Une étude de complexité : la sélection

Philippe Duchon

DIU EIL - Université de Bordeaux

27 juin 2019

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \ldots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)
- $m_1 = 0$: rien à faire ; $m_2 = 1$: une seule paire à comparer ; $m_3 = 3$: avec deux comparaisons on n'est pas sûr de terminer.

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)
- $m_1 = 0$: rien à faire ; $m_2 = 1$: une seule paire à comparer ; $m_3 = 3$: avec deux comparaisons on n'est pas sûr de terminer.
- $m_4 = 5$: pas très difficile ($2^5 = 32$, $4! = 24$).

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)
- $m_1 = 0$: rien à faire ; $m_2 = 1$: une seule paire à comparer ; $m_3 = 3$: avec deux comparaisons on n'est pas sûr de terminer.
- $m_4 = 5$: pas très difficile ($2^5 = 32$, $4! = 24$).
- $m_5 = 7$: pas si facile ($2^7 = 128$, $5! = 120$).

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)
- $m_1 = 0$: rien à faire ; $m_2 = 1$: une seule paire à comparer ; $m_3 = 3$: avec deux comparaisons on n'est pas sûr de terminer.
- $m_4 = 5$: pas très difficile ($2^5 = 32$, $4! = 24$).
- $m_5 = 7$: pas si facile ($2^7 = 128$, $5! = 120$).
- Plus facile : $m_6 \leq m_5 + 3$; $m_7 \leq m_6 + 3$, ensemble montrent que $m_6 = 10$ et $m_7 = 13$ ($2^{10} = 1024$, $6! = 720$; $2^{13} = 8192$, $7! = 5040$).

Préambule : trier optimalement des *petits* ensembles

- **Problème :** on a k valeurs, $a_1 \dots a_k$; on veut les trier.
- **Contrainte :** on ne fait que des comparaisons de valeurs 2 à 2
- **Question :** combien, en fonction de k, faut-il **au minimum** de comparaisons pour trier (dans tous les cas) k valeurs ? On note m_k ce nombre.
- **Classique :** on a forcément $2^{m_k} \geq k!$ (pourquoi ?)
- $m_1 = 0$: rien à faire ; $m_2 = 1$: une seule paire à comparer ; $m_3 = 3$: avec deux comparaisons on n'est pas sûr de terminer.
- $m_4 = 5$: pas très difficile ($2^5 = 32$, $4! = 24$).
- $m_5 = 7$: pas si facile ($2^7 = 128$, $5! = 120$).
- Plus facile : $m_6 \leq m_5 + 3$; $m_7 \leq m_6 + 3$, ensemble montrent que $m_6 = 10$ et $m_7 = 13$ ($2^{10} = 1024$, $6! = 720$; $2^{13} = 8192$, $7! = 5040$).
- m_{16} n'est pas connu (45 ou 46) ; m_{12} est le premier pour lequel on a $m_k > \lceil \log_2(k!) \rceil \dots$

Le problème : sélection

- **En entrée :** un tableau de n valeurs $x_1, \ldots, x_n$, dans un univers totalement ordonné ; un entier $1 \leq k \leq n$
- **En sortie :** le k-ème plus petit des x_i ($k = 1$: le minimum ; $k = n$: le maximum ; $k = \lfloor n/2 \rfloor$: une médiane
- **Hypothèse :** on procède par comparaisons de deux valeurs (et donc, on ne dépend pas de la nature de ce qu'on compare)
- **Complexité :** on mesure la complexité en temps par le nombre de comparaisons ; tous les algorithmes qu'on va étudier peuvent s'écrire avec une complexité en espace $O(n)$ – pas forcément en place.
- **Notation :** x_k désigne le k-ème élément du tableau de départ ; $x^{(k)}$, le k-ème du tableau si on le trie par ordre croissant ; on cherche donc à trouver $x^{(k)}$.

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$
- En toute généralité, on a une solution qui consiste à trier, puis à prendre le k-ème : complexité $\Theta(n \log(n))$ comparaisons avec un algorithme optimal de tri.

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$
- En toute généralité, on a une solution qui consiste à trier, puis à prendre le k-ème : complexité $\Theta(n \log(n))$ comparaisons avec un algorithme optimal de tri.
- Si $k = 1$: on cherche le minimum ; on peut le faire en parcourant le tableau non trié, en $n - 1$ comparaisons.

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$
- En toute généralité, on a une solution qui consiste à trier, puis à prendre le k-ème : complexité $\Theta(n \log(n))$ comparaisons avec un algorithme optimal de tri.
- Si $k = 1$: on cherche le minimum ; on peut le faire en parcourant le tableau non trié, en $n - 1$ comparaisons.
- Pareil si $k = n$, on cherche le maximum.

