Chapitre 6

Tris rapides

Option informatique MPSI1 & MPSI2

Nous allons étudier dans ce chapitre des méthodes de tris plus sophistiquées que celles vues dans un précédent chapitre. Cette complication de l'écriture permettra une rapidité d'exécution meilleure.

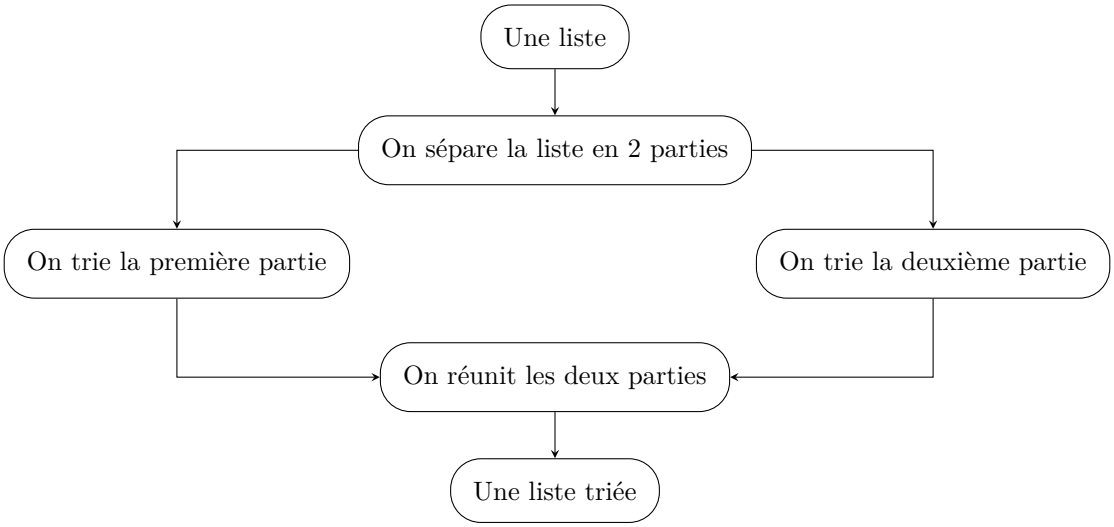
I Structures récursive des tris

On peut réunir les les deux algorithmes de tri de liste déjà vus sous la forme

1. on sépare un élément du reste : le plus petit dans le cas du tri par sélection, la tête de la liste dans le cas du tri par insertion,
2. on trie les éléments restants de manière récursive,
3. on ajoute l'élément séparé : c'est la création directe d'une liste dans le cas du tri par sélection, c'est l'insertion dans le cas du tri par insertion.

On peut généraliser en utilisant une méthode diviser pour régner :

1. On sépare en deux parties.
2. On trie chaque partie de manière récursive
3. On assemble les deux parties triées.



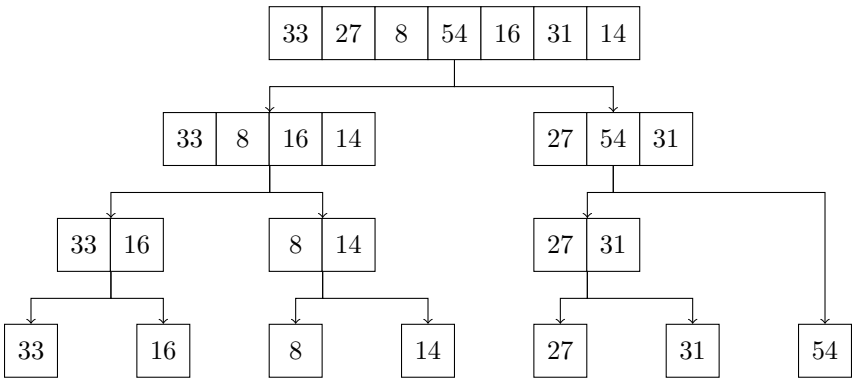

```
let rec tri liste =
  if casTerminal
  then traitementDirect
  else let l1, l2  = separation liste in
       let t1 = tri l1 in
       let t2 = tri l2 in
       assemblage t1 t2;;
```

Les deux tris que nous allons étudier diffèrent dans leur manière de séparer la liste initiale.

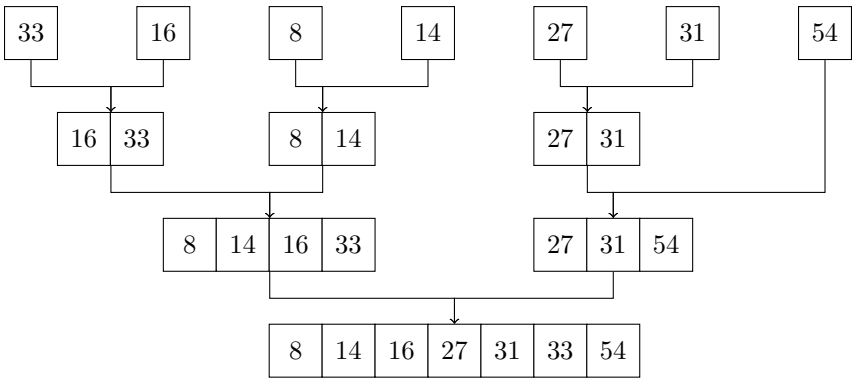
II Tri fusion d'une liste

Le tri-fusion sépare la liste en deux parts presque égales, sans critère. On envoie alternativement les éléments dans les listes qui reçoivent les éléments ; la complexité est linéaire.

```
let rec separation liste =
  match liste with
  |[] -> [],[]
  |[a] -> [a], []
  |a::(b::q) -> let l1, l2 = separation q in
                (a::l1), (b::l2);;
```



Quand on parvient à une liste de taille 0 ou 1, on obtient une liste triée et on assemble alors les morceaux triés.



