

# Module algorithmie et programmation

Tris

---

Pierre Künzli

Adapté des cours de Paul Albuquerque et Orestis Malaspinas

# Rappel: Complexité

Soit `tab` un tableau de longueur `N` de `double`.

1. Comment déclare-t-on un tel tableau?

# Rappel: Complexité

Soit `tab` un tableau de longueur `N` de `double`.

1. Comment déclare-t-on un tel tableau?

```
double tab[N];
```

ou

```
double *tab = malloc(N*sizeof(double));
```

2. Quelle est la complexité du calcul de la moyenne de `tab`?

# Rappel: Complexité

Soit `tab` un tableau de longueur `N` de `double`.

1. Comment déclare-t-on un tel tableau?

```
double tab[N];
```

ou

```
double *tab = malloc(N*sizeof(double));
```

2. Quelle est la complexité du calcul de la moyenne de `tab`?

```
double mean = 0.0;
for (int i = 0; i < N; ++i) { // N assignments
    mean += tab[i];           // N additions
}
mean /= N; // O(N)
```

3. Quelle est la complexité du calcul de l'écart type de tab  $\sigma = \sqrt{\sum_i (t_i - m)^2 / N}$ ?

# Rappel: Complexité

3. Quelle est la complexité du calcul de l'écart type de tab  $\sigma = \sqrt{\sum_i (t_i - m)^2 / N}$ ?

```
double mean = moyenne(N, tab); // O(N)
double dev = 0.0;
for (int i = 0; i < N; ++i) {
    dev += pow(tab[i] - moyenne, 2); // N tours
}
dev = sqrt(dev); dev /= N; // O(2*N) = O(N)
```

## **But**

- trier un tableau par ordre croissant

## **Rappel**

- tri par sélection

# Tri par insertion

## Algorithme

Prendre un élément du tableau et le mettre à sa place parmi les éléments déjà triés du tableau.

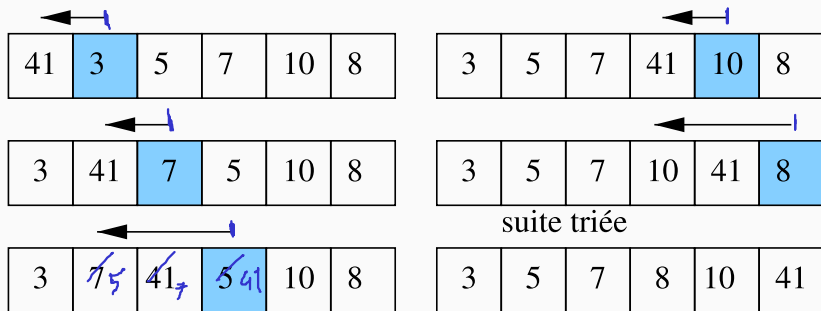


Figure 1: Tri par insertion d'un tableau d'entiers



# Tri par insertion

Et en C à quoi ça ressemble ?

# Tri par insertion

## L'algorithme en C

```
void tri_insertion(int N, int tab[N]) {  
    for (int i = 1; i < N; i++) {  
        int tmp = tab[i];  
        int pos = i;  
        while (pos > 0 && tab[pos - 1] > tmp) {  
            tab[pos] = tab[pos - 1];  
            pos      = pos - 1;  
        }  
        tab[pos] = tmp;  
    }  
}
```

**Question: Quelle est la complexité?**

**Question: Quelle est la complexité?**

- Parcours de tous les éléments ( $N - 1$  passages dans la boucle)
  - Placer: en moyenne  $i$  comparaisons et affectations à l'étape  $i$

## Question: Quelle est la complexité?

- Parcours de tous les éléments ( $N - 1$  passages dans la boucle)
  - Placer: en moyenne  $i$  comparaisons et affectations à l'étape  $i$
- Moyenne:  $\mathcal{O}(N^2)$
- Pire des cas, liste triée à l'envers:  $\mathcal{O}(N^2)$
- Meilleurs des cas, liste déjà triée:  $\mathcal{O}(N)$

# L'algorithme à la main

## Exercice *sur papier*

- Trier par insertion le tableau [5, -2, 1, 3, 10]

5, -2, 1, 3, 10  
-2, 5, 1, 3, 10  
-2, 1, 5, 3, 10  
-2, 1, 3, 5, 10

$tmp = -2$

$tmp = 1$

$tmp = 3$

$tmp = 10$

# Tri rapide ou quicksort

**Idée: algorithme diviser pour régner (divide-and-conquer)**

- Diviser: découper un problème en sous problèmes;
- Régner: résoudre les sous-problèmes (souvent récursivement);
- Combiner: à partir des sous problèmes résolu, calculer la solution.

