Méthodes efficaces de tri

tri par sélection du maximum

extension type liste
opérations

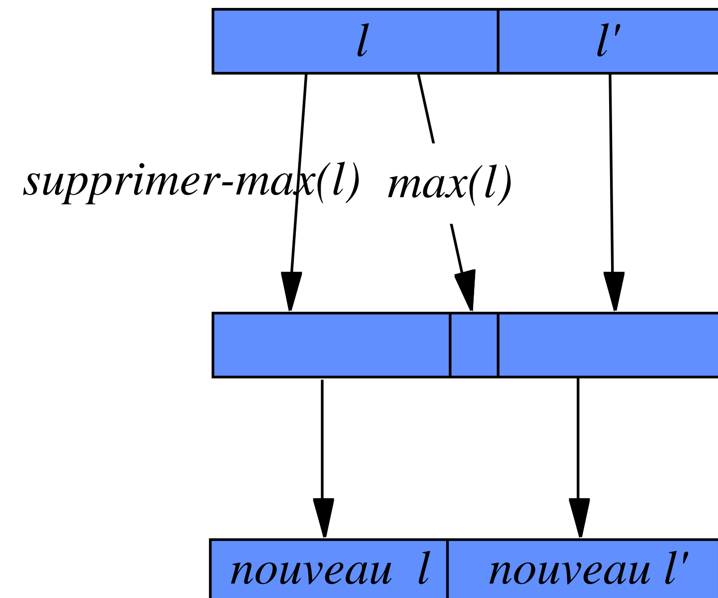
max : liste / élément
supprimer-max : liste / liste
tri-sélect-iter : liste × liste liste
tri-sélect : liste liste

préconditions

max (l): ¬ estvide (l)
supprimer-max (l): ¬ estvide (l)

sémantique

e: élément, l, l': liste
e l e max (l)
est-permut (max(l) :: supprimer-max(l), l)
tri-sélect-iter (l', l) = **si** l = listevide **alors** l'
sinon tri-sélect-iter(max(l) :: l', supprimer-max(l)) **fsi**



tri par tas (1)

- Structurer les éléments pour optimiser les opérations *max* et *supprimer-max*
- un tas:

  arbre binaire parfait   représentation par niveau

  tout nœud est plus grands que tous ses descendants   *max* à la racine

  o   fils-gauche(o) et o   fils-droit(o)

- toute liste est la représentation d'un arbre binaire parfait:

  racine est en ième(`l`, 1)

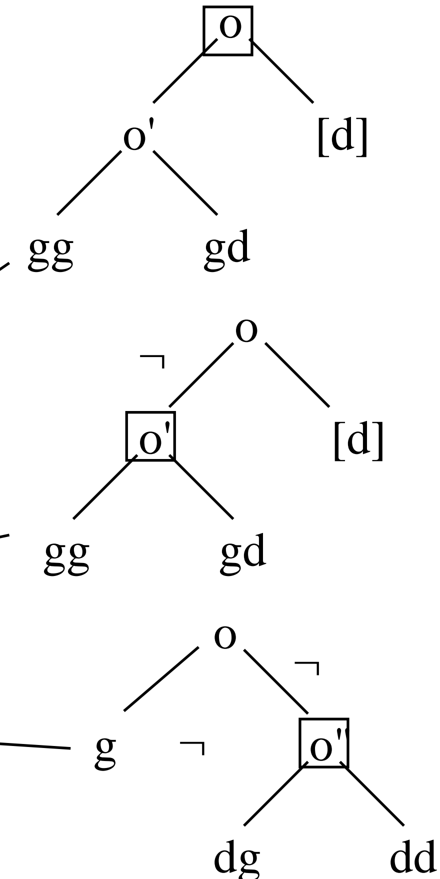
  nœud x en i, son père est en i/2

  nœud x en i, son fils gauche en 2*i si 2*i   longueur(`l`)

  nœud x en i, son fils droit en 2*i+1 si 2*i+1   Longueur(`l`)