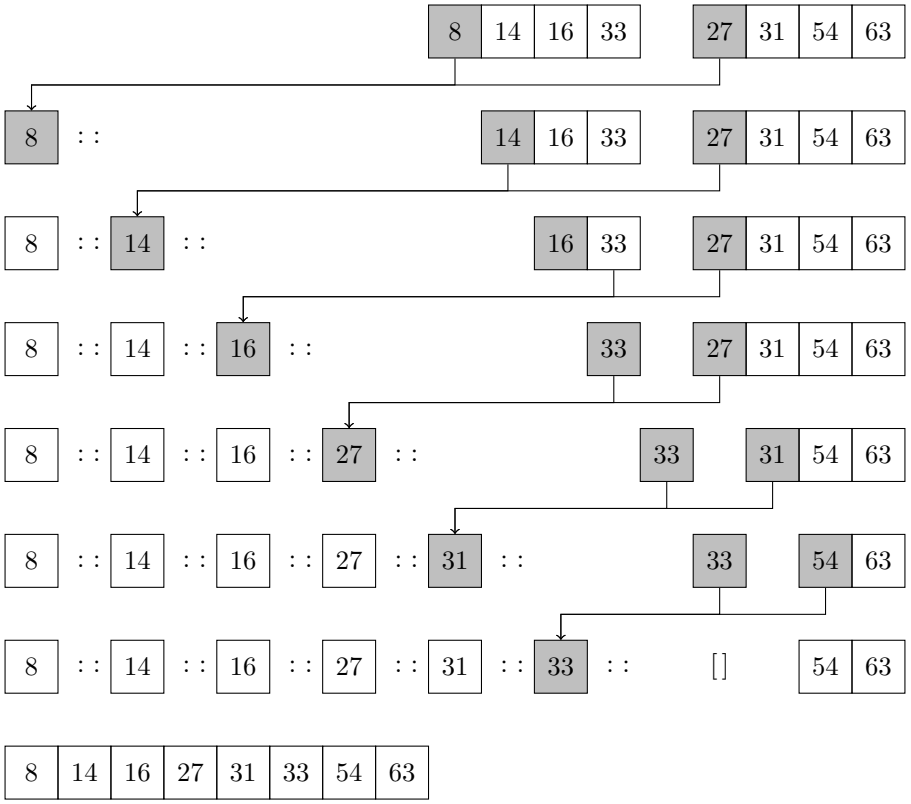
On doit, à chaque étape fusionner deux listes triées obtenues récursivement.
Dans un algorithme récursif, on a besoin la tête du résultat et de la queue.

- La tête est le plus petit élément, ce doit être une des deux têtes des listes triées.
- Il reste alors à fusionner le reste de la liste dont on a gardé la tête et l'autre liste, intacte.
- On ne fait cette opération que lorsqu'aucune liste est vide, sinon la fusion d'une liste et d'une liste vide renvoie la liste non vide.

```
let rec fusion l1 l2 =
  match l1,l2 with
  |[], _ -> l2
  |_, [] -> l1
  |t1::q1, t2::q2 -> if plusGrand t2 t1
                    then t1::(fusion q1 l2)
                    else t2::(fusion l1 q2);;
```

La fonction de tri découle de l'introduction.

```
let rec triFusion liste =
  match liste with
  |[] -> []
  |[x] -> [x]
  |_ -> let l1, l2 = separation liste in
        let t1 = triFusion l1 in
        let t2 = triFusion l2 in
        fusion t1 t2;;
```



II.1 Analyse du tri fusion

Terminaison

La fonction `separation` termine pour des listes de taille 0 ou 1 et, sinon, est appelée récursivement en diminuant la taille de 2. Elle termine donc en un temps fini.
Dans la fonction `fusion` la somme des longueurs des deux listes diminue strictement lors d'un appel récursif, c'est un variant donc la fonction termine.
La fonction `triFusion` termine car elle fait appel à des fonctions qui terminent et, lors des appels récursifs, reçoit une liste de longueur strictement inférieure comme paramètre.

Preuve

La fonction de tri doit renvoyer une liste avec les mêmes éléments placés dans l'ordre croissant.
Il faut donc prouver que `separation` renvoie deux liste qui contiennent exactement tous les éléments de la liste initiale ; cela se fait par une récurrence simple.
On prouve ensuite par récurrence sur la somme des tailles des listes que `fusion` appliquée à deux listes triées renvoie une liste triée avec les mêmes éléments.

- C'est évident si la somme des longueur est nulle car alors les deux listes sont vides et `fusion` renvoie une liste vide, qui est triée.
- On suppose prouvé le résultat dans le cas de listes dont la somme des tailles est n.
  On considère deux listes de somme de longueurs égale à $n+1$.
  — Si l'une est vide la fonction renvoie l'autre qui est triée et contient tous les éléments.
  — Si aucune n'est vide alors l'élément en tête est un minorant de chaque liste et le minimum des deux têtes est un minorant de l'union.
  On suppose `l1 = t1::q1` et `l2 = t2::q2` avec t1 <= t2.
  On applique la fonction récursivement à `q1` et `t2::q2` dont la somme des tailles est n donc l'hypothèse de récurrence implique que le résultat est trié et contient tous les éléments de l'union, c'est-à-dire tous les éléments de `l1` et `l2` sauf `t1`. En ajoutant `t1` en tête on obtient bien tous les éléments du départ dans une liste triée car `t1` est un minorant et le reste est trié.

La preuve du tri se fait alors par récurrence (généralisée) sur la taille des listes à trier.