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$
- En toute généralité, on a une solution qui consiste à trier, puis à prendre le k-ème : complexité $\Theta(n \log(n))$ comparaisons avec un algorithme optimal de tri.
- Si $k = 1$: on cherche le minimum ; on peut le faire en parcourant le tableau non trié, en $n - 1$ comparaisons.
- Pareil si $k = n$, on cherche le maximum.
- Pour $k = 2, 3, 4 \ldots$ ou $k = n - 1, n - 2, \ldots$ (k proche de 1, ou proche de n), on peut procéder itérativement, en calculant k fois le minimum (maximum) de ceux qui restent ; complexité $\Theta(k.n)$

Cas particuliers, étude préliminaire

- Si le tableau est déjà trié : il suffit de prendre x_k, donc complexité $O(1)$
- En toute généralité, on a une solution qui consiste à trier, puis à prendre le k-ème : complexité $\Theta(n \log(n))$ comparaisons avec un algorithme optimal de tri.
- Si $k = 1$: on cherche le minimum ; on peut le faire en parcourant le tableau non trié, en $n - 1$ comparaisons.
- Pareil si $k = n$, on cherche le maximum.
- Pour $k = 2, 3, 4 \ldots$ ou $k = n - 1, n - 2, \ldots$ (k proche de 1, ou proche de n), on peut procéder itérativement, en calculant k fois le minimum (maximum) de ceux qui restent ; complexité $\Theta(k.n)$
- **Est-ce que c'est toujours une bonne idée ?** Pour $k = \lfloor n/2 \rfloor$ (médiane), $k = \lfloor n/4 \rfloor$ (quartile), c'est $\Theta(n^2)$, donc **plus coûteux** que le tri.

Une remarque préliminaire

- N'importe quelle méthode qui est sûre de trouver le bon élément $x^{(k)}$, va aussi, sans comparaison supplémentaire, identifier $k-1$ éléments inférieurs ou égaux à $x^{(k)}$, les autres étant supérieurs ou égaux à $x^{(k)}$.

Une remarque préliminaire

- N'importe quelle méthode qui est sûre de trouver le bon élément $x^{(k)}$, va aussi, sans comparaison supplémentaire, identifier $k-1$ éléments inférieurs ou égaux à $x^{(k)}$, les autres étant supérieurs ou égaux à $x^{(k)}$.
- **Donc,** pour tout k, on a une forcément besoin d'au moins $n-1$ comparaisons, y compris pour le **meilleur** des cas.

Une remarque préliminaire

- N'importe quelle méthode qui est sûre de trouver le bon élément $x^{(k)}$, va aussi, sans comparaison supplémentaire, identifier $k-1$ éléments inférieurs ou égaux à $x^{(k)}$, les autres étant supérieurs ou égaux à $x^{(k)}$.
- **Donc,** pour tout k, on a une forcément besoin d'au moins $n-1$ comparaisons, y compris pour le **meilleur** des cas.
- (Exercice pour plus tard : pour chaque idée d'algorithme, trouver comment l'adapter pour qu'il fournisse la liste, pas forcément triée, des k plus petits au lieu de juste $x^{(k)}$; et ce, sans comparaisons supplémentaires)

Deux pistes. . .

- “On adapte le tri rapide” : on prend un pivot, et on partitionne comme pour le tri rapide ; puis on applique récursivement l’algorithme sur l’un des sous-tableaux.

Deux pistes. . .

- “On adapte le tri rapide” : on prend un pivot, et on partitionne comme pour le tri rapide ; puis on applique récursivement l’algorithme sur l’un des sous-tableaux.
- **Diviser pour régner :** on trouve un moyen de se ramener à un ou plusieurs problèmes de même nature, mais significativement plus petits, avec un coût supplémentaire pas trop élevé.

Deux pistes...

- "On adapte le tri rapide" : on prend un pivot, et on partitionne comme pour le tri rapide ; puis on applique récursivement l'algorithme sur l'un des sous-tableaux.
- **Diviser pour régner :** on trouve un moyen de se ramener à un ou plusieurs problèmes de même nature, mais significativement plus petits, avec un coût supplémentaire pas trop élevé.
- On sait déjà que n'importe quel algorithme nécessite $\Omega(n)$ comparaisons ; ce qu'on aimerait savoir, c'est s'il est possible d'avoir un algorithme qui, pour tout k, se contente de $O(n)$ comparaisons.

“Sélection rapide” (d’après le tri rapide)

- On a un tableau T ; on s’intéresse aux cases entre d et f ; et un entier k, $d \leq k \leq f$.
- On veut identifier l’élément qui, si on triait le tableau, se retrouverait en position k.
- On choisit un élément (par exemple $T[d]$), et on l’utilise comme pivot pour partitionner ; en sortie,
  - le pivot se trouve dans une case ℓ ;
  - de la case d à la case $\ell - 1$, on a des valeurs inférieures au pivot ;
  - de la case $\ell + 1$ à la case f, on a des valeurs supérieures au pivot.

