Algorithmes de tris simples

Philippe Langlais

Plan

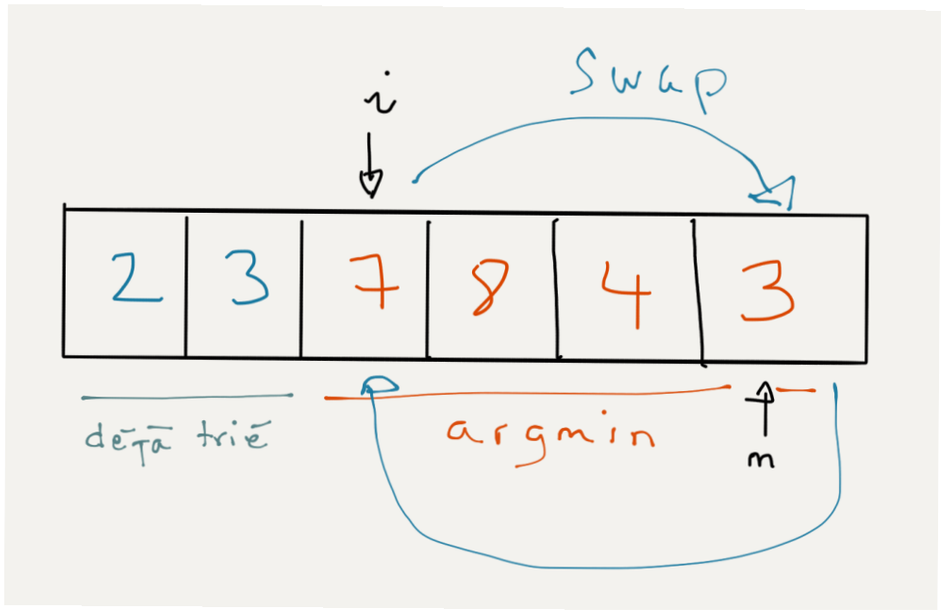
- Tri par sélection
- Tri à bulles
- Tri par insertion
- Notion de complexité

codes dans *sort.js*

Versions — **en place** — de tris de tableaux numériques
en ordre croissant

1-Tri par sélection

- **Idée** au *i*e pas de boucle
  - *t[0,i[* est déjà trié
  - Échanger *t[i]* avec la plus petite valeur dans *t[i,|t|[*



```
/*
 * tri par selection -- en place -- du tableau t en ordre croissant
 * @param t tableau
 * @return l'adresse du tableau
 */
var triSelection = function(t) {

    if (Array.isArray(t)) {                    // pas nécessaire
      for (var i=0; i<t.length-1; ++i) {
          var m = argmin_take2(t,i);  // indice valeur minimale dans t sur [i,t.length[
          if (i !== m) swap(t,i,m);
      }

      return t;
    }
    // return undefined
};
```

- **argmin_take2** est dans *algo-tab.js*
- **swap** dans *tools.js*

Complexité

On mesure le nombre de comparaisons faites:

- 1*er* pas de boucle: *n-1* comparaisons
- 2*e* pas de boucle: *n-2* comparaisons