Complexité

- La séparation ne comporte aucune comparaison
- La fusion effectue une comparaison par étape tant que les deux listes ne sont pas vides. Le dernier élément placé n'est pas comparé donc la fusion effectue au plus $n-1$ comparaisons.
- La séparation d'une liste de taille n donne deux listes de tailles n_1 et n_2 ; de plus, si on a $n \leqslant 2^p$ alors $n_1 \leqslant 2^{p-1}$ et $n_2 \leqslant 2^{p-1}$.
  On note u_p le nombre maximal de comparaisons effectuées lors du tri d'une liste de taille 2^p au plus, on a alors $u_p \leqslant u_{p-1} + u_{p-1} + n - 1 \leqslant 2u_{p-1} + 2^p$. et $u_0 = 0$
  Si on pose $v_p = 2^{-p}u_p$ on a $v_p \leqslant v_{p-1} + 1$ et $v_0 = 0$ donc $v_p \leqslant p$ puis $u_p \leqslant 2^p.p$.
  Pour une liste de taille n avec $2^{p-1} < n \leqslant 2^p$, le nombre de comparaisons dans le tri est majoré par $p \cdot 2^p \leqslant 2 \cdot n \cdot \big(\log_2(n) + 1\big)$.

Théorème 1 : Complexité du tri fusion

Le tri fusion d'une liste de taille n a une complexité en $\mathcal{O}\big(n \cdot \log_2(n)\big)$ dans le pire des cas.

On va calculer plus précisément le nombre maximal de comparaisons nécessaire pour trier une liste de n éléments, on le note $C(n)$. On note $n \div 2$ le quotient entier de n par 2.
On a $n \div 2 = \left\lfloor \frac{n}{2} \right\rfloor$ et $n - n \div 2 = \left\lceil \frac{n}{2} \right\rceil$.

Exercice 1 - Formule de récurrence
Montrer que $C(n) = C(n \div 2) + C(n - n \div 2) + n - 1$ avec $C(1) = 0$.

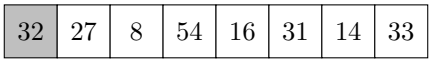
Exercice 2 - Simplification
On pose $D(n) = C(n+1) - C(n)$; prouver que $D(n) = D(n \div 2) + 1$ avec $D(1) = 1$.

Exercice 3 - Calcul des incréments
Prouver que $D(n) = p + 1$ pour $2^p \leqslant n < 2^{p+1}$.

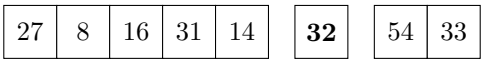
Exercice 4 - Téléscopage
Prouver que $C(n) = n \cdot \lfloor \log_2(n) \rfloor + n + 1 - 2^{\lfloor \log_2(n) \rfloor + 1}$.

III Tri pivot d'une liste

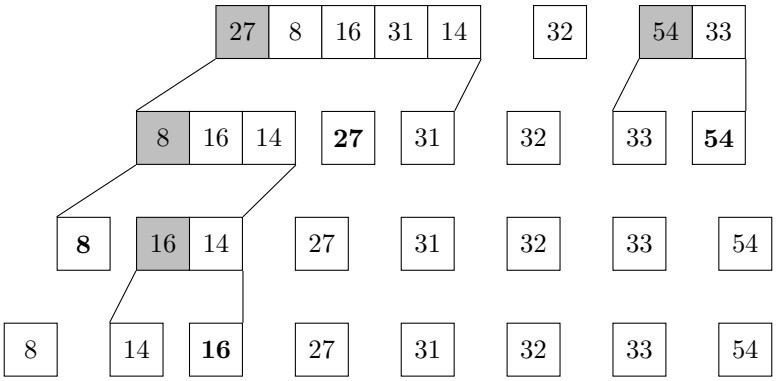
Le tri pivot (QUICKSORT en anglais) procède autrement : plutôt que fusionner deux sous-listes triées qui viennent d'un découpage arbitraire, on découpe intelligemment en séparant les petites valeurs et les grandes. Pour cela on sélectionne un terme de la liste, le *pivot*, et on crée deux listes, l'une contenant les termes inférieurs au pivot, l'autre les termes supérieurs. Contrairement au tri-fusion, on n'est passuré que les deux listes soient de taille semblables.
Le choix le plus simple pour le pivot est le premier terme de la liste.



peut donner, avec le premier terme comme pivot,



On peut continer récursivement, en s'arrêtant aux listes vides ou celle de taille 1.
On voit, qu'à la fin, il suffit d'assembler les morceaux.



Le découpage est assez simple dans le cas des listes

```
let rec decoupageSelon liste pivot =
  match liste with
  |[] -> [], []
  |t::q -> let petit, grand = decoupageSelon q pivot in
           if plusGrand pivot t
           then t::petit, grand
           else petit, t::grand;;
```

On a choisit ici de mettre les termes égaux au pivot fans la liste des grands.
Le tri suit alors l'algorithme décrit

```
let rec triPivot liste =
  match liste with
  |[] -> []
  |[a] -> [a]
  |t::q -> let l1, l2 = decoupageSelon q t in
           let t1 = triPivot l1 in
           let t2 = triPivot l2 in
           t1 @ (t :: t2);;
```

III.1 Analyse du tri pivot

Terminaison

La longueur de la liste diminue à chaque appel de `decoupageSelon` donc la fonction termine en un temps fini.
Lors de l'appel récursif dans le tri pivot, les liste filtrées ont une longueur strictement inférieure car elles ne contiennent pas le terme pivot. La terminaison se prouve alors par récurrence sur la longueur de la liste.

Preuve de `decoupageSelon`

On prouve par récurrence sur la longueur de la liste que `decoupageSelon liste pivot` renvoie deux listes `petit` et `grand` telles que `petit` contient les éléments de la liste originale qui sont strictement inférieurs au pivot avec la même multiplicité et `grand` contient les éléments de la liste originale qui sont supérieurs ou égaux au pivot avec la même multiplicité.

- C'est évident dans le cas d'une liste vide.
- On suppose que la propriété est vraie pour les liste de taille n.
  `liste = t::q` est de taille $n+1$
  Lors de l'exécution de `filtrage liste pivot`, l'appel récursif de `filtrage q pivot` concerne une liste de taille n. L'hypothèse de récurrence implique que `filtrage q pivot` renvoie deux liste qui contiennent tous les éléments de `q` avec la même multiplicité et qui sont séparés par le pivot. On ajoute alors l'élément manquant de la liste initiale dans la liste adaptée ce qui prouve la propriété pour les liste de taille $n+1$.

