Algorithmique et modélisation

Présentation du cours ALGO6 en L3 INFO

Jean-Marc Vincent[1]

[1] Laboratoire LIG
Équipe-Projet MESCAL
Jean-Marc.Vincent@imag.fr

Organisation : équipe pédagogique

TD 1

Nicolas Gast (LIG, Mescal)
TD1 coordination

Jean-Marc Vincent (LIG, Mescal)
Cours et TD1

TD 2

Gwenaël Delaval (LIG, NanoSim)
TD2 et Apnees coordination

Cyril Labbé (LIG, SIGMA)
TD2 et Apnees

Communication avec l'équipe pédagogique

Mail et adresses électroniques

Adresse Mail enseignant : Prénom.Nom@imag.fr
SUJET : [L3INFO :ALGO6] Cours/TD1/TD2/Apnée sujet explicite

envoyer votre mail avec votre adresse officielle **e.ujf-grenoble.fr**
toute adresse de provenance différente risque d'être "grey/black-listée" et d'atterrir dans une poubelle

le mail officiel de la L3-INFO est la liste **etu-2014-im2ag-l3info@ujf-grenoble.fr**, toute annonce officielle (quicks, apnées, déplacements de créneaux horaires,. . .) passera par ce mail (que vous devez lire quotidiennement)

Destinataires

cours/examens. . . : Jean-Marc Vincent

les **TD1** : Nicolas Gast ou Jean-Marc Vincent

les **TD2 et les Apnées** : Gwenael Delaval ou Cyril Labbé selon votre groupe

Supports pédagogiques

Sites web

Site web pédagogique :
`http://mescal.imag.fr/membres/jean-marc.vincent/index.html/ALGO6/`

Serveur Web : consulter le placard électronique de l'UFR

Objectif pédagogique de l'UE ALGO6

Savoir rattacher un problème à une classe de problèmes, en déduire une approche adaptée pour sa résolution algorithmique, valider la correction de la solution proposée, et en analyser sa complexité.

approche selon trois plans (ou points de vue)

- **raisonnement** informel mais rigoureux, liant la réalisation d'un algorithme à ses spécifications, raffinement d'un schéma d'algorithme vers une réalisation particulière ;
- **méthodes classiques** de résolution dont le critère principal est la complexité (algorithmes gloutons, diviser pour régner, programmation dynamique. . .) ;
- **types de problèmes classiques** (parcours de graphe, énumération d'un ensemble de candidats. . .), et comment l'expression d'une solution (itérative, récursive) est liée à la structure sous-jacente.

Organisation de la semaine

Cours : principes fondamentaux de l'algorithmique.
Le cours sera décomposé en 2 parties, une partie synthétique sur les concepts et une partie sur un algorithme classique mettant en oeuvre un schéma ou une méthode particuliers afin de se constituer une culture algorithmique de référence.

TD1 : Sous forme d'exercices sur feuille les TD1 permettent de renforcer la compréhension des concepts vus en cours.

TD2 : Les TD2 portent sur la mise en œuvre des concepts et préparent aux activités pratiques (structures de données en C, programmation).

APNEE : Les activités pratiques non encadrées permettent la validation des concepts et l'évaluation de la compréhension.

Travail personnel :
- prévoir 1 à 2h de travail à la maison pour 1h de cours ou TD (ici de 5 à 9h de travail),
- exercices à la maison (pour préparer quick et examen),
- programmation des exemples simples vus en cours/TD

Évaluation UE ALGO6

Contrôle continu :

- 2 quicks ou DM (semaines 6 et 9 (environ))
- Apnee : 5-6 comptes rendus

Examen : 3h sans document ni calculatrice

Coefficients :

- CC = $\frac{1}{2}$ moyenne(apnees) + $\frac{1}{2}$ moyenne(quicks)
  Toute absence ou devoir/apnee rendu hors délai ne sera pas évalué (note=0)
- Une note d'assiduité pourra être intégrée à la note de CC si nécessaire
- Note finale : voir le règlement d'examen

Session 2 : en juin

Contenu indicatif

Algorithmique et complexité

1. Complexité d'un problème — *Exponentiation*
2. Analyse en moyenne, Tables de Hachage (1) — *Algorithme de Rabin Karp*
3. Tables de Hachage (2) — *Bucket sort*
4. Randomisation — *Algorithme de Miller-Rabin*

Diviser pour régner et récursivité

5. Récursivité et énumération — *Parties d'un ensemble*
6. Programmation dynamique
7. Diviser pour régner et enveloppes convexes

