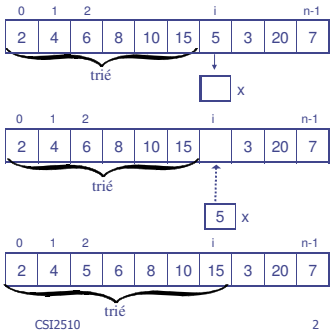
Tris quadratiques

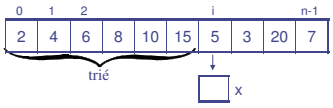
Révision
- Tri par insertion
- Tri par sélection

Tri à bulle (Bubble Sort)

Insertion Sort (tableau)




```
for i=1 to n-1
  x ← A[i]
  j ← i-1
  while x.key < A[j].key
        and j >= 0
     A[j+1] ← A[j]
     j ← j-1
  A[j+1] ← x
```



Complexité — Insertion sort pour tableaux

Comparaisons:

MIN: $(C_i)_{min} = 1$, i=2..n (deja trié) → $C_{min} = n-1 = O(n)$

MAX: $(C_i)_{max} = i$, i=2..n (trié en ordre invers) → $C_{max} = \sum_{i=2}^{n} i$

$= [n(n+1)/2] - 1$

$= O(n^2)$

Et avec une liste chaînée ?

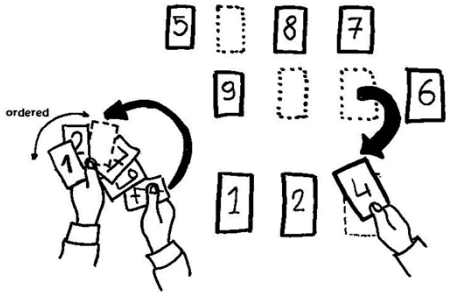
Complexité — Insertion sort pour tableaux

Nb. de déplacements et d'affectations

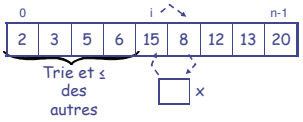
idem Nb. de comparaisons

Et avec une liste chaînée ?

Selection Sort (tableaux)



Selection Sort (tableaux)



Chercher la plus petite clé

```
for i=1 to n-1
  k ← i
  x ← A[i]
  for j=i+1 to n
    if A[j].key < x.key
          k ← j
          x ← A[j]
  A[k] ← A[i]
  A[i] ← x
```


Complexité de Selection Sort (tableau)

Nb. de comparaisons ne dépend pas de l'ordre des clés

$$C = \sum_{i=1}^{n-1} (n - i) = n(n-1) - n(n-1)/2$$
$$= n(n-1)/2$$
$$= O(n^2)$$

Et avec une liste chaînée ?

Complexité de Selection Sort (tableau)

Nb. de d'affectations et de déplacements:

MIN	Tableau déjà trié	$D_{min} = n-1 =$ $O(n)$
MAX	$(D_i)_{max} = i$ (i=2..n) (trié en ordre inverse)	$D_{max} = \sum_{i=1}^{n-1}(n-i)$ $O(n^2)$

Tri à bulle (Bubblesort)

- Un algorithme de tri assez simple
- Comme son nom l'indique, son principe consiste à "faire remonter" les éléments jusqu'à leur bonne position (comme des bulles d'air à la surface de l'eau)
- On compare successivement les paires d'éléments adjacents du tableau et on les permute si le premier est supérieur au second

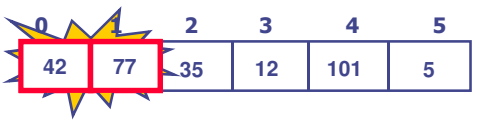
Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin

0	1	2	3	4	5
77	42	35	12	101	5

Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin



Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin

Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin

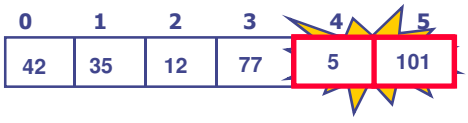
Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin

Pas besoin de permuter

Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin



Faire monter le plus grand élément

- **Traverser une collection d'éléments**
  - Parcourir le tableau du début vers la fin
  - Déplacer le plus grand élément vers la fin du tableau en comparant les éléments deux par deux et en les permutant si besoin

0	1	2	3	4	5
42	35	12	77	5	101

Le plus grand élément est placé à la bonne position

Tri à bulles (Bubble sort)

- Seul le plus grand élément est bien placé dans le tableau
- Les autres éléments du tableau sont toujours en désordre
- Il faut « faire remonter » les autres éléments aux bonnes positions → répéter la procédure

0	1	2	3	4	5
42	35	12	77	5	101

Le plus grand élément est placé à la bonne position

Tri à bulles (Bubble sort)

- Si nous avons N éléments…
- Et si chaque fois nous faisons remonter un élément et nous le plaçons à la bonne position
- Alors nous répétons la procédure "faire remonter" N – 1 fois
- Après ces procédures, le tableau sera nécessairement trié

Faire monter tout les éléments

N - 1

0	1	2	3	4	5
42	35	12	77	5	101
35	12	42	5	77	101
12	35	5	42	77	101
12	5	35	42	77	101
5	12	35	42	77	101

Tri a bulles(tableau) - Complexité

Nb. de comparaisons (ne dépend pas de l'ordre):

$C = \sum_{i=1}^{n-1}(n-i)$ = n(n - 1) - n(n - 1) / 2 = n(n - 1) / 2 = $O(n^2)$

Nb. de déplacements et affectations:

D_{min} = 0 Tableau dans l'ordre

D_{max}= 3*C = $O(n^2)$ Tableau dans l'ordre inverse

Tri a bulles(tableau)

- Et si le tableau est déjà trié ?
- Et si seulement quelques éléments ne sont pas en ordre et après quelques déplacements le tableau est trié ?
- Nous voulons pouvoir détecter ceci et "arrêter tôt" !

0	1	2	3	4	5
5	12	35	42	77	101

Tri a bulles(tableau)

- Nous pouvons utiliser une variable booléenne pour déterminer si n'importe quelle permutation est arrivée pendant la dernière remontée d'élément
- Si aucune permutation n'est arrivée, alors nous savons que le tableau est déjà trié!
- Ce booléen "swapped" a besoin d'être remis à l'état initial après chaque remontée d'élément

Tri a bulles (tableau)

```
j ← 0
swapped ← true
while (swapped)
  swapped ← false
  j ← j+1
  for i=0 to n-j-1
     if A[i].key > A[i+1].key
        tmp= A[i]
        A[i]= A[i+1]
        A[i+1]= tmp
        swapped ← true
```