Preuve de `triPivot`

La fonction de tri doit renvoyer une liste avec les mêmes éléments placés dans l'ordre croissant.
C'est bien entendu le cas quand la liste initiale est vide.
On suppose que la propriété est vérifiée pour toutes les listes de taille inférieure ou égale à n. On considère une liste de taille $n+1$: `liste = t::q`.

1. `filtrage q t` renvoie deux liste `l1` et `l2` telles que `l1` contient les éléments de `q` qui sont strictement inférieurs à `t` avec la même multiplicité et `l2` contient les éléments de `q` qui sont supérieurs ou égaux à `t` avec la même multiplicité.
2. D'après l'hypothèse de récurrence, `t2` contient les éléments de `l2` triés. Il y a donc les éléments de `q` qui sont supérieurs ou égaux à `t` avec la même multiplicité et triés.
3. Ainsi `t :: t2` contient les éléments de la liste initiale qui sont supérieurs ou égaux à `t` avec la même multiplicité sous forme triée.
4. D'après l'hypothèse de récurrence, `t1` contient les éléments de `l1` triés. Il y a donc les éléments de `q` qui sont strictement inférieurs à `t` avec la même multiplicité et triés.
5. On conclut que `t1 @ (t :: t2)` est formée des éléments de la liste initiale avec la même multiplicité et triés.

Complexité

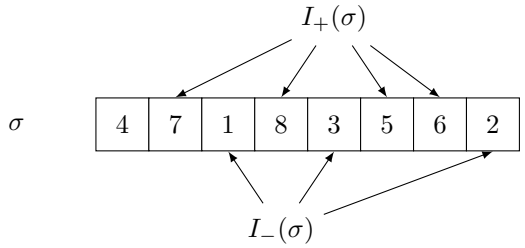
Dans la fonction `filtrage pivot liste` on fait une comparaison par élément de `liste` donc la complexité est la longueur de `liste`.
On note `C(liste)` le nombre de comparaisons dans le tri de `liste`.
Comme il n'y a pas de comparaisons dans l'assemblage des listes on conclut que, si `|liste|` est la longueur d'une liste et si `decoupageSelon pivot l` renvoie les listes `lg` et `ld` on a
`C([]) = 0` et `C(l) = |l| - 1 +C(lg)+ C(ld)`. Si on suppose que `liste` est formée de n éléments en ordre strictement croissant alors la fonction `filtrage t q` renverra toujours le couple `[], q`. La formule de la complexité devient alors `C(t::q)= |liste| - 1 + 0 + C(q)`. Dans ce cas on obtient `C(liste)` $= \sum_{k=1}^{n} k-1 = \frac{n(n-1)}{2}$, la complexité, dans le pire des cas, est quadratique.

III.2 Complexité moyenne du tri pivot

$C(\ell)$ est le nombre de comparaisons effectuées lors du tri de la liste ℓ.
On a $C([]) = 0$ et $C(\ell) = |\ell| - 1 + C(\ell_g) + C(\ell_d)$ où ℓ_g et ℓ_g sont les sous-listes de gauche et de droite obtenues après le filtrage de ℓ.
Pour calculer la complexité moyenne on ne va considérer que les différentes listes obtenues avec les permutations d'une liste initiale $[a_1; a_2; \cdots; a_n]$.
On supposera qu'on a $a_1 < a_2 < \cdots < a_n$ et on admet que les calculs sont indépendants des valeurs des a_i. On note $\ell_\sigma = [a_{\sigma(1)}; a_{\sigma(2)}; \cdots; a_{\sigma(n)}]$

La complexité moyenne est donc définie par $\overline{C}(n) = \dfrac{1}{n!} \displaystyle\sum_{\sigma \in \mathfrak{S}_n} C(\ell_\sigma)$ et on pose $\overline{C}(0) = 0$.

Lors de la fonction de découpage, le pivot est le premier élément, $k = \sigma(1)$.
Le découpage de la liste crée deux listes $\ell_{\sigma,g}$ et $\ell_{\sigma,d}$ avec $\ell_{\sigma,g}$ qui est une permutation de $[a_1; a_2; \cdots; a_{k-1}]$ et $\ell_{\sigma,d}$ qui est une permutation de $[a_{k+1}; a_{k+2}; \cdots; a_n]$.
Le valeurs $a_1, a_2, \ldots, a_{k-1}$ décrivent un sous-ensemble $I_-(\sigma) = \sigma^{-1}\big(\{1, 2, \ldots, k-1\}\big) \subset \{2, 3, \ldots, n\}$ de même, $a_{k+1}, \ldots, a_n$ décrivent $I_+(\sigma) = \sigma^{-1}\big(\{k+1, \ldots, n\}\big) = \{2, 3, \ldots, n\} \setminus I_-(\sigma)$.



Pour tout $k \in \{1, 2, \ldots, n\}$ et pour tout sous-ensemble A de cardinal $k-1$ inclus dans $\{2, 3, \ldots, n\}$ on note $\mathfrak{S}_n(k, A)$ l'ensemble des permutations de $\sigma \in \{1, 2, \ldots, n\}$ telles que
$\sigma(1) = l$, $I_-(\sigma) = A$ et $I_+(\sigma) = \{2, 3, \ldots, n\} \setminus A$.