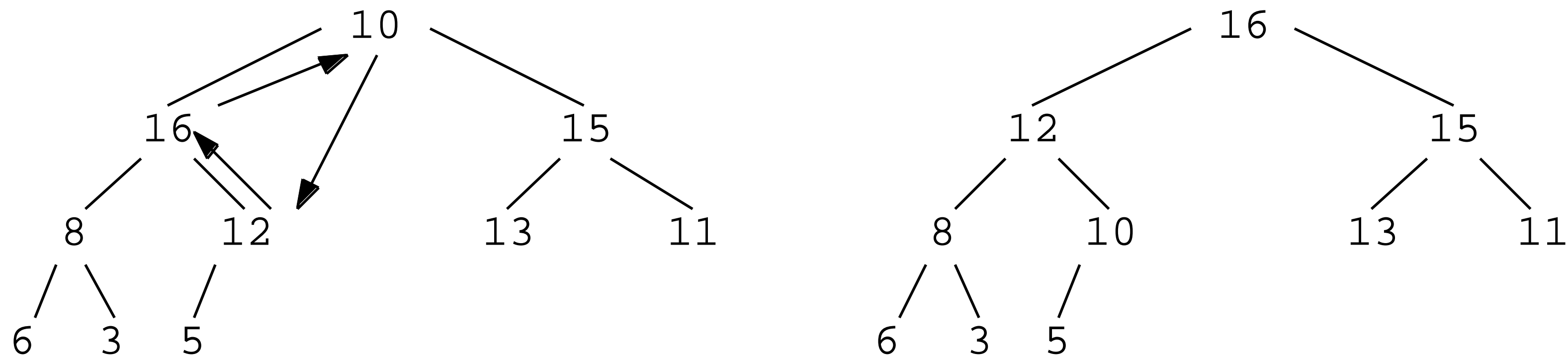
Tri par tas (2)

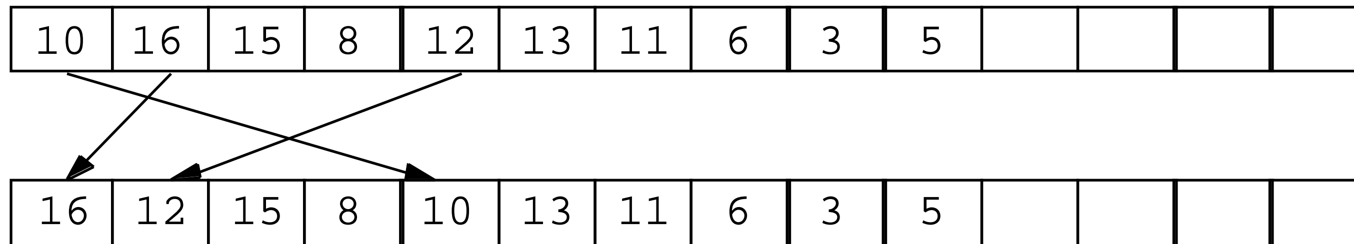
extension type arbre
    ordonner : arbre / arbre
préconditions
    ordonner (a): a Ø (g(a) = Ø d(a) = Ø)
sémantique
    ordonner (a) =
        si a = Ø a = <o, Ø, Ø> **alors** a
        sinon soit a = <o, <o', gg, gd>, d>;
            si o o' (d = Ø o racine(d)) **alors** a
            sinsi ¬ o o' (d = Ø o' racine(d)) **alors**
                <o', ordonner(<o, gg, gd>), d>
            sinon soit a = <o, g, <o", dg, dd>>;
                <o", g, ordonner(<o, dg, dd>)>
            fsi
        fsi



si g(a) et d(a) partiellement ordonnés, ordonner(a) l'est

Tri par tas (3)





- transformation de la liste en tas:

  appliquer ordonner sur les nœuds en remontant depuis les feuilles jusque la racine

Tri par tas (4)


```
procedure Ordonner (En, Max : Indice) is
    Le_Fils : Indice; Le_Pere : Indice := En;
    U Element := La_Liste.Ieme (En);
begin
    while 2 * Le_Pere <= Max loop                   -- Il y a un fils
        Le_Fils := 2 * Le_Pere;                     -- Choisir le plus grand fils
        if Le_Fils < Max and then
            La_Liste.Ieme(Le_Fils) < La_Liste.Ieme(Le_Fils+1) then
            Le_Fils := Le_Fils + 1;
        end if;
        exit when La_Liste.Ieme (Le_Fils) < U;      -- bien placé
        La_Liste.Ieme (Le_Pere) := La_Liste.Ieme (Le_Fils);
        Le_Pere := Le_Fils;           -- remonter le fils et poursuivre en dessous
    end loop;
    La_Liste.Ieme (Le_Pere) := U;                   -- place finale
end Ordonner;
```

