Algo

Expression booléenne :

* == : égal
* != : différent
* < : plus petit
* <= : strictement plus petit
* > : plus grand
* >= : strictement plus grand
* && : et
* || : ou
* ! : non

Méthodes en Java :

* Le if

Syntaxe :

If (condition) {

instruction

} else {

instruction

}

On peut imbriquer les if pour avoir d’autres conditions dans le premier if ou else

* Switch

Syntaxe :

switch( expression ) {

case « test » :

instruction

break ;

case « test2 » :

instruction

break ;

default :

break ;

Le switch permet dans certains cas de ramener plusieurs if en une seule et même méthode

* For

Syntaxe :

for ( init ; condition de continuation ; incrément ) {

instruction

}

Le programme sort de la boucle lorsque la condition de continuation n’est plus vérifiée

* while

Syntaxe :

while( condition de continuation ) {

instruction

}

Le progamme sort de la boucle lorsque la condition de continuation n’est plus vérifiée

* do while

Syntaxe :

do {

instruction

} while (condition de continuation) ;

Le programme contrairement à la boucle while rentre une premiere fois dans la boucle avant de vérifier la condition de continuation. Il s’arrete lorsqu’elle est plus vérifée

Dans les trois boucles énoncées ci-dessous, nous pouvons mettre dans la boucle une sortie prématurée. Soit un return, soit un break. Mais quel est la différence entre les deux. Le return fait sortir dans la méthode alors que le break seulement de la boucle.

Pour développer un programme, il existe deux méthodes :

* Bottom-Up : on commence par réaliser une série de méthodes avant de s’attaquer au programme en lui-meme
* Top-Down : on pense d’abord au programme principal avant de faire des méthodes et réduire le nombre de répétitons d’instructions

Qu’est ce qu’une méthode :

* C’est le fait de prendre dans le code des instructions qui se répètent pour les mettre ensemble et y faire appel à plusieurs endroits.
* Une méthode a un emplacement en mémoire à part, cad que l’on peut déclarer des variables qui ont le même nom que celle présente dans le main
* Syntaxe :

Public/private static typeDeRetour nomDeLaMéthode( paramètres ) {

Instruction

return variable ou si méthode void pas de return kassos, il nous fait donc sortir de la boucle}

Les tableaux

1. A une dimension

Un tableau commencer à la valeur 0 donc si on a un tableau de 12 cases, l’index se termine à 11

Pour parcourir le tableau on utilise une boucle en utilisant un int qui permet de récupérer une valeur dans une des cases t[i]

Mais attention à l’exeption ArrayIndexOutOfBoundsExeption

Le cout d’une méthode :

* N 🡪 nombre d’éléments dans la table.
* O(1) 🡪 le coût (d’exécution) ne dépend pas de la taille de la table

Quel que soit le nombre d’éléments dans la table, le coût est fixe

coût « constant »

* O(N) 🡪 le coût (d’exécution) dépend de la taille de la table

Plus il y a d’éléments, plus le coût sera élevé !

coût « linéaire »

Différence entre taille logique et physique

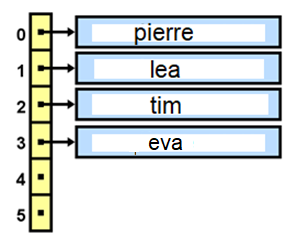
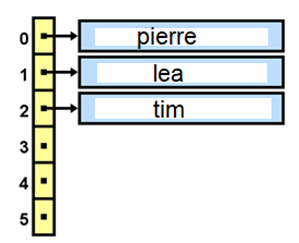
Logique : le nombre d’éléments qu’il est possible de mettre dans la table

Physique : le nombre d’éléments dans la table

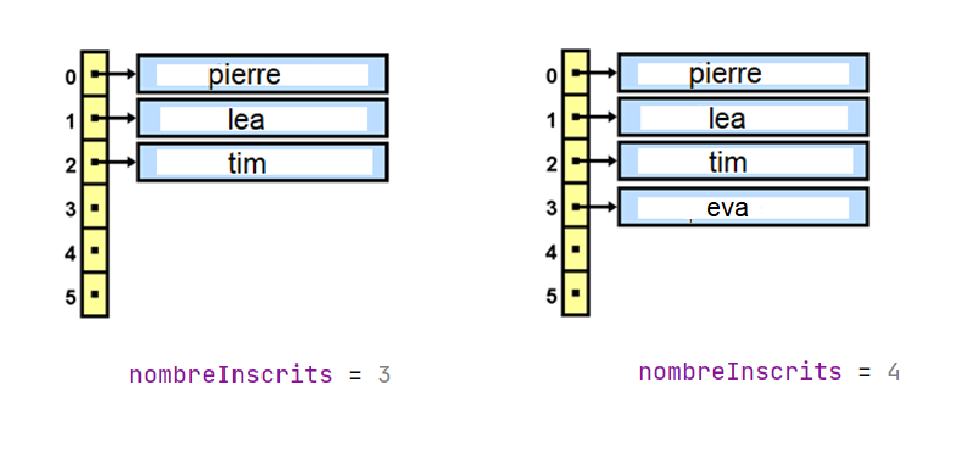
Les algorithmes de base :