Graphes et cheminements

8. Énumération de l'ensemble des chemins d'un graphe — *Algorithme de Dijkstra*
9. Approche algébrique pour explorer l'ensemble des chemin — *Algorithme de Danzig*

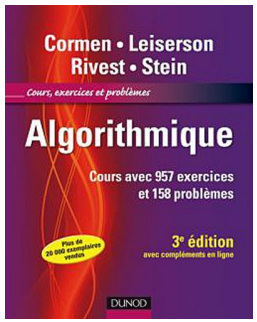
Exploration intelligente

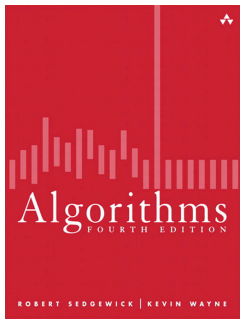
10. Exploration — *Algorithme de minimax*
11. Exploration (2) — *Algorithme alpha/beta*

Bibliographie : Ouvrages de référence du cours

- **Algorithmique** *Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein.*. Dunod, 2010.
  Ouvrage de référence internationale en algorithmique. Très pédagogique il peut être utilisé en autoformation, lorsque les bases sont acquises. Couvre l'ensemble du cours.
- **Algorithms** *Robert Sedgewick and Kevin Wayne*. Addison Wesley, 2011.
  Une approche thématique permettant de reprendre les différents et paradigmes de l'algorithmique. La présentation est soignée, les détails des implémentations en Java sont très utiles.

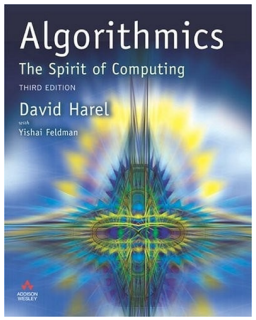
  Des versions précédentes en français : *Robert Sedgewick* Algorithmes en C ou Algorithmes en Java chez Dunod

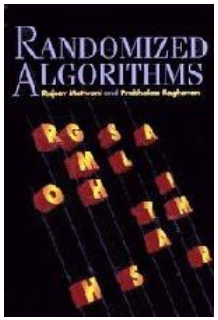




Bibliographie : Ouvrages plus avancés

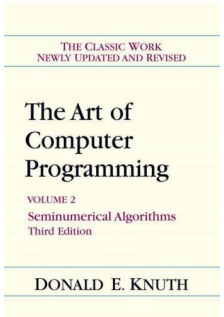
- **The Design and Analysis of Algorithms** *Dexter C. Kozen* Springer, 1991.
  Excellent ouvrage pour de l'algorithmique avancée. Présenté sous forme de séquence de lectures "indépendantes" il va directement à l'essentiel. Les principes algorithmiques sont ainsi mis en valeur.
- **Algorithmics : The Spirit of Computing** *David Harel and Yishai Feldman* Addison Wesley, 2004.
  Orienté méthodologie, cet ouvrage propose une vue transversale en abordant successivement, méthode et analyse, limitations et robustesse, extensibilité... intéressant pour le recul pris.
- **Introduction à l'analyse des algorithmes** *Robert Sedgewick and Philippe Flajolet* Addison Wesley 1995
  Ouvrage théorique sur l'analyse de la complexité des algorithmes
- **Randomized Algorithms**, *R. Motwani and P. Raghavan*, Cambridge University Press, 1995.

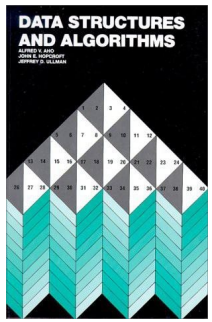




Bibliographie : Ouvrages historiques de référence

- **The Art of Computer Programming, Vol 1-4** *Donald E. Knuth*, Addison-Wesley, 1998.
  Ouvrage historique et encore d'actualité pour la conception et l'analyse d'algorithmes
- **Data Structures and Algorithms** *Alfred V. Aho, J.E. Hopcroft, et Jeffrey D. Ullman* Addison Wesley 1983
- *Jean-Luc Chabert et al.* **Histoires d'algorithmes** Belin 2010
  Une histoire des algorithmes avec un point de vue calcul et calcul numérique





Vous avez dit *algorithme* ?