**Le pivot**

- Trouver le **pivot**, un élément qui divise le tableau en 2, tels que:
  1. Éléments à gauche sont **plus petits** que le pivot.
  2. Éléments à droite sont **plus grands** que le pivot.

# Tri rapide ou quicksort

## Algorithme quicksort(tableau)

1. Choisir le pivot et l'amener à sa place:
  - Les éléments à gauche sont plus petits que le pivot.
  - Les éléments à droite sont plus grand que le pivot.
2. quicksort(tableau\_gauche) en omettant le pivot.
3. quicksort(tableau\_droite) en omettant le pivot.
4. S'il y a moins de deux éléments dans le tableau, le tableau est trié.



# Tri rapide ou quicksort

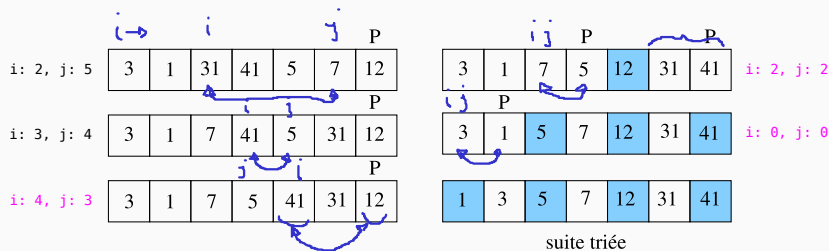
## Algorithme quicksort(tableau)

1. Choisir le pivot et l'amener à sa place:
  - Les éléments à gauche sont plus petits que le pivot.
  - Les éléments à droite sont plus grand que le pivot.
2. quicksort(tableau\_gauche) en omettant le pivot.
3. quicksort(tableau\_droite) en omettant le pivot.
4. S'il y a moins de deux éléments dans le tableau, le tableau est trié.

Compris?

Non c'est normal, faisons un exemple.

# Tri rapide ou quicksort



**Figure 2:** Un exemple de quicksort.

# Tri rapide ou quicksort

Deux variables sont primordiales:

```
entier ind_min, ind_max; // les indices min/max des tableaux à trier
```

## Pseudocode: quicksort

```
rien quicksort(entier tableau[], entier ind_min, entier ind_max)
    si (longueur(tab) > 1)
        ind_pivot = partition(tableau, ind_min, ind_max)
        si (longueur(tableau[ind_min:ind_pivot-1]) > 1)
            quicksort(tableau, ind_min, ind_pivot - 1)
        si (longueur(tableau[ind_pivot+1:ind_max-1]) > 1)
            quicksort(tableau, ind_pivot + 1, ind_max)
```

# Tri rapide ou quicksort

## Pseudocode: partition

```
entier partition(entier tableau[], entier ind_min, entier ind_max)
    pivot = tableau[ind_max] // choix arbitraire
    i = ind_min
    j = ind_max-1
    tant que i < j:
        en remontant i trouver le premier élément > pivot
        en descendant j trouver le premier élément < pivot
        échanger(tableau[i], tableau[j])
        // les plus grands à droite
        // mettre les plus petits à gauche

    // on met le pivot "au milieu"
    échanger(tableau[i], tableau[ind_max])
    retourne i // on retourne l'indice pivot
```

## Tri rapide ou quicksort

A quoi ressemblent les fonctions quicksort et partition en C ?

# Tri rapide ou quicksort

**A quoi ressemblent les fonctions quicksort et partition en C ?**

```
void quicksort(int* array, int first,
               int last)
{
    if (first < last) {
        int midpoint = partition(array, first, last);
        if (first < midpoint - 1) {
            quicksort(array, first, midpoint - 1);
        }
        if (midpoint + 1 < last) {
            quicksort(array, midpoint + 1, last);
        }
    }
}
```

# Tri rapide ou quicksort

## A quoi ressemblent les fonctions quicksort et partition en C ?

```
void swap(int* a, int* b){
```

```
    int tmp = *a;
```

```
    *a = *b;
```

```
    *b = tmp; }
```

```
int partition(int* array, int first, int last) {
```

```
    int pivot = array[last];
```

```
    int i = first - 1, j = last;
```

```
    do {
```

```
        do {
```

```
            i += 1;
```

```
        } while (array[i] < pivot && i < j);
```

```
        do {
```

```
            j -= 1;
```

```
        } while (array[j] > pivot && i < j);
```

```
        if (j > i) {
```

```
            swap(&array[i], &array[j]);
```

```
        }
```

```
    } while (j > i);
```

```
    swap(array+i, array+last);
```

```
    return i; }
```

*swap (&array[i], &array [last])*

# Tri rapide ou quicksort

Quelle est la complexité du tri rapide?



