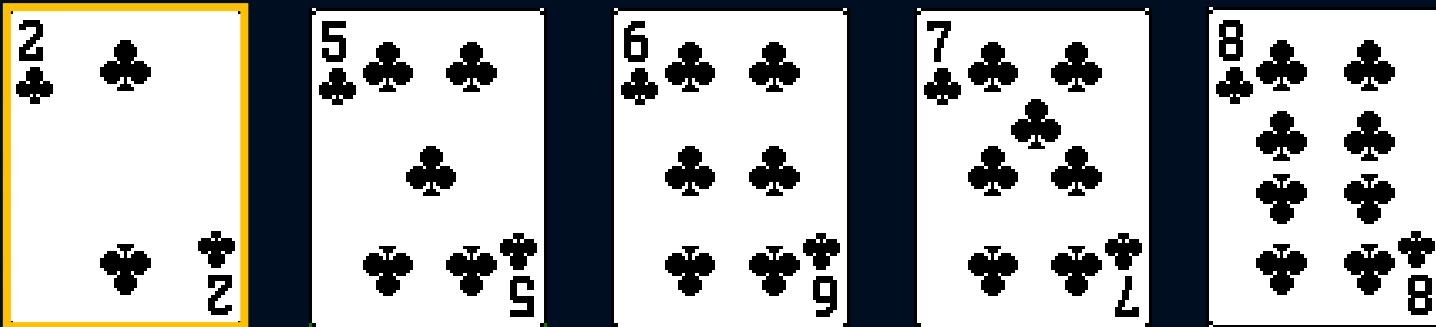
- La liste est **triée**.

Activité préliminaire



- On ajoute une carte de valeur 2.
- $2 < 8, 2 < 7, 2 < 6$ et $2 < 5$:
 - La carte 2 se place alors en **première** position.

Activité préliminaire



- On ajoute une carte de valeur 2.
- $2 < 8, 2 < 7, 2 < 6$ et $2 < 5$:
 - La carte 2 se place alors en **première** position.
- La liste est **triée**.

Présentation de l'algorithme

- Méthode de tri naturelle mais pas la plus efficace.
- On parcourt tous les éléments, chacun est inséré à sa place dans les éléments qui le précèdent.
- Propriétés du tri par insertion :
 - Tri **en place**, pas besoin de faire une copie de la liste;
 - Tri **stable**, deux éléments égaux resteront dans le même ordre;
 - Tri **en ligne**, les éléments de la liste pourraient être fournis au fur et à mesure;

Présentation de l'algorithme



Présentation de l'algorithme

➤ Exercice : Adapter en **pseudo-code** le déroulement de l'algorithme décrit ci-dessous.

On admet les variables ci-dessous :

- **tab** correspondant à un tableau d'entiers.
- **taille** correspondant à la taille du tableau.

1. Pour chaque valeur du tableau :

1. On stocke la valeur **courante** dans la variable **cle**
2. On stocke l'indice de l'élément **juste à gauche** de l'élément courant dans **i**
3. Tant que **i** est supérieur ou égal à **zéro** et que l'élément du tableau à la position **i** est supérieur à l'élément **courant** :
 1. On stocke l'élément du tableau contenu à la position **i** à la position juste à droite.
 2. On stocke dans **i** l'indice de l'élément juste à gauche de l'élément à la position **i**.
4. On stocke dans la position juste à droite de la position **i** du tableau l'élément contenu dans **cle**.

Présentation de l'algorithme

- Voici l'algorithme en pseudo-code :

Pour **j** allant de 1 à **taille** – 1 :

cle prend la valeur **tab[j]**
 i prend la valeur **j** – 1

 Tant que **i** ≥ 0 et que **tab[i]** $>$ **cle** :

tab[i + 1] prend la valeur **tab[i]**
 i prend la valeur **i** – 1

tab[i + 1] prend la valeur **cle**

- Exercice : Rappeler la fonction Python permettant de déterminer la taille d'une liste.
- Exercice : Implémenter l'algorithme en Python.