n(n-1)/2

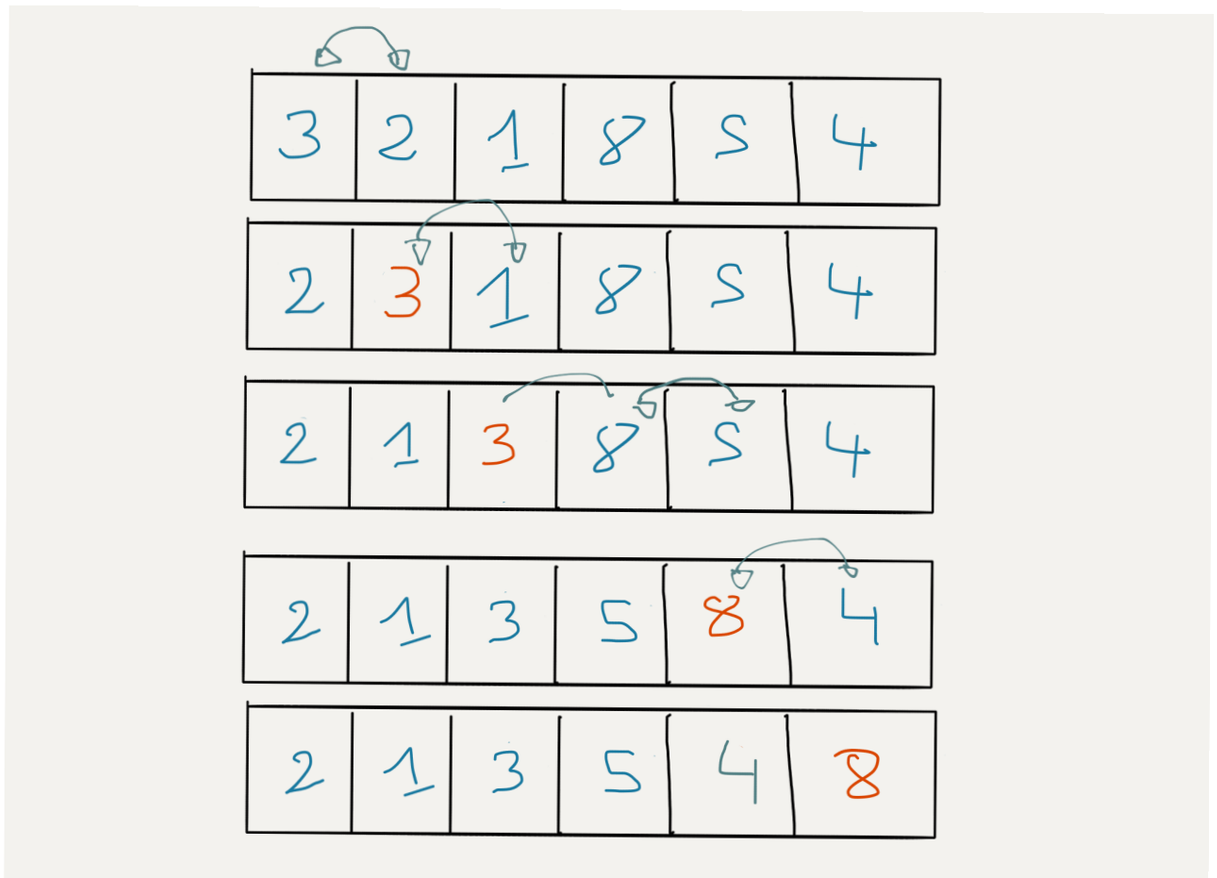
- i*e* pas de boucle: *n-i* comparaisons
- n-1*e* pas de boucle: *1* comparaison

On dit que l'algorithme est en $o(n^2)$
ou quadratique (en la taille *n* du tableau)

- Pros:
    - Si le tableau est (presque) trié: pas (ou peu) de swaps
- Cons:
    - Toujours la même complexité (que le tableau soit trié ou pas)

2-Tri à bulles

- lors d'un **passage** sur le tableau
  - échanger deux valeurs adjacentes qui ne sont pas dans le bon ordre
- recommencer si un échange a été effectué



1er passage

```
var triBulle = function(t) {

    if (Array.isArray(t)) {

      var echange, passage = 1;

      do {
          echange = false;

          for (var i=0; i<t.length-passage; ++i)
            if (t[i] > t[i+1]) {
                swap(t,i,i+1);
                echange = true;
            }

          ++passage;
      } while (echange);

      return t;
    }
};
```