Exercice 5 - Regroupements
Déterminer le cardinal de $\mathfrak{S}_n(k, A)$. Prouver que
$$\sum_{\sigma \in \mathfrak{S}_n(k,A)} C(\ell_{\sigma,g}) = (n-k)! \cdot (k-1)! \cdot \overline{C}(k-1) \quad \text{et} \quad \sum_{\sigma \in \mathfrak{S}_n(k,A)} C(\ell_{\sigma,d}) = (n-k)! \cdot (k-1)! \cdot \overline{C}(n-k)$$

Indication : on peut remarquer (sans démonstration) que les permutations dans $\ell_{\sigma,g}$ et $\ell_{\sigma,d}$ ne dépendent que de la distribution des petites et des grandes valeurs relativement à k donc de A

Exercice 6 - Récurrence
En déduire que $\overline{C}(n) = n - 1 + \dfrac{2}{n} \cdot \displaystyle\sum_{i=0}^{n-1} \overline{C}(i)$ puis $n \cdot \overline{C}(n) = (n+1) \cdot \overline{C}(n-1) + 2 \cdot (n-1)$.
Conclure que $\overline{C}(n) = 2 \cdot (n+1) \cdot \left(\displaystyle\sum_{k=1}^{n} \frac{1}{k} \right) - 4 \cdot n$.

Théorème 2 : Complexité du tri pivot

Le tri pivot d'une liste de taille n a une complexité en $\mathcal{O}\big(n^2\big)$ dans le pire des cas et une complexité en $\mathcal{O}\big(n \ln(n)\big)$ en moyenne.

IV Tri pivot d'un tableau

La traduction du tri par pivot dans un tableau peut se faire en place, cela donne un tri efficace qui est très employé. On l'écrit en employant une fonction auxiliaire récursive qui trie entre deux bornes et que l'on appelle avec les bornes 0 et $n-1$.
Pour trier entre i et j

- on découpe entre i et j avec une fonction qui choisit le terme pivot, et permute les valeurs du tableau entre i et j en plaçant les termes inférieurs au pivot puis le pivot puis les termes plus grands. La fonction renvoie la position du pivot, k
- on trie alors entre i et $k-1$ (le pivot est à sa place) et entre $k+1$ et j.
- Le cas d'arrêt est celui où $i = j$, un seul terme, ou $i > j$, aucun terme à trier

```
let triPivot t =
  let rec auxPivot a b =
    if b > a
    then begin
      let k = separation t a b in
      auxPivot a (k-1);
      auxPivot (k+1) b end in
  auxPivot 0 (Array.length t - 1);;
```

Tout le travail est fait dans la séparation.

IV.1 Séparation

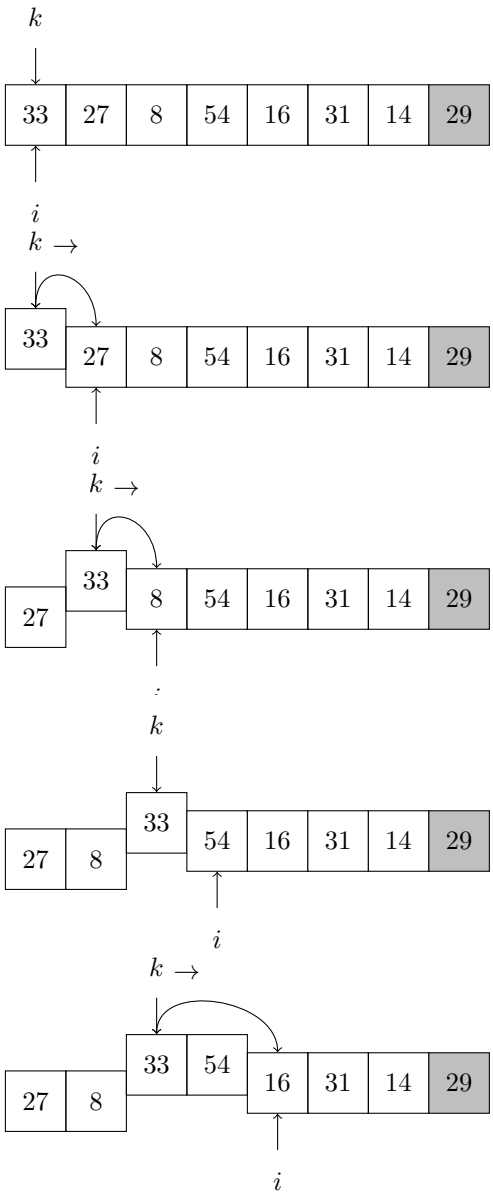
Pour la séparation entre a et b

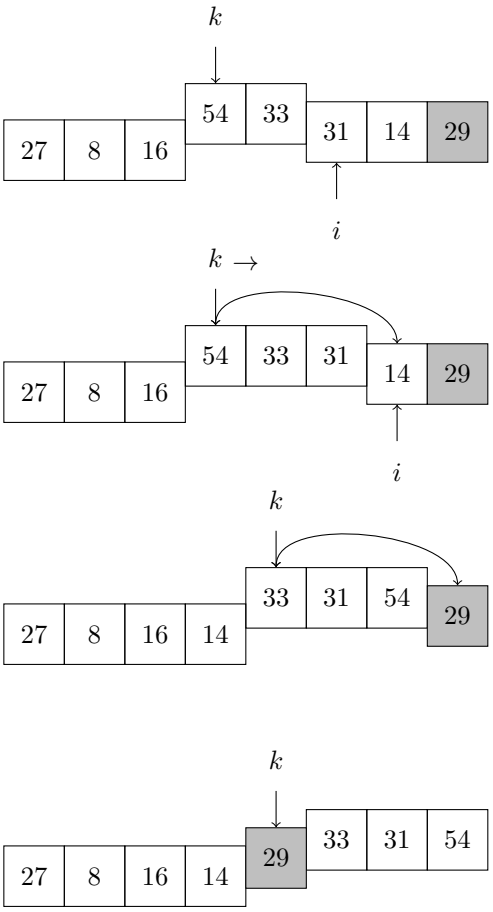
- on choisit la pivot en b, et on parcourt le tableau de a à $b-1$ (`for i`)
- on maintient une variable mutable k qui sépare les petits éléments des grands, sa valeur intitiale est a. Il permet de définir un invariant :
  - les éléments entre a et $k-1$ sont inférieurs au pivot
  - les éléments entre k et $i-1$ sont supérieurs au pivot
- le maintien de l'invariant impose
  - d'échanger les éléments en i et k si `t.(i)` est inférieur au pivot puis incrémenter k
  - de ne rien faire sinon,
- à la fin de la boucle l'invariant donne, pour $i = b$,
  - les éléments entre a et $k-1$ sont inférieurs au pivot
  - les éléments entre k et $b-1$ sont supérieurs au pivot
- on échange alors es éléments en i et k pour placer le pivot.