Tri par tas (5)


```
for K in reverse 1..La_Liste.Longueur/2 loop
    Ordonner (K, La_Liste.Longueur);
end loop;

for K in reverse 2 .. La_Liste.Longueur loop     -- extraction du tas
    Echanger (1, K);                             -- le plus grand et la dernière feuille
    Ordonner (1, K - 1);          -- ordonner avec la dernière feuille à la racine
end loop;
```

- comparaisons: $2 \sum_{K=2}^{n} \log_2(K-1)$ (n log n)
- transferts: 1/2 comparaisons (n log n)
- place mémoire supplémentaire: (1)

Tri rapide (1)

extension type liste

opérations

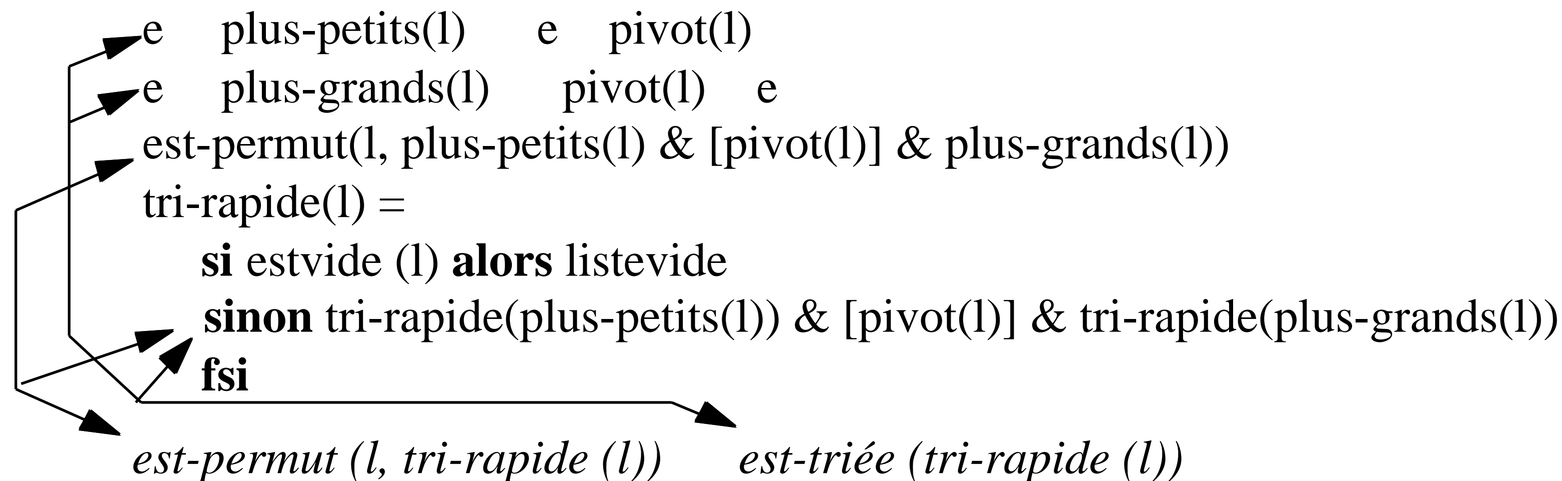
plus-petits : liste / liste
pivot : liste / élément
plus-grands : liste / liste
tri-rapide : liste liste

préconditions

plus-petits(l): ¬ estvide(l)
pivot(l): ¬ estvide(l)
plus-grands(l): ¬ estvide(l)