swap dans *tools.js*

Complexité

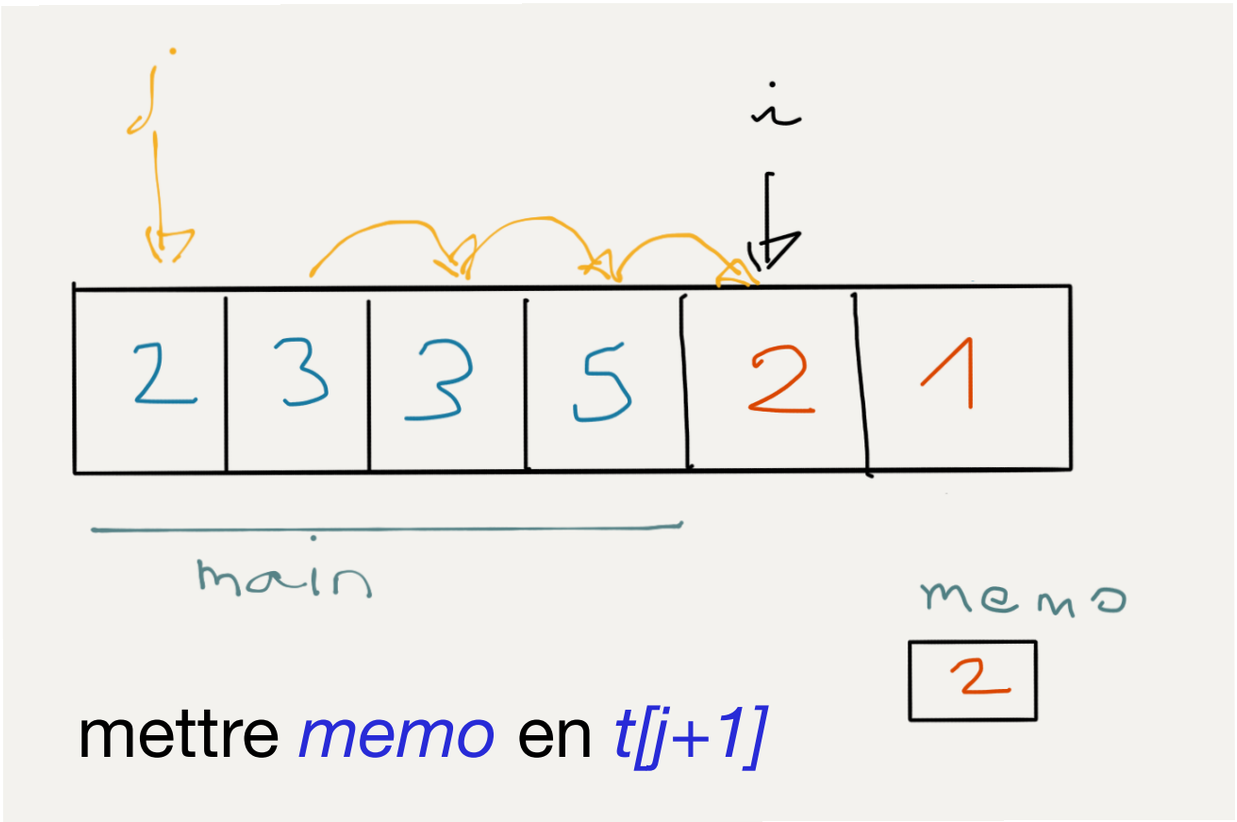
- Algorithme en o(n^2) dans le **pire des cas**
- Si le tableau est trié
  - un seul passage est effectué => o(n)

Note:

- Après un passage, la valeur la plus grande se trouve à la fin
- Il n'est donc pas nécessaire de considérer cette valeur ultérieurement
  - c'est l'usage de la variable *passage*

