Cours 2 : Méthodes d'analyse des algorithmes récursifs - Illustration sur les algorithmes de tri

Jean-Stéphane Varré

Université Lille 1

jean-stephane.varre@lifl.fr

Rappels

Algorithme récursif :

- un algorithme qui se rappelle lui-même
- type de récursivité
  - simple ou linéaire (par exemple factorielle, tours de Hanoï)
  - croisée ou mutuelle (par exemple test de parité)
- on peut dessiner les appels grâce à un arbre

Rappels

Conception :

- tout algorithme récursif doit distinguer plusieurs cas
- il doit y avoir au moins un cas qui ne comporte pas d'appel récursif : les cas de base
  (par exemple lorsque le tableau a 0 éléments)
- définir les bons cas de base : ils doivent être atteignables quelque soit l'exemplaire en entrée
  (par exemple en s'assurant que le tableau dans l'appel récursif est plus petit)

> Permet souvent d'exprimer de manière simple la résolution d'un problème

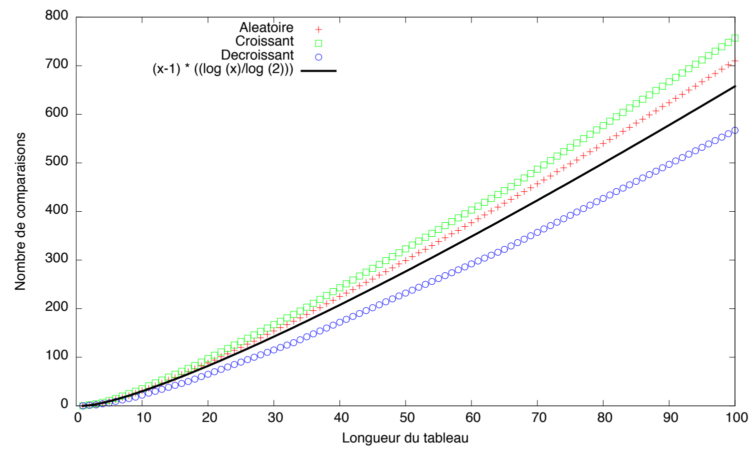
Le tri par insertion dichotomique

- principe, exemple
- code de l'insertion (la partie tri ne change pas)

```
(* insertion de t.(i) dans la tranche t.(0..i) *)
let inserer_dichotomique t i =
  let pos = rechercher_position_rec t 0 (i-1) t.(i)
  and aux = t.(i) in
  (* decalage des valeurs *)
  for k = (i-1) downto pos do
    t.(k+1) <- t.(k);
  done;
  (* placement de la nouvelle valeur *)
  t.(pos) <- aux
```

```
(* recherche de la position d'insertion de elt
   dans la tranche t.(gauche..droite) *)
let rec rechercher_position_rec t gauche droite elt =
  if gauche >= droite - 1 then begin
    (* cas de base *)
    nb_cmp := !nb_cmp + 1;
    if elt < t.(gauche) then
      gauche
    else begin
      nb_cmp := !nb_cmp + 1;
      if elt < t.(droite) then
        droite
      else
        droite + 1
    end
  end else
    (* cas general *)
    let m = (gauche + droite) / 2
    in
    nb_cmp := !nb_cmp + 1;
    if elt < t.(m) then
      rechercher_position_rec t gauche m elt
    else
      rechercher_position_rec t m droite elt
```

Tri par insertion dichotomique - analyse



Les divisions entières par 2

Il arrive fréquemment que les équations de récurrences soient du type :

$$c(n) = c\left(\frac{n}{2}\right) + 1$$

Théorème

Soit $c(n)$ une fonction croissante de IN vers IR. Si, pour tout n assez grand de la forme $n = 2^p$, $c(n) = \mathcal{O}(n^\alpha (\log n)^\beta)$ avec $\alpha \geq 0$ et $\beta \geq 0$, alors l'égalité reste vraie pour tout n assez grand.

Cela signifie que si on résout l'équation pour des n de la forme 2^p alors la solution sera valable pour n'importe quel n, pourvu qu'il soit grand.

Le tri fusion

- principe, exemple
- code

```
let rec tri_fusion t =
  let n = Array.length t
  in
  if n = 1 then
    (* cas de base *)
    Array.copy t
  else
    (* cas general *)
    let t1 = tri_fusion (Array.sub t 0 ((n-1)/2+1))
    and t2 = tri_fusion (Array.sub t ((n-1)/2+1) ((n-1)-((n-1)/2+1)+1))
    in
    fusionner t1 t2
```

```
let fusionner t1 t2 =
  let n1 = (Array.length t1)
  and n2 = (Array.length t2) in
  let t = Array.concat [ t1; t2]
  and i = ref 0 and j = ref 0 and k = ref 0 in
  while !i < n1 && !j < n2 do
    nb_cmp := !nb_cmp + 1;
    if t1.(!i) < t2.(!j) then begin
      t.(!k) <- t1.(!i);
      i := !i + 1
    end else begin
      t.(!k) <- t2.(!j);
      j := !j + 1
    end;
    k := !k + 1
  done;
  while !i < n1 do
      t.(!k) <- t1.(!i); i := !i + 1; k := !k + 1
  done;
  while !j < n2 do
      t.(!k) <- t2.(!j); j := !j + 1; k := !k + 1
  done;
  t
```

Tri fusion - analyse