sémantique

e: élément, l: liste

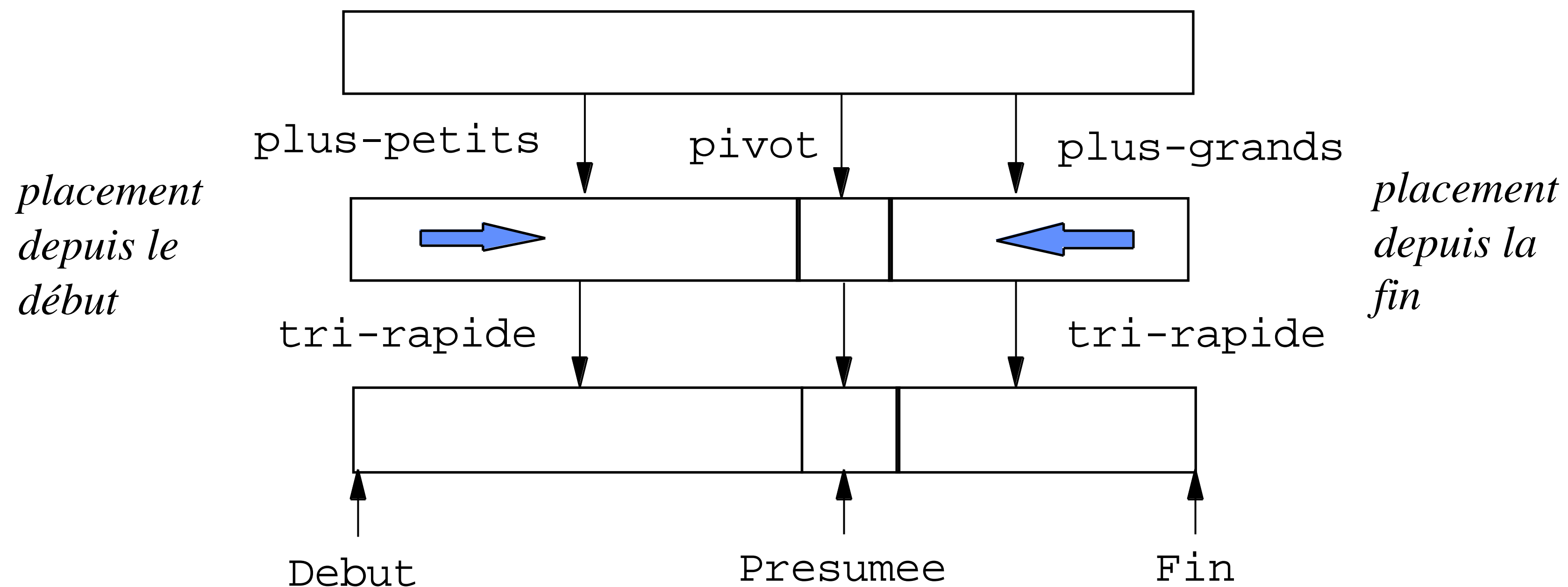


e plus-petits(l) e pivot(l)
e plus-grands(l) pivot(l) e
est-permut(l, plus-petits(l) & [pivot(l)] & plus-grands(l))
tri-rapide(l) =
si estvide (l) **alors** listevide
sinon tri-rapide(plus-petits(l)) & [pivot(l)] & tri-rapide(plus-grands(l))
fsi

est-permut (l, tri-rapide (l)) *est-triée (tri-rapide (l))*

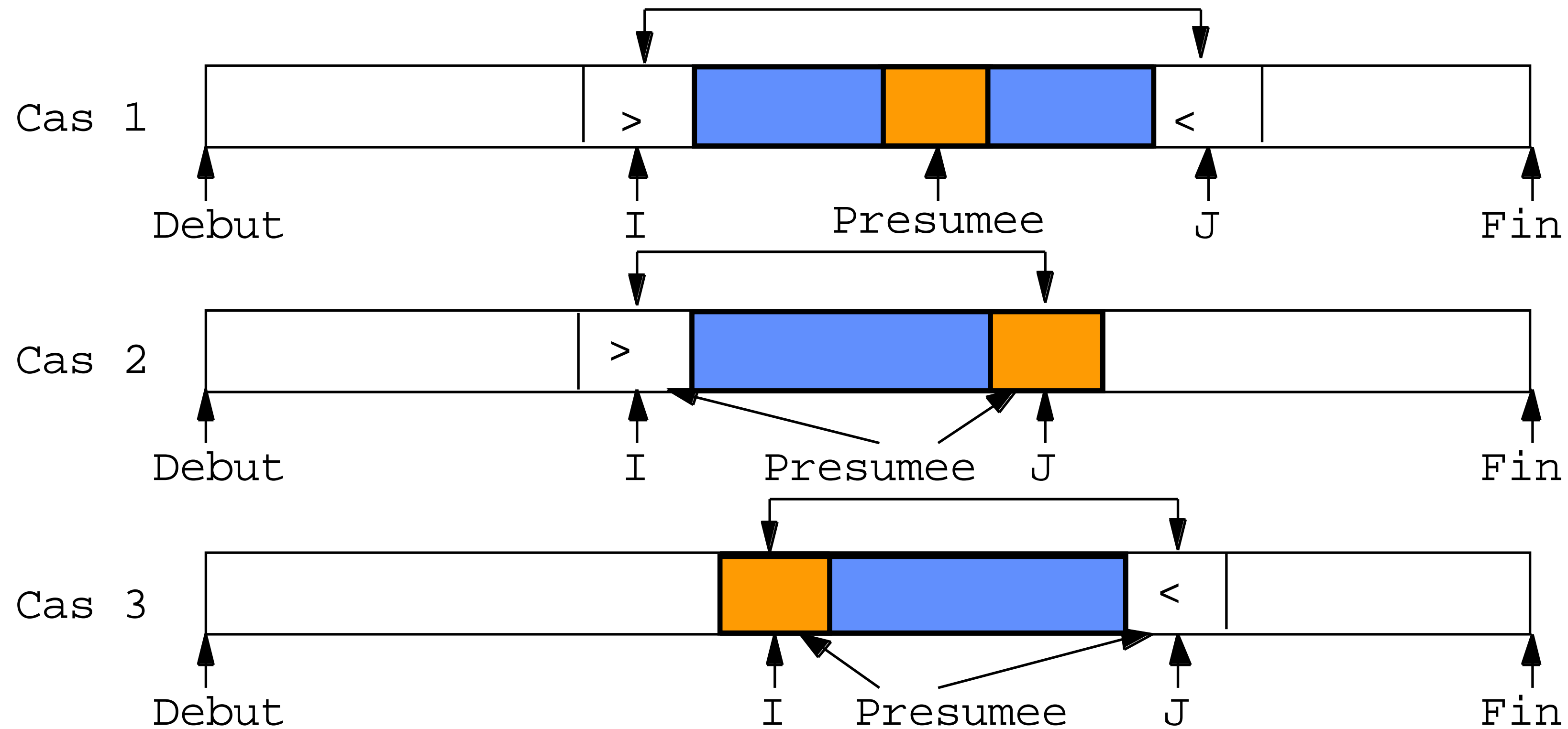
Tri rapide (2)