À retenir

On a vu que la complexité du tri pivot était mauvaise dans le cas de listes presque triées, c'est la cas aussi pour les tableau. Un solution pour éviter cela est de choisir un pivot aléatoirement entre a et b puis d'effectuer un échange pour le placer en b.

Dans la représentation suivante on abaisse les éléments que l'on sait inférieur au pivot et on éléve ceux qui sont testés supérieurs.






```
let separation t a b =
  let k = ref a in
  let pivot = t.(b) in
  for i = a to (b-1) do
    if plusGrand pivot t.(i)
    then begin
      echanger t !k i;
      incr k end done;
  echanger t !k b;
  !k;;
```

V Médiane

Lors d'une étude d'un ensemble de valeurs on souhaite souvent trouver une valeur représentative. On utilise souvent la moyenne à cet effet.
Cependant il y a quelques inconvénients :

- la moyenne n'est que rarement un élément de l'ensemble
- elle est sensible aux valeurs erratiques : une suite de 99 termes de valeurs 1 et 1 terme de valeur 100 aura une moyenne de 1,99, ce qui peut être peu signifiant.

Un autre outil possible est la médiane : dans une liste de n termes on cherche le terme de rang $\left\lfloor \frac{n+1}{2} \right\rfloor$.
Dans l'ensemble $\{3, 12, 5, 8, 17, 9, 13\}$ la médiane est 9 ;
dans l'ensemble $\{3, 12, 5, 8, 17, 9, 4, 13\}$ la médiane est 8 (on aurait pu choisir 9).

Pour résoudre le problème on va le généraliser :
trouver le terme de rang p dans un ensemble
Une méthode immédiate consiste à trier l'ensemble puis à déterminer le terme de rang p.
On va améliorer cela (en moyenne) en utilisant le découpage du tri pivot. En effet celui-ci partitionne un ensemble de taille n en trois sous-ensembles :

- le pivot,
- les éléments de l'ensemble strictement inférieurs au pivot, de taille k avec $0 \leqslant k \leqslant n-1$,
- les autres éléments (ils sont supérieurs ou égaux au pivot, il y en a $n-k-1$).

Il y a alors 3 cas selon la valeur de k :

- si $k = p-1$ le pivot est la valeur recherchée,
- si $p \leqslant k$ on cherche le p-ième élément dans le premier sous-ensemble,
- si $k < p-1$ on cherche le $p-1-k$-ième élément dans le second sous-ensemble.

Exercice 7
Écrire l'algorithme.

Exercice 8
Quelle est la complexité moyenne d'une recherche ?

Solutions

Solution de l'exercice 1 - Formule de récurrence
Si n est pair `separation` donne deux parties de même taille égale à $\frac{n}{2} = n \div 2 = n - n \div 2$.
Si $n = 2p + 1$ est impair `separation` donne deux parties de taille $p = n \div 2$ et $p + 1 = n - n \div 2$.
La fusion demande au plus $n - 1$ comparaisons car on peut parvenir à deux liste avec un élément avec $n - 2$ comparaisons et il reste une comparaison pour ces deux termes.
$C(1) = 0$ correspond au cas terminal d'une liste à un élément.

Solution de l'exercice 2 - Simplification
Pour $n = 2p$, $C(n) = 2C(p) + n - 1$ et $C(n + 1) = C(p) + C(p + 1) - n$
donc $D(n) = C(p + 1) - C(p) + 1 = D(p) + 1$
Pour $n = 2p + 1$, $C(n) = C(p) + C(p + 1) + n - 1$ et $C(n + 1) = 2C(p + 1) - n$
donc $D(n) = C(p + 1) - C(p) + 1 = D(p) + 1$.
Dans les deux on a $p = n \div 2$ d'où la formule de récurrence.
On a $C(2) = 2C(1) + 1 = 1$ donc $D(1) = C(2) - C(1) = 1$.

Solution de l'exercice 3 - Calcul des incréments
On a $D(2) = D(1)+1 = 2$, $D(3) = D(1)+1 = 2$. On calcule aussi $D(4) = D(5) = D(6) = D(7) = 3$.
Ce sont ces calculs qui permettent de conjecturer le résultat.
Le résultat est donc vrai pour $p = 0$, $p = 1$ et $p = 2$.
S'il est vrai pour p, on considère n tel que $2^{p+1} \leqslant n < 2^{p+2}$.
On a $D(n) = D(n \div 2) + 1$ et $2^p \leqslant n \div 2 < 2^{p+1}$ donc, d'après l'hypothèse de récurrence,
$D(n \div 2) = p + 1$ d'où $D(n) = p + 2$. La propriété est vraie pour tout p.