PRÉPARATION 10 min
CUISSON 15 min
POUR **12 madeleines**

100 g de farine
3 g de levure chimique
100 g de beurre
1/4 de citron non traité
2 œufs
120 g de sucre en poudre

Madeleines

1 Tamisez ensemble la farine et la levure au-dessus d'un bol.
2 Faites fondre le beurre dans une petite casserole et laissez-le refroidir.
3 Hachez finement le zeste du 1/4 de citron.
4 Cassez les œufs dans une terrine, versez le sucre par-dessus. Fouettez pendant 5 minutes pour bien les faire mousser ; ajoutez le mélange farine-levure en pluie, puis le beurre et le zeste haché, sans cesser de tourner.
5 Préchauffez le four à 220 °C.
6 Beurrez légèrement la plaque à madeleines et remplissez-la de pâte seulement aux deux tiers. Mettez au four pendant 5 minutes à 220 °C puis baissez la température à 200 °c et laissez cuire encore 10 minutes.
7 Démoulez les madeleines tièdes et laissez-les refroidir.

Extrait du Larousse des Desserts par Pierre Hermé

Informellement :
un **algorithme** est un processus systématique pour réaliser un objectif.

Un peu d'histoire : les premiers algorithmes

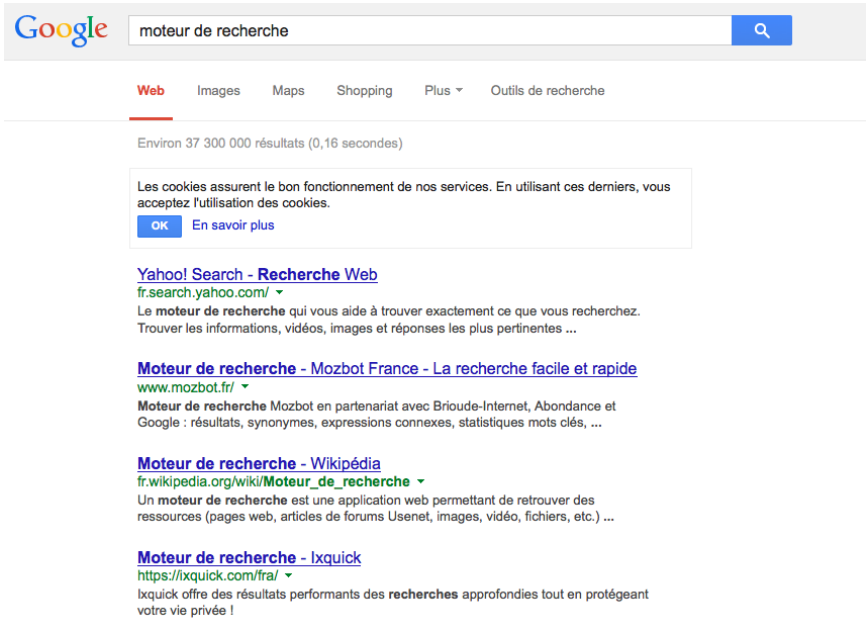
Euclide d'Alexandrie
III$^{\text{ième}}$ siècle avant JC.

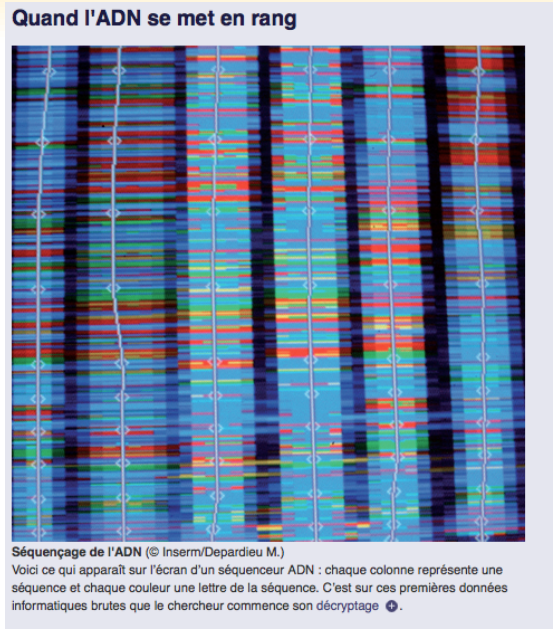
Calcul sur les entiers
$PGCD(a, b)$

Muhammad ibn Musa al-Khwarizmi
IX$^{\text{ième}}$ siècle.