- implantation de la liste résultat dans l'espace origine
- initialement, on ne connaît pas la longueur des listes



Tri rapide (3)

- trois cas de la constructions des sous-listes



procédure de Répartition

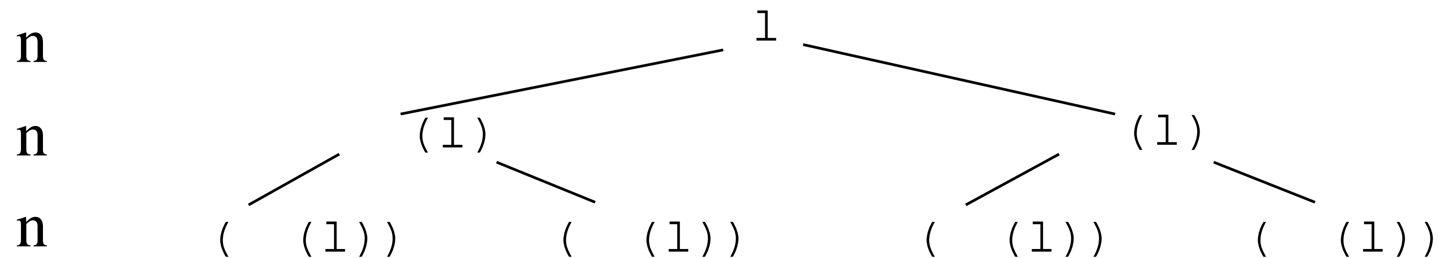
- Choisir le pivot
  - Ici, le médian parmi le premier, celui situé au milieu et le dernier
- Classer les éléments entre Debut et Fin
  - Debut .. Presumee -1 les plus-petits
  - Presumee reçoit le pivot
  - Presumee+1 .. Fin les plus-grands
- Au retour, recommencer sur
  - [Debut .. Presumee-1]
  - [Presumee+1 .. Fin]
- Sauver un des deux couples pour faire la liste plus tard
  - la plus petite d'abord => la pile est limitée à $\log_2 n$

Procédure principale


```
Debut := 1;  Fin := La_Liste.Longueur;
loop
   while Fin - Debut > 1 loop
       Repartition;                    -- couper la liste en deux
       if Presumee-Debut < Fin-Presumee then  -- plus petite d'abord
          Empiler ((Presumee+1, Fin)); Fin := Presumee - 1; -- c'est celle du début
       else
          Empiler((Debut, Presumee-1)); Debut := Presumee+1; -- celle de la fin
       end if;
   end loop;                          -- jusqu'à ce que la liste soit suffisamment petite
   if Fin = Debut + 1 and then La_Liste.Ieme(Fin) < La_Liste.Ieme(Debut) then
     Echanger (Debut, Fin);
   end if;
   exit when Est_Vide;                                 -- toutes les listes sont faites
   Cour := Sommet; Depiler;        -- Debut := Cour.Debut; Fin := Cour.Fin
end loop;
```