Présentation de l'algorithme

- Voici l'algorithme en Python :

```
1  def tri_insertion(tab):
2      """ Effectue le tri insertion sur un tableau passé en entrée.
3          :param tab: (Tableau d'int) Tableau à trier
4          :return: PAS DE RETOUR """
5
6      taille = len(tab)
7      for j in range(1, taille):
8          cle = tab[j]
9          i = j - 1
10         while i >= 0 and tab[i] > cle:
11             tab[i + 1] = tab[i]
12             i = i - 1
13         tab[i + 1] = cle
14
```

- Pour voir le déroulement pas à pas d'un algorithme : <http://pythontutor.com/>

Présentation de l'algorithme

➤ Exercice : Compléter le tableau suivant :

```
1 def tri_insertion(tab):
2     taille = len(tab)
3     for j in range(1, taille):
4         cle = tab[j]
5         i = j - 1
6         while i >= 0 and tab[i] > cle:
7             tab[i + 1] = tab[i]
8             i = i - 1
9             tab[i + 1] = cle
10
11
12 tab = [9, 8, 5, 4, 7, 6]
13 tri_insertion(tab)
```

Valeur de j	Tableau avant la boucle for	Valeur de la clé	Tableau en fin de boucle for
1			
2			
3			
4			
5			

Présentation de l'algorithme

➤ Correction :

```
1 def tri_insertion(tab):
2     taille = len(tab)
3     for j in range(1, taille):
4         cle = tab[j]
5         i = j - 1
6         while i >= 0 and tab[i] > cle:
7             tab[i + 1] = tab[i]
8             i = i - 1
9             tab[i + 1] = cle
10
11
12 tab = [9, 8, 5, 4, 7, 6]
13 tri_insertion(tab)
```

Valeur de j	Tableau avant la boucle for	Valeur de la clé	Tableau en fin de boucle for
1	[9,8,5,4,7,6]	8	[8,9,5,4,7,6]
2	[8,9,5,4,7,6]	5	[5,8,9,4,7,6]
3	[5,8,9,4,7,6]	4	[4,5,8,9,7,6]
4	[4,5,8,9,7,6]	7	[4,5,7,8,9,6]
5	[4,5,7,8,9,6]	6	[4,5,6,7,8,9]

Coût de l'algorithme

- On souhaite essayer de déterminer le nombre de comparaisons effectuées entre les valeurs du tableau jusqu'à ce qu'il soit trié.
- Exercice : Complétez le tableau suivant :

	Cas 1 : [2, 7, 9, 12, 15]	Cas 2 : [3, 4, 2, 3, 7]	Cas 3 : [15, 11, 7, 2, 1]
Nombre de comparaisons :			

Coût de l'algorithme

- On souhaite essayer de déterminer le nombre de comparaisons effectuées entre les valeurs du tableau jusqu'à ce qu'il soit trié.

➤ Exercice : Complétez le tableau suivant :

	Cas 1 : [2, 7, 9, 12, 15]	Cas 2 : [3, 4, 2, 3, 7]	Cas 3 : [15, 11, 7, 2, 1]
Nombre de comparaisons :	4	6	10

- Parmi ces 3 cas, quel est :
- Le meilleur des cas ? (Celui nécessitant le moins de comparaisons)
 - Le pire des cas ? (Celui nécessitant le plus de comparaisons)

Coût de l'algorithme

- Rappels sur les complexités :
- Logarithmique $\Theta(\lg n)$: la complexité évolue moins vite que le nombre n de données (par exemple : si on multiplie le nombre de données n par 100, le temps d'exécution n'est multiplié que par 8)
- Linéaire $\Theta(n)$: la complexité évolue comme le nombre n de données (par exemple : si on multiplie le nombre de données n par 2, le temps d'exécution est multiplié par 2)
- Quadratique $\Theta(n^2)$: la complexité évolue comme le carré du nombre n de données (par exemple : si on multiplie le nombres de données n par 2, le temps d'exécution est multiplié par 4, 2^2)
- Exponentielle $\Theta(x^n)$: la complexité évolue à terme beaucoup plus vite que n'importe quelle fonction polynomiale du nombre n de données (par exemple : si on multiplie le nombre de données n par 100, le temps d'exécution est multiplié par 2^{100} , soit 1267650600228229401496703205376)