Calcul algébrique
$5 = 3.x + 2$

À quoi ça sert ?





http://interstices.info/decoder-vivant

Derrière les algorithmes

Un algorithmes est conçu pour **résoudre** un (des) problème(s) :

- un problème dont on peut trouver une solution en temps "raisonnable" (polynomial en la taille des entrées)
- pour les autres : l'idée intuitive est qu'il est plus facile de vérifier la solution d'un problème plutôt que le résoudre...
- et dans ce cas proposer des solutions approchées au problème

Un algorithme est un objet de **communication**

- systématisation : dans un langage de référence
- à niveau d'abstraction
- dans un contexte permettant l'analyse (preuve et complexité)
- et opérationnel (permettant la programmation)

Expression d'un algorithme

Le langage algorithmique est une convention qui permet d'exprimer à un **lecteur**

1. l'idée de l'algorithme (principe, déroulement,...)
2. lui permettre de faire la preuve de celui-ci et de pouvoir analyser sa complexité
3. de pouvoir le traduire facilement dans un langage de programmation

Le langage algorithmique est donc plus ou moins proche d'un langage de programmation.
Exemple : tri par insertion

Tri par insertion : TD d'algorithmique

Une itération ($i = 2$ à n) à chaque pas de laquelle on insère à sa place l'élément d'indice i dans la séquence triée formée des $i - 1$ premiers éléments.

Initialement : l'élément 1 forme une séquence triée.

Finalement : les n éléments sont triés.

On effectue l'insertion par une recherche séquentielle de l'emplacement k de l'élément i, et un décalage vers la droite des éléments de k à $i - 1$.

L'algorithme classique effectue ces deux opérations ensemble, c'est-à-dire décale l'élément i vers la gauche (par un échange) jusqu'à ce qu'il atteigne sa "bonne" place.

Tri par insertion : Ouvrage de Cormen et al.

```
TRI-INSERTION(A)
1  pour j ← 2 à longueur[A]
2      faire clé ← A[j]
3           ▷ Insère A[j] dans la séquence triée A[1 . . j − 1].
4           i ← j − 1
5           tant que i > 0 et A[i] > clé
6               faire A[i + 1] ← A[i]
7                    i ← i − 1
8           A[i + 1] ← clé
```

seule l'idée est donnée,

il n'y a pas de déclaration de variables,

il est implicite que A est un tableau,

clé l'élément qui permet la comparaison, i,j des indices de tableau,...

Tri par insertion : Ouvrage de Sedgewick et al.

ALGORITHM 2.2 Insertion sort

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {  // Sort a[] into increasing order.
      int N = a.length;
      for (int i = 1; i < N; i++)
      {  // Insert a[i] among a[i-1], a[i-2], a[i-3]... ..
         for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
      }
   }
   // See page 245 for less(), exch(), isSorted(), and main().
}
```

parti pris de décrire les algorithmes dans un langage de programmation ici Java

abstraction des opérateurs (less à la place de <),

syntaxe et propriétés du langage de programmation.

Tri par insertion : Ouvrage de Denenberg et al.

procedure *InsertionSort*(**table** $A[0\,..\,n-1]$)**:**
{Sort by inserting each item in position in the table of elements to its left}
    for i **from** 1 **to** $n-1$ **do**
        $j \leftarrow i$ {j scans to the left to find where $A[i]$ belongs}
        $x \leftarrow A[i]$
        while $j \geq 1$ **and** $A[j-1] > x$ **do**
            $A[j] \leftarrow A[j-1]$
            $j \leftarrow j-1$
        $A[j] \leftarrow x$

Algorithm 11.1 Insertion Sort.

mélange des approches

symboles et syntaxe spécifique

Tri par insertion : Ouvrage de Knuth

5.2.1. Sorting by Insertion

One of the important families of sorting techniques is based on the "bridge player" method mentioned near the beginning of Section 5.2: Before examining record R_j, we assume that the preceding records $R_1, \ldots, R_{j-1}$ have already been sorted; then we insert R_j into its proper place among the previously sorted records. Several interesting variations on this basic theme are possible.

Straight insertion. The simplest insertion sort is the most obvious one. Assume that $1 < j \leq N$ and that records $R_1, \ldots, R_{j-1}$ have been rearranged so that

$$K_1 \leq K_2 \leq \cdots \leq K_{j-1}.$$

(Remember that, throughout this chapter, K_j denotes the key portion of R_j.) We compare the new key K_j with K_{j-1}, K_{j-2}, ..., in turn, until discovering that R_j should be inserted between records R_i and R_{i+1}; then we move records R_{i+1}, ..., R_{j-1} up one space and put the new record into position $i+1$. It is convenient to combine the comparison and moving operations, interleaving them as shown in the following algorithm; since R_j "settles to its proper level" this method of sorting has often been called the *sifting* or *sinking* technique.

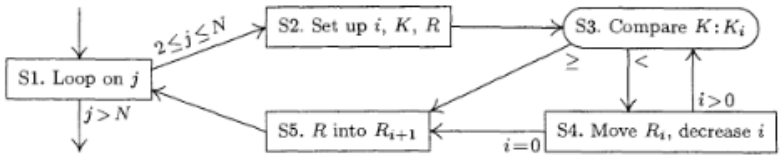


Fig. 10. Algorithm S: Straight insertion.

Algorithm S (*Straight insertion sort*). Records $R_1, \ldots, R_N$ are rearranged in place; after sorting is complete, their keys will be in order, $K_1 \leq \cdots \leq K_N$.

S1. [Loop on j.] Perform steps S2 through S5 for $j = 2, 3, \ldots, N$; then terminate the algorithm.

S2. [Set up i, K, R.] Set $i \leftarrow j-1$, $K \leftarrow K_j$, $R \leftarrow R_j$. (In the following steps we will attempt to insert R into the correct position, by comparing K with K_i for decreasing values of i.)

S3. [Compare $K : K_i$.] If $K \geq K_i$, go to step S5. (We have found the desired position for record R.)

S4. [Move R_i, decrease i.] Set $R_{i+1} \leftarrow R_i$, then $i \leftarrow i-1$. If $i > 0$, go back to step S3. (If $i = 0$, K is the smallest key found so far, so record R belongs in position 1.)

S5. [R into R_{i+1}.] Set $R_{i+1} \leftarrow R$. ∎

représentation graphique

schéma itératif (steps), non modulaire

Tri par insertion : Ouvrage de Aho et al.

Insertion Sorting

The second sorting method we shall consider is called "insertion sort," because on the i^{th} pass we "insert" the i^{th} element $A[i]$ into its rightful place among $A[1], A[2], \ldots, A[i-1]$, which were previously placed in sorted order. After doing this insertion, the records occupying $A[1], \ldots, A[i]$ are in sorted order. That is, we execute