* Ajout

Variante 1



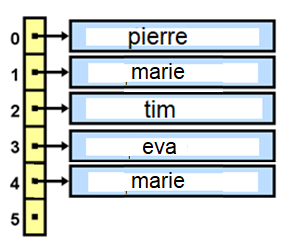
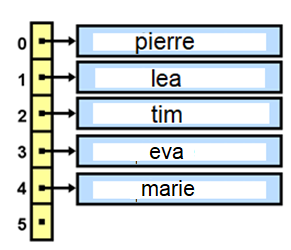
Variante 2



Le nombre d’inscrits passent de 3 à 4

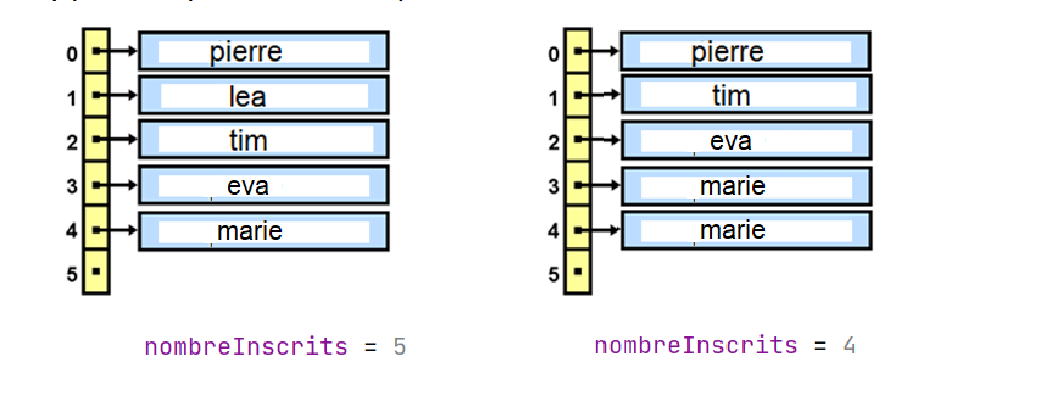
* Suppression

Variante 1 : Algorithme de suppression : « on remplace l’élément à supprimer par le dernier » mais il faut d’abord rechercher l’élément à supprimer

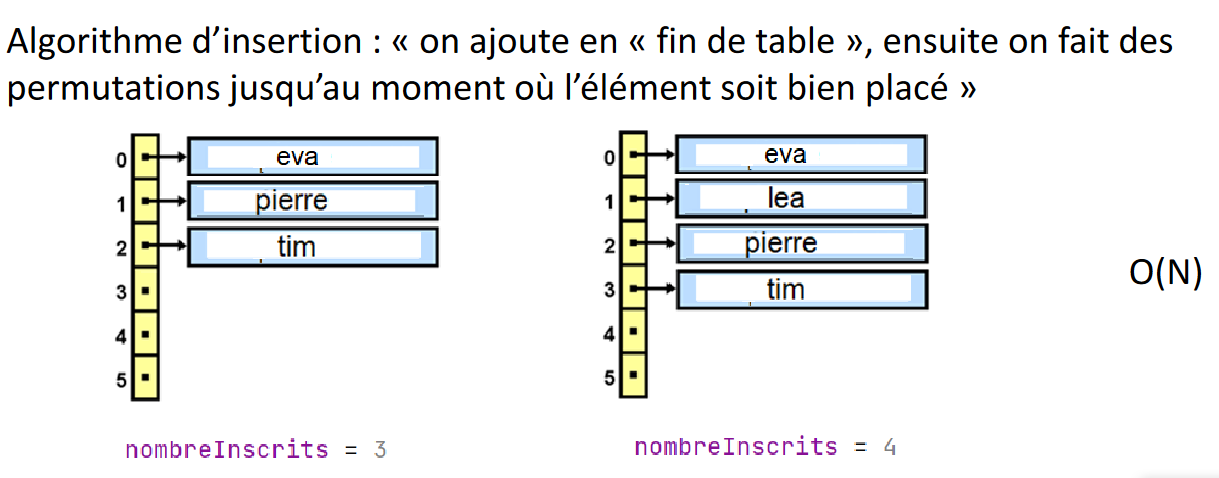


Algorithme de suppression : « on remplace l’élément à supprimer par le dernier »