Coût de l'algorithme

- Observez le tableau ci-dessous. Que peut-on en dire de la complexité du meilleur et du pire des cas ?

	Meilleur cas : [2, 7, 9]	Pire cas : [15, 11, 7]
Nombre de comparaisons :	2	3
	Meilleur cas : [2, 7, 9, 12, 15, 20]	Pire cas : [21, 15, 11, 7, 2, 1]
Nombre de comparaisons :	5	15
	Meilleur cas : [2, 7, 9, 12, 15, 20, 22, 24, 26, 30, 31, 33]	Pire cas : [40, 35, 33, 31, 27, 25, 21, 15, 11, 7, 2, 1]
Nombre de comparaisons :	11	66

Exercices

- Exercice : Voici une autre manière d'écrire le tri par insertion. Complétez l'algorithme suivant :

```
1  def TriInsertion(L):
2      n = -----
3      for i in range(1,n):
4          temp = L[i]
5          p = 0
6          while ----- < temp:
7              p = p + 1
8          for j in range(i-1,p-1,-1):
9              L[j+1] = -----
10         L[p] = -----
11     return L
```

Exercices

- Correction :

```
1 def TriInsertion(L):
2     n = len(L)
3     for i in range(1,n):
4         temp = L[i]
5         p = 0
6         while L[p] < temp:
7             p = p + 1
8         for j in range(i-1,p-1,-1):
9             L[j+1] = L[j]
10        L[p] = temp
11    return L
```

- Question : A quoi correspondent les variables **i** et **p** dans le cadre de la liste **L** ?

Exercices

➤ Correction :

```
1 def TriInsertion(L):
2     n = len(L)
3     for i in range(1,n):
4         temp = L[i]
5         p = 0
6         while L[p] < temp:
7             p = p + 1
8         for j in range(i-1,p-1,-1):
9             L[j+1] = L[j]
10        L[p] = temp
11    return L
```

➤ Question : A quoi correspondent les variables **i** et **p** dans le cadre de la liste **L** ?

i : Indice de l'élément traité dans le premier **for**

p : Indice de la nouvelle position de l'élément traité (à la fin du **while**)

Exercices

- Exercice : Modifiez l'algorithme pour trier dans l'ordre décroissant.

```
1  def TriInsertion(L):
2      n = len(L)
3      for i in range(1,n):
4          temp = L[i]
5          p = 0
6          while L[p] < temp:
7              p = p + 1
8          for j in range(i-1,p-1,-1):
9              L[j+1] = L[j]
10         L[p] = temp
11     return L
```

Exercices

➤ Correction :

```
1 def TriInsertion(L):
2     n = len(L)
3     for i in range(1,n):
4         temp = L[i]
5         p = 0
6         while L[p] > temp:
7             p = p + 1
8             for j in range(i-1,p-1,-1):
9                 L[j+1] = L[j]
10            L[p] = temp
11    return L
```

Exercices

- Quel **module** et quelle **fonction** de ce **module** utiliser pour générer des nombres aléatoires ? Ecrire une fonction **generer_tab** qui retourne un tableau contenant 5 nombres aléatoires compris entre 1 et 99.

- Ecrire une fonction **est_triee** qui prend un tableau en entrée et retourne **True** s'il est trié par ordre croissant, **False** sinon.
- Faire le test avec les tableaux **tab1 = [3, 2, 9, 1, 5, 6]** et **tab2 = [2, 4, 5, 10, 16, 22]**

FIN