Complexité

- Répartition sur k élément: k comparaisons, 3k/2 transferts
- listes sur lesquelles on applique Répartition:



- comparaisons: au pire n h(A)

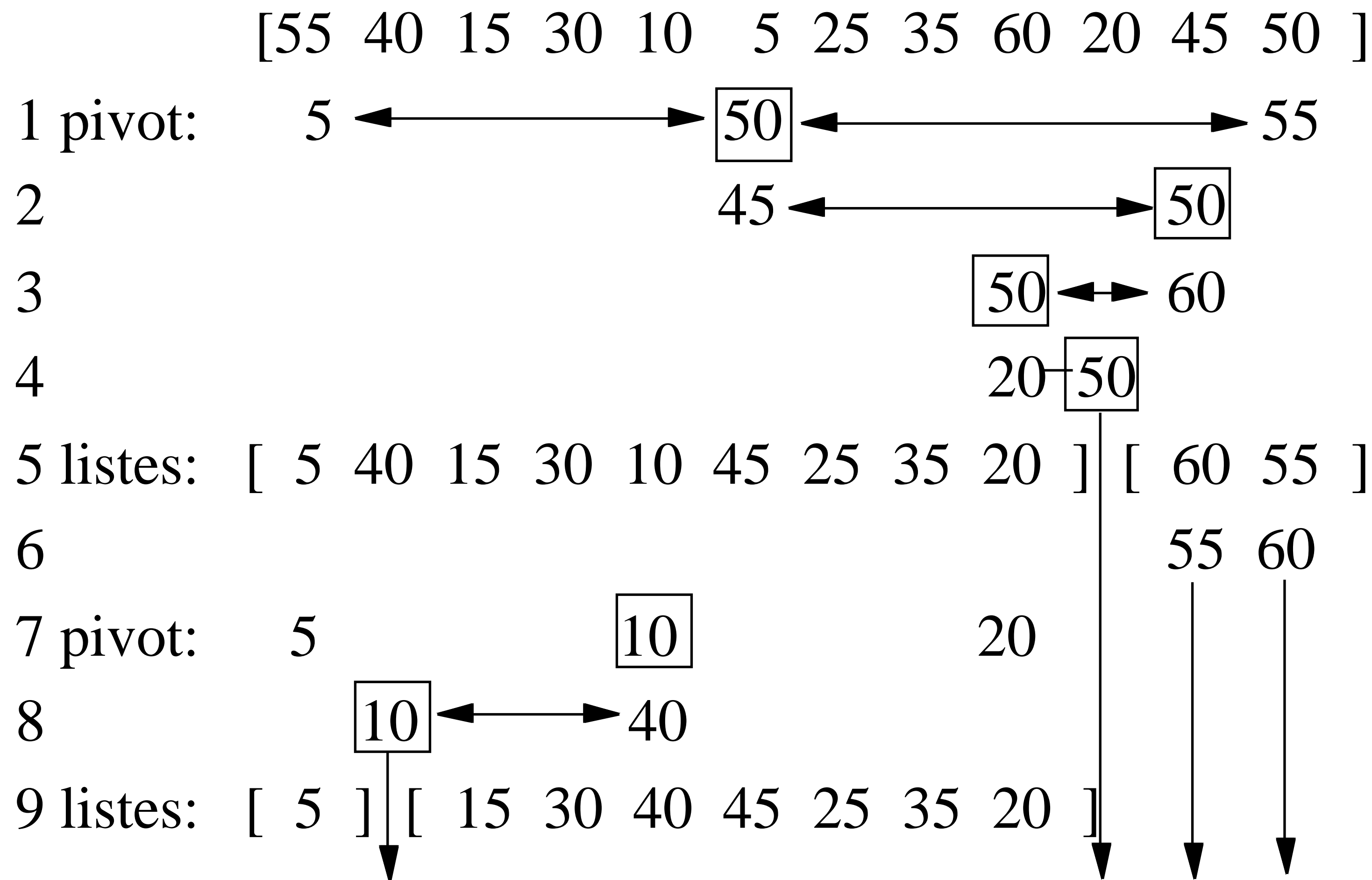
  un nœud = un pivot donc au plus n nœuds

  au pire (n^2)

  en moyenne et au mieux (n log n)

- place mémoire (log n)

Exemple de tri rapide (1)



Exemple de tri rapide (2)