```
for i := 2 to n do
    move A[i] forward to the position j ≤ i such that
        A[i] < A[k] for j ≤ k < i, and
        either A[i] ≥ A[j−1] or j = 1
```

To make the process of moving $A[i]$ easier, it helps to introduce an element $A[0]$, whose key has a value smaller than that of any key among $A[1], \ldots, A[n]$. We shall postulate the existence of a constant $-\infty$ of type keytype that is smaller than the key of any record that could appear in practice. If no constant $-\infty$ can be used safely, we must, when deciding whether to push $A[i]$ up before position j, check first whether $j = 1$, and if not, compare $A[i]$ (which is now in position j) with $A[j-1]$. The complete program is shown in Fig. 8.5.

```
(1)   A[0].key := −∞;
(2)   for i := 2 to n do begin
(3)       j := i;
(4)       while A[j] < A[j−1] do begin
(5)           swap(A[j], A[j−1]);
(6)           j := j−1
          end
      end
```

Fig. 8.5. Insertion sort.

Tri par insertion : Wikipedia Fr - En

En français

```
procédure tri_insertion(tableau T, entier n)
    pour i de 1 à n-1
        x ← T[i]
        j ← i
        tant que j > 0 et T[j - 1] > x
            T[j] ← T[j - 1]
            j ← j - 1
        fin tant que
        T[j] ← x
   fin pour
fin procédure
```

En anglais version1

```
for i ← 1 to length(A) - 1
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
```

version 2

```
for i = 1 to length(A) - 1
   x = A[i]
   j = i
   while j > 0 and A[j-1] > x
       A[j] = A[j-1]
       j = j - 1
   A[j] = x
```

Autres sites

http://openclassrooms.com/courses/le-tri-par-insertion
http://rosettacode.org/wiki/Sorting_algorithms/Insertion_sort
http://www.sorting-algorithms.com/insertion-sort

Principes pour l'expression des algorithmes

Ma position personnelle (que certains de mes collègues ne partagent pas) :

- utiliser les conventions
  (i,j sont des indices, x un réel, n un entier par exemple une taille de tableau,. . .).
- la forme est importante
  l'indentation doit permettre de comprendre la structure de l'algorithme
- mettre des commentaires dans le code et accompagner l'algorithme par une explication en français
- ne mettre que l'information importante
  minimiser l'encre
- être cohérent
  utiliser le même formalisme pour toutes les présentations d'algorithmes

de manière plus générale

favoriser tout ce qui aide à la compréhension et éliminer ce qui peut perturber la lecture