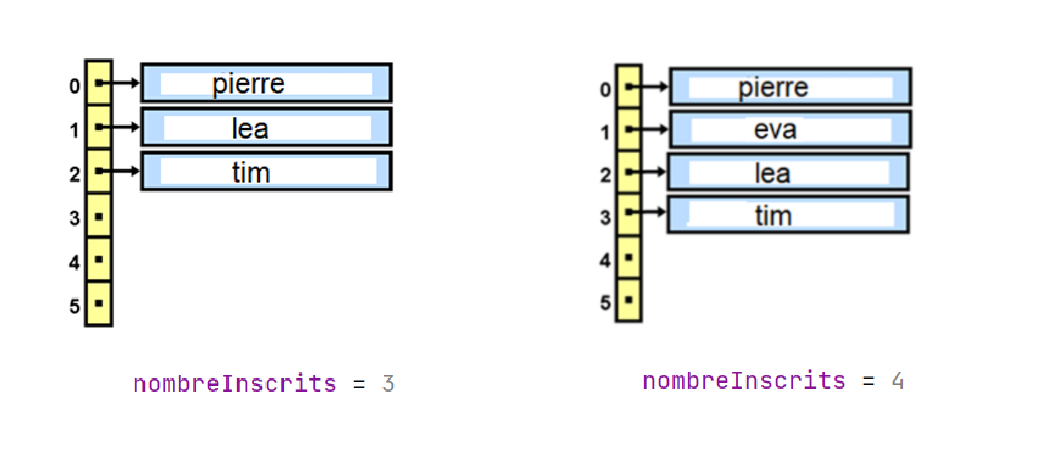
Variante 2



Variante 3

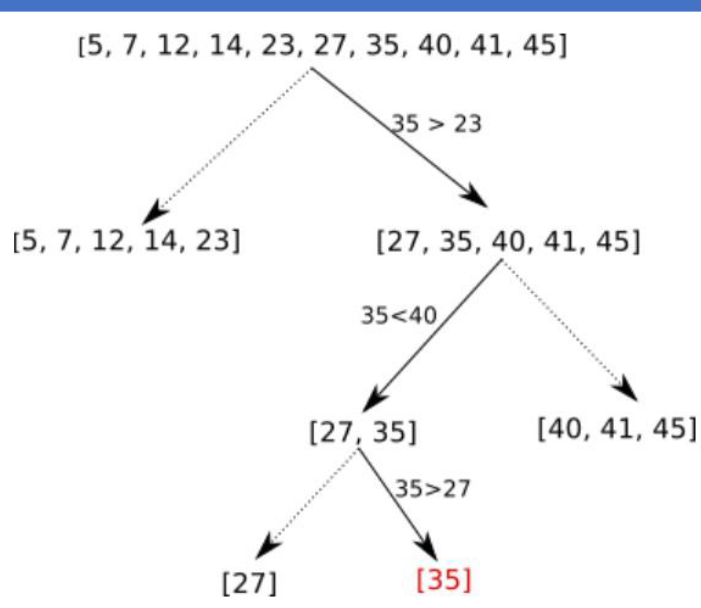


* Insertion



Recherche

Une image contenant texte

Description générée automatiquementUne image contenant texte, lumière, blanc

Description générée automatiquement

Les algorithmes de tri

En O(N²)

* Tri bulle
* Tri par sélection

On cherche d'abord dans toute la table quel est le plus petit élément, et  
on vient le placer en première position en le permutant avec celui qui s'y  
trouve. Ensuite, on recommence pour le plus petit de ceux qui restent, et  
on le place en deuxième position. Et ainsi de suite jusqu'à l'avant-dernier.  
Le dernier est forcément déjà bien placé quand on y arrive.

* Tri par insertion

On considère chaque élément du tableau les uns après les autres.   
Pour chaque élément, on l’insère à la bonne place parmi les éléments   
déjà triés.   
Pour insérer un élément, il va falloir faire des décalages pour libérer une   
place pour mettre l’élément au bon endroit.  
Au moment où on considère un élément, les éléments qui le précèdent   
sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore   
triés.

Ou

On considère chaque élément du tableau les uns après les autres.   
Pour chaque élément, on l’insère à la bonne place parmi les éléments   
déjà triés.   
On « remonte » l’élément jusqu’au moment où on en trouve un plus petit   
que lui.  
Au moment où on considère un élément, les éléments qui le précèdent   
sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore   
triés.

En O(NlogN)

* Tri par fusion
* Quick sort (tri rapide)
* Heap sort

En O(N)

* Tri par comptage

Construction d’un tableau de comptage, puis balayage de celui-ci afin de  
reconstruire les données triées.