3- Tri par insertion

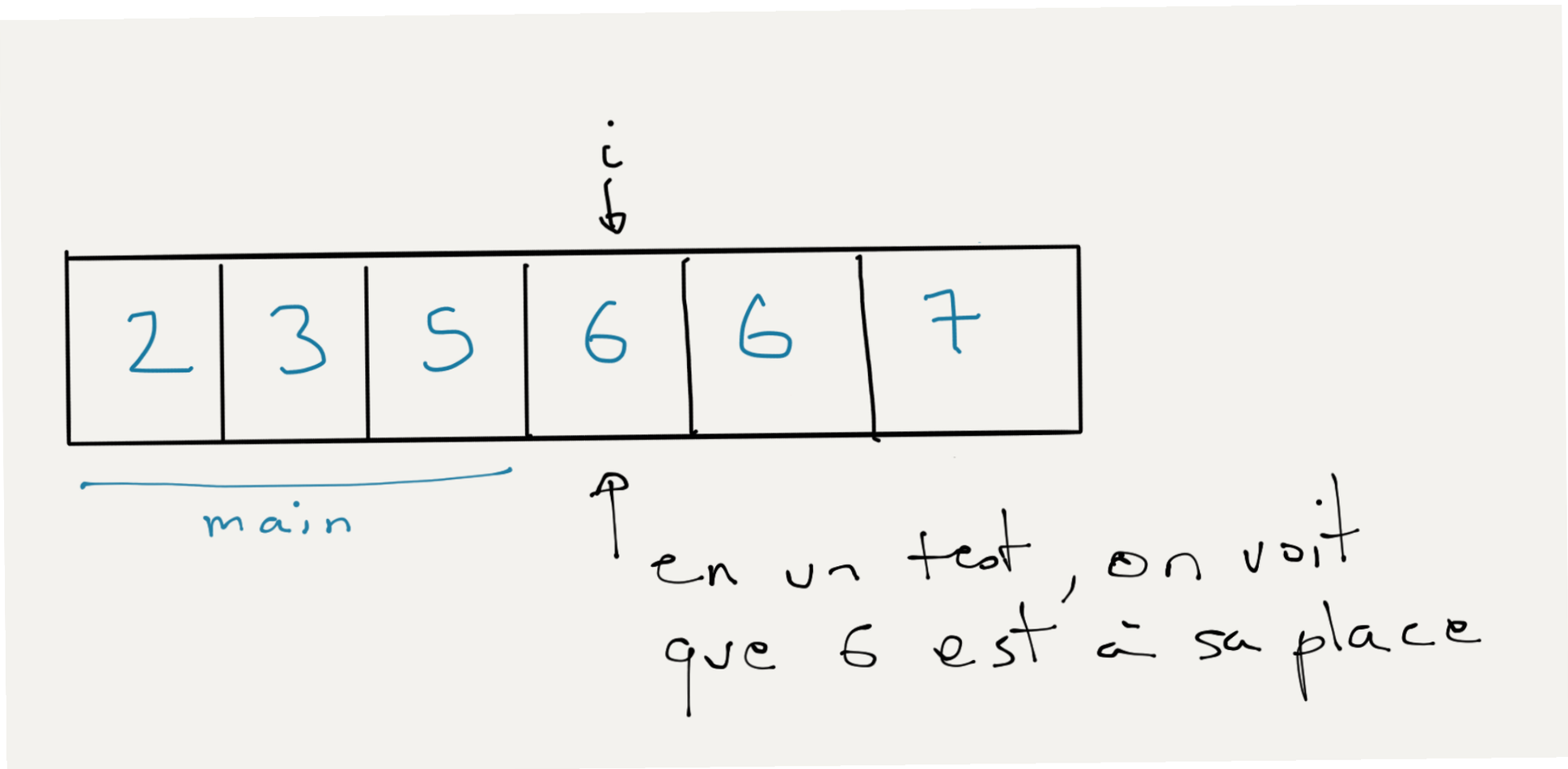
- Maintenir une **main** triée
  - Au début la **main** est vide
  - Augmente à chaque itération
  - [0,*i*[représente la **main**
- À l'itération *i*
  - Insérer au bon endroit *t[i]* dans la main par décalages successifs



```
var triInsertion = function(t) {

    if (Array.isArray(t)) {
      var memo,j;
      for (var i=1; i<t.length; ++i) {
          memo = t[i];
          for (j=i-1; (j >= 0) && (t[j] > memo); --j)
            t[j+1] = t[j];
          t[j+1] = memo;
      }

      return t;
    }

};
```

Complexité

- Quadratique — o(n2) —dans le pire cas
- Linéaire — o(n) si le tableau est trié (ou presque)



Tests unitaires

```
assert( isSorted(triSelection(aleaTab(100,2,100))) );
assert( egal( triInsertion([4,5,6,7,1,2,3,8,9]),
                triBulle([1,2,3,5,6,7,8,4,9])
            ));
```

aleaTab, isSorted, egal dans *algo-tab.js*

Note: code possible car **triInsertion**, **triBulle**, **triSelection** et **aleaTab** retournent l'adresse du tableau. **isSorted** et **egal** retournent un booléen.

Tests unitaires

```
var test_tri = function (nb) {

  var t = aleaTab(nb,2,51);  // tab de nb elts sur [2,51]

  triInsertion(t);
  assert( isSorted(t), "echec tri_insertion" );

  shuffle(t);
  triBulle(t);
  assert( isSorted(t), "echec tri_bulle" );

  shuffle(t);
  triSelection(t);
  assert( isSorted(t), "echec tri_selection" );
};
```

aleaTab, isSorted, shuffle dans *algo-tab.js*