Solution de l'exercice 4 - Téléscopage
On a $C(n) = C(n) - C(1) = \sum_{k=1}^{n-1} C(k + 1) - C(k) = \sum_{k=1}^{n-1} D(k)$.
On pose q tel que$2^q \leqslant n < 2^{q+1} - 1$. On somme par paquets en séparant aux 2^p,
$C(n) = \sum_{p=0}^{q-1} \sum_{k=2^p}^{2^{p+1}-1} D(k) + \sum_{k=2^q}^{n-1} D(k) = \sum_{p=0}^{q-1} \sum_{k=2^p}^{2^{p+1}-1} (p + 1) + \sum_{k=2^q}^{n-1} (q + 1)$.
En comptant le nombre d'éléments égaux on obtient $C(n) = \sum_{p=0}^{q-1} 2^p \cdot (p + 1) + (n - 2^q) \cdot (q + 1)$
En écrivant $2^p = 2^{p+1} - 2^p$ on peut simplifier

$$\sum_{p=0}^{q-1} 2^p \cdot (p + 1) = \sum_{p=0}^{q-1} 2^{p+1} \cdot (p + 1) - \sum_{p=0}^{q-1} 2^p \cdot (p + 1) = \sum_{p=1}^{q} 2^p \cdot p - \sum_{p=0}^{q-1} 2^p \cdot (p + 1)$$
$$= q \cdot 2^q + \sum_{p=1}^{q-1} 2^p \cdot (p - p - 1) - 1 = q \cdot 2^q - \sum_{p=1}^{q-1} 2^p - 1 = q \cdot 2^q - (2^q - 2) - 1$$

d'où $C(n) = q \cdot 2^q - (2^q - 2) - 1 + n \cdot q + n - q \cdot 2^q - 2^q = n \cdot q + n - 2 \cdot 2^q + 1$.
C'est le résultat demandé car $q = \lfloor \log_2(n) \rfloor$.

Solution de l'exercice 5 - Regroupements

Chaque élément de $S_n(A)$ est défini de manière unique par sa restriction à $I_- = A$ (à valeurs dans $\{1, 2, \dots, k-1\}$) et à $I_+(\sigma) = \widehat{A} = \{2, 3, \dots, n\} \setminus A$ (à valeurs dans $\{k+1, \dots, n\}$).
Ces restrictions sont des bijections de A vers $\{1, 2, \dots, k-1\}$ et de $\widehat{A}$ vers $\{k+1, \dots, n\}$.
Inversement tout couple de bijection permet de définir une bijection σ de $A \cup \widehat{A}$ vers $E_{1,n}$ puis, en prolongeant par $\sigma(k) = 1$ en une permutation de $\{1, 2, \dots, n\}$.
On peut donc assimiler $\mathfrak{S}_n(k, A)$ au produit des ensembles de bijections $\sigma \leftrightarrow (f_1, f_2)$
$S_n(A)$ est donc de cardinal $(k-1)!(n-k)!$, produit des cardinaux des ensembles de bijections.

L'indication permet de remarquer que $\ell_{\sigma,g}$ ne dépend que de f_1. Ainsi comme le nombre de bijections f_2 est $(n-k)!$, $\displaystyle\sum_{\sigma\in\mathfrak{S}_n(k,A)} C(\ell_{\sigma,g}) = \sum_{f_1}\sum_{f_1} C(\ell_{\sigma,g}) = (n-k)! \cdot \sum_{f_1} C(\ell_{\sigma,g})$.
De plus chaque bijection f_1 va donner une permutation différente de $[a1; a_2; \dots; a_{k-1}]$ car la séparation produit une bijection constante de I_- vers $\{1, 2, \dots, i-1\}$ d'où $\displaystyle\sum_{f_1} C(\ell_{\sigma,g}) = (k-1)! \cdot \overline{C}(k-1)$.

Ainsi $\displaystyle\sum_{\sigma\in\mathfrak{S}_n(k,A)} C(\ell_{\sigma,g}) = (n-k)! \cdot (k-1)! \cdot \overline{C}(k-1)$. De même pour l'autre formule.

Solution de l'exercice 6 - Récurrence

On a $\displaystyle\sum_{\sigma\in\mathfrak{S}_n} (n-1) = |\mathfrak{S}_n| \cot(n-1) = n! \cdot (n-1)$.

$$\sum_{\sigma\in\mathfrak{S}_n} (n-1)C(\ell_g) = \sum_{k=1}^{n} \sum_{|A|=k-1} \sum_{\sigma\in\mathfrak{S}_n(k,A)} C(\ell_{\sigma,g}) = \sum_{k=1}^{n} \sum_{|A|=k-1} (n-k)! \cdot (k-1)! \cdot \overline{C}(k-1).$$

Il y a $\binom{n-1}{k-1}$ sous ensembles de cardinal k de $\{2, 3, \dots, n\}$ donc

$$\sum_{\sigma\in\mathfrak{S}_n} (n-1)C(\ell_g) = \sum_{k=1}^{n} \binom{n-1}{k-1} \cdot (n-k)! \cdot (k-1)! \cdot \overline{C}(k-1) = (n-1)! \cdot \sum_{k=1}^{n} \overline{C}(k-1).$$

De même $\displaystyle\sum_{\sigma\in\mathfrak{S}_n} (n-1)C(\ell_d) = (n-1)! \cdot \sum_{k=1}^{n} \overline{C}(n-k)$.

On a alors, en divisant par $n!$ $\overline{C}(n) = n - 1 + \dfrac{1}{n} \cdot \displaystyle\sum_{k=1}^{n} \overline{C}(k-1) + \dfrac{1}{n} \cdot \sum_{k=1}^{n} \overline{C}(n-k)$.