## Quelle est la complexité du tri rapide?

- Pire des cas plus:  $\mathcal{O}(N^2)$ 
  - Quand le pivot sépare toujours le tableau de façon déséquilibrée ( $N - 1$  éléments d'un côté 1 de l'autre).
  - $N$  boucles et  $N$  comparaisons  $\Rightarrow N^2$ .
- Meilleur des cas (toujours le meilleur pivot):  $\mathcal{O}(N \cdot \log_2(N))$ .
  - Chaque fois le tableau est séparé en 2 parties égales.
  - On a  $\log_2(N)$  partitions, et  $N$  boucles  $\Rightarrow N \cdot \log_2(N)$ .
- En moyenne:  $\mathcal{O}(N \cdot \log_2(N))$ .

# L'algorithme à la main

## Exercice *sur papier*

- Trier par tri rapide le tableau [5, -2, 1, 3, 10, 15, 7, 4]

$i$   $j$   $P$   
5, -2, 1, 3, 10, 15, 7, 4  
 $j$   $i$   $P$   
3, -2, 1, 5, 10, 15, 7, 4  
 $i$   $j$   $P$   
3, -2, 1, 4, 10, 15, 7, 5  
 $j$   $i$   
-2, 3, 1  
 $j$   $P$   $i$   
-2, 1, 3, 4  
 $i$   $j$   
-2, 1, 3, 4

## Algorithme

- Parcours du tableau et comparaison des éléments consécutifs:
  - Si deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
- On recommence depuis le début du tableau jusqu'à avoir plus d'échanges à faire.

**Que peut-on dire sur le dernier élément du tableau après un parcours?**

## Algorithme

- Parcours du tableau et comparaison des éléments consécutifs:
  - Si deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
- On recommence depuis le début du tableau jusqu'à avoir plus d'échanges à faire.

## Que peut-on dire sur le dernier élément du tableau après un parcours?

- Le plus grand élément est **à la fin** du tableau.
  - Plus besoin de le traiter.
- A chaque parcours on s'arrête un élément plus tôt.

# Tri à bulle

## Exemple

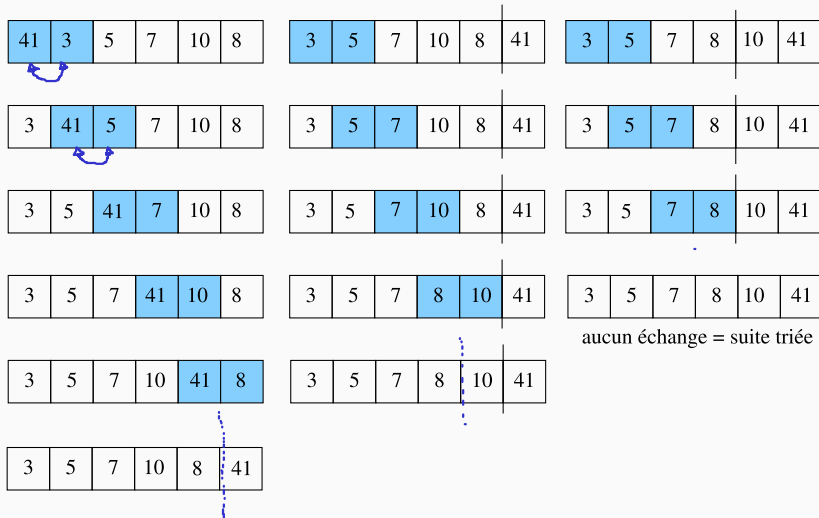


Figure 3: Tri à bulles d'un tableau d'entiers

# Tri à bulle

## L'algorithme du tri à bulle

```
rien tri_a_bulles(entier tableau[])  
  pour i de longueur(tableau)-1 à 1:  
    trié = vrai  
    pour j de 0 à i-1:  
      si (tableau[j] > tableau[j+1])  
        échanger(array[j], array[j+1])  
        trié = faux  
  
  si trié  
    retourner
```

**Quelle est la complexité du tri à bulles?**

## Quelle est la complexité du tri à bulles?

- Dans le meilleurs des cas:
  - Le tableau est déjà trié:  $\mathcal{O}(N)$  comparaisons.
- Dans le pire des cas,  $N \cdot (N - 1)/2 \sim \mathcal{O}(N^2)$ :

$$\sum_{i=1}^{N-1} i \text{ comparaison et } \sum_{i=1}^{N-1} i \text{ affectations (swap)} \Rightarrow \mathcal{O}(N^2).$$

- En moyenne,  $\mathcal{O}(N^2)$ .



# L'algorithme à la main

## Exercice *sur papier*

- Trier par tri à bulles le tableau [5, -2, 1, 3, 10, 15, 7, 4]

