Tris

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA

2022-11-16

En partie inspirés des supports de cours de P. Albuquerque

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];
```

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];
```

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

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

Rappel: Complexité

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

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

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

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

```
double mean = 0.0;
for (int i = 0; i < N; ++i) { // N assignations
    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

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

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

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

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

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

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

```
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)
```

Tri par insertion (1/3)

But

- trier un tableau par ordre croissant

Algorithme

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

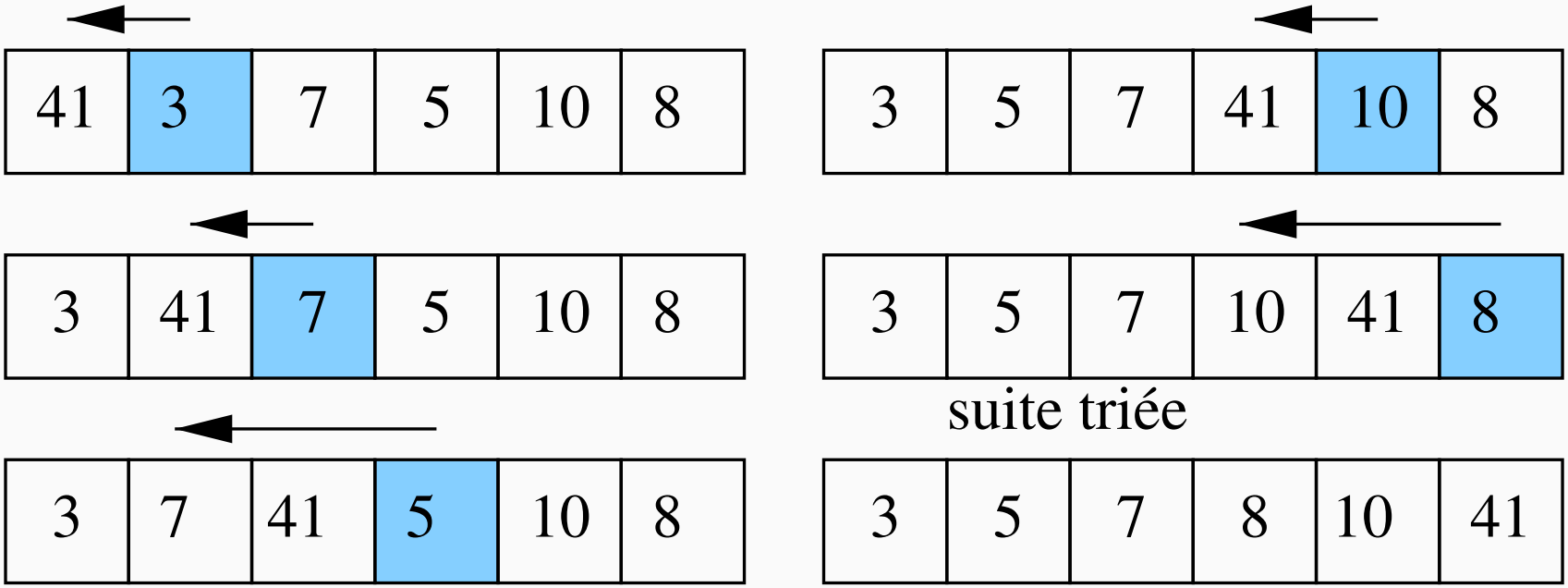

Figure 1: Tri par insertion d'un tableau d'entiers

Tri par insertion (2/3)

Exercice: Proposer un algorithme (en C)

Tri par insertion (2/3)

Exercice: Proposer un algorithme (en C)

```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;
    }
}
```

Tri par insertion (3/3)

Question: Quelle est la complexité?

Tri par insertion (3/3)

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: $\mathcal{O}(N^2)$

Tri par insertion (3/3)

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]`

Tri rapide ou quicksort (1/8)

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. Élements à droite sont **plus grands** que le pivot.

Tri rapide ou quicksort (2/8)

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 (2/8)

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?

Tri rapide ou quicksort (2/8)

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 (3/8)

Deux variables sont primordiales:

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

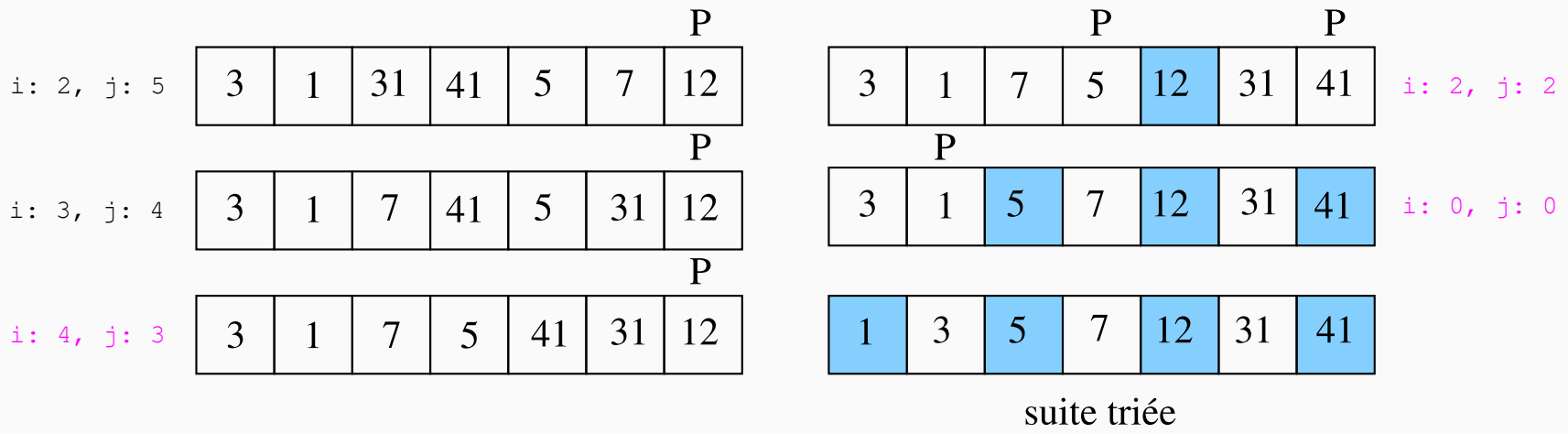

Figure 2: Un exemple de quicksort.

Tri rapide ou quicksort (4/8)

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]) != 0)
            quicksort(tableau, ind_min, pivot_ind - 1)
        si (longueur(tableau[ind_pivot+1:ind_max-1]) != 0)
            quicksort(tableau, ind_pivot + 1, ind_max)
```

Tri rapide ou quicksort (5/8)

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 (6/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

Tri rapide ou quicksort (6/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

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

Tri rapide ou quicksort (7/8)

Exercice: implémenter les fonctions `quicksort` et `partition`

```
int partition(int size, int array[size], 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;
}
```

Tri rapide ou quicksort (8/8)

Quelle est la complexité du tri rapide?

Tri rapide ou quicksort (8/8)

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]`