Des changements d'indices donnent $\overline{C}(n) = n - 1 + \dfrac{1}{n} \cdot \displaystyle\sum_{i=0}^{n-1} \overline{C}(i) + \dfrac{1}{n} \cdot \sum_{i=0}^{n-1} \overline{C}(i)$

On trouve alors

$$\begin{aligned} n \cdot \overline{C}(n) - (n-1) \cdot \overline{C}(n-1) &= n \cdot (n-1) - (n-1) \cdot (n-2) + 2 \cdot \sum_{i=0}^{n-1} \overline{C}(i) - 2 \cdot \sum_{i=0}^{n-2} \overline{C}(i) \\ &= 2.(n+1) + 2 \cdot \overline{C}(n-1) \end{aligned}$$

donc $n \cdot \overline{C}(n) = (n+1) \cdot \overline{C}(n-1) + 2 \cdot (n-1)$.
On divise par $n \cdot (n+1)$: $\dfrac{\overline{C}(n)}{n+1} = \dfrac{\overline{C}(n-1)}{n} + \dfrac{2 \cdot (n-1)}{n \cdot (n+1)} = \dfrac{\overline{C}(n-1)}{n} + \dfrac{4}{n+1} - \dfrac{2}{n}$.
Par télescopage on a alors

$$\begin{aligned} \frac{\overline{C}(n)}{n+1} = \frac{\overline{C}(n)}{n+1} - \frac{\overline{C}(0)}{0+1} &= \sum_{k=1}^{n} \frac{\overline{C}(k)}{k+1} - = \frac{\overline{C}(k-1)}{k} = \sum_{k=1}^{n} \frac{4}{k+1} - \frac{2}{k} \\ &= 4 \cdot \sum_{k=1}^{n} \frac{1}{k+1} - 2 \cdot \sum_{k=1}^{n} \frac{1}{k} = 4 \cdot \sum_{k=2}^{n+1} \frac{1}{k} - 2 \cdot \sum_{k=1}^{n} \frac{1}{k} = \frac{4}{n+1} + 2 \cdot \sum_{k=1}^{n} \frac{1}{k} - 4 = 2 \cdot \sum_{k=1}^{n} \frac{1}{k} - \frac{4 \cdot n}{n+1} \end{aligned}$$

On en déduit le résultat demandé et la complexité du théorème car $\displaystyle\sum_{k=1}^{n} \frac{1}{k} \sim \ln(n)$.

Solution de l'exercice 7
On commence par le découpage, on ajoute le décompte du premier sous-ensemble.

```
let rec filtrerC liste pivot =
  match liste with
  |[] -> [], [], 0
  |t::q -> let l1, l2, k = filtrerC q pivot in
           if t < pivot
             then t::l1, l2, k+1
             else l1, t::l2, k;;
```

On traduit ensuite l'algorithme décrit.

```
let rec selection liste p =
  match liste with
  |[] -> failwith "il n'y a pas assez d'éléments dans la liste"
  |t::q -> let l1, l2, k = filtrerC q t in
           if k = p -1 then t
                       else (if k < p-1
                               then selection l2 (p-1-k)
                               else selection l1 p);;
```

Solution de l'exercice 8
On va reproduire l'étude du tri pivot.
On note $C_p(n)$ la complexité moyenne de recherche du p-ième élément dans une liste de taille n et $C(n) = \frac{1}{n}\sum_{p=1}^{n} C_p(n)$ la complexité moyenne de recherche d'un élément dans une liste de taille n.

$C_p(n) = n - 1 + \frac{1}{n}\left(\sum_{k=0}^{p-1} C_{p-1-k}(n-k-1) + 0 + \sum_{k=p}^{n-1} C_p(k)\right)$ d'où

$nC(n) = \sum_{p=1}^{n}(n-1) + \frac{1}{n}\sum_{p=1}^{n}\sum_{k=0}^{p-2} C_{p-1-k}(n-k-1) + \frac{1}{n}\sum_{p=1}^{n}\sum_{k=p}^{n-1} C_p(k).$

On a $\sum_{p=1}^{n}(n-1) = n(n-1)$, $\sum_{p=1}^{n}\sum_{k=p}^{n-1} C_p(k) = \sum_{k=1}^{n-1}\sum_{p=1}^{k} C_p(k) = \sum_{k=1}^{n-1} kC(k)$

et, en posant $q = p-1-k$ puis $l = n-1-k$,

$\sum_{p=1}^{n}\sum_{k=0}^{p-2} C_{p-1-k}(n-k-1) = \sum_{k=0}^{n-2}\sum_{p=k+2}^{n} C_{p-1-k}(n-k-1) = \sum_{k=0}^{n-2}\sum_{q=1}^{n-1-k} C_q(n-k-1)$

$\sum_{p=1}^{n}\sum_{k=0}^{p-2} C_{p-1-k}(n-k-1) = \sum_{k=0}^{n-2}(n-1-k)C(n-k-1) = \sum_{l=1}^{n-1} lC(l)$

Ainsi $nC(n) = n(n-1) + \frac{2}{n}\sum_{k=1}^{n-1} kC(k)$ donc $n^2C(n) = n^2(n-1) + 2\sum_{k=1}^{n-1} kC(k)$.
En calculant la différence de termes consécutifs

$(n+1)^2C(n+1) - n^2C(n) = (n+1)^2 n + 2\sum_{k=1}^{n} kC(k) - n^2(n-1) - 2\sum_{k=1}^{n-1} kC(k)$

on obtient $(n+1)^2C(n+1) - n^2C(n) = n(3n-2) + 2nC(n)$
d'où $(n+1)^2C(n+1) = n(3n-2) + n(n+2)C(n)$ avec $C(1) = 0$.
On a $n^2 + 2n \leqslant n^2 + 2n + 1 = (n+1)^2$ et $n(3n-2) \leqslant 3n^2 + 6n + 3 = 3(n+1)^2$ donc
$C(n+1) \leqslant 3 + C(n)$ puis $C(n) \leqslant 3(n-1)$.
On obtient un complexité linéaire, légèrement meilleure que celle d'un tri.

On pouvait faire plus précis en posant $u_n = \frac{n}{n+1}C(n)$. On a alors, en divisant par $(n+1)(n+2)$, $u_{n+1} = u_n + 3 + \dfrac{5}{n+1} - \dfrac{16}{n+2}$ et $u_1 = 0$.

On a donc $u_n = 3(n-1) + 5\displaystyle\sum_{k=2}^{n} \frac{1}{k} - 16\sum_{k=3}^{n+1} \frac{1}{k} = 3n - 9H_n + 4 - \frac{16}{n+1}$