“Sélection rapide” (d’après le tri rapide)

- On a un tableau T ; on s’intéresse aux cases entre d et f ; et un entier k, $d \leq k \leq f$.
- On veut identifier l’élément qui, si on triait le tableau, se retrouverait en position k.
- On choisit un élément (par exemple $T[d]$), et on l’utilise comme pivot pour partitionner ; en sortie,
  - le pivot se trouve dans une case ℓ ;
  - de la case d à la case $\ell - 1$, on a des valeurs inférieures au pivot ;
  - de la case $\ell + 1$ à la case f, on a des valeurs supérieures au pivot.
- Trois cas sont possibles :
  - $\ell = k$: coup de bol ; c’est fini, la valeur cherchée est $T[k]$.
  - $\ell < k$: l’élément cherché est entre la case d et la case $\ell - 1$; on relance l’algorithme entre d et $\ell - 1$, et on retournera le résultat.
  - $\ell > k$: l’élément cherché est entre la case $d + 1$ et la case f ; on relance l’algorithme entre $\ell + 1$ et f, et on retournera le résultat.

Quelques mots sur l'analyse de cet algorithme

- La complexité du pire cas n'est pas bonne : comme pour le tri rapide, pour tout n et tout k, il existe des tableaux de taille n pour lesquels cet algorithme compare **toutes** les paires de clés, donc $n(n-1)/2$ comparaisons.

Quelques mots sur l'analyse de cet algorithme

- La complexité du pire cas n'est pas bonne : comme pour le tri rapide, pour tout n et tout k, il existe des tableaux de taille n pour lesquels cet algorithme compare **toutes** les paires de clés, donc $n(n-1)/2$ comparaisons.
- On peut faire une analyse en moyenne, en supposant que le tableau est mélangé uniformément avant le début.

Quelques mots sur l'analyse de cet algorithme

- La complexité du pire cas n'est pas bonne : comme pour le tri rapide, pour tout n et tout k, il existe des tableaux de taille n pour lesquels cet algorithme compare **toutes** les paires de clés, donc $n(n-1)/2$ comparaisons.
- On peut faire une analyse en moyenne, en supposant que le tableau est mélangé uniformément avant le début.
- On peut aussi "randomiser" l'algorithme : à chaque étape de choix du pivot, on choisit son indice aléatoirement ; pour ne pas compliquer l'algorithme, on échange le pivot avec la position de début (d) et on applique ensuite le partitionnement classique.

Quelques mots sur l'analyse de cet algorithme

- La complexité du pire cas n'est pas bonne : comme pour le tri rapide, pour tout n et tout k, il existe des tableaux de taille n pour lesquels cet algorithme compare **toutes** les paires de clés, donc $n(n-1)/2$ comparaisons.
- On peut faire une analyse en moyenne, en supposant que le tableau est mélangé uniformément avant le début.
- On peut aussi "randomiser" l'algorithme : à chaque étape de choix du pivot, on choisit son indice aléatoirement ; pour ne pas compliquer l'algorithme, on échange le pivot avec la position de début (d) et on applique ensuite le partitionnement classique.
- L'analyse en moyenne est alors la même, et est un peu technique mathématiquement (manipulation de sommes)

Quelques mots sur l'analyse de cet algorithme

- La complexité du pire cas n'est pas bonne : comme pour le tri rapide, pour tout n et tout k, il existe des tableaux de taille n pour lesquels cet algorithme compare **toutes** les paires de clés, donc $n(n-1)/2$ comparaisons.
- On peut faire une analyse en moyenne, en supposant que le tableau est mélangé uniformément avant le début.
- On peut aussi "randomiser" l'algorithme : à chaque étape de choix du pivot, on choisit son indice aléatoirement ; pour ne pas compliquer l'algorithme, on échange le pivot avec la position de début (d) et on applique ensuite le partitionnement classique.
- L'analyse en moyenne est alors la même, et est un peu technique mathématiquement (manipulation de sommes)
- **Résultat :** *en moyenne* (en espérance), le nombre de comparaisons est linéaire : le maximum est atteint pour $k = \lfloor n/2 \rfloor$, et est de

$$2n(1 + \ln(2)) + o(n)$$

Un algorithme de type “diviser pour régner”

Idée générale : (choisir un entier ℓ, qui sera un paramètre)

- diviser le tableau T de n valeurs en $n/(2\ell+1)$ “petits” tableaux de taille $2\ell+1$;
- trouver la médiane de chacun des $n/(2\ell+1)$ petits tableaux ; chaque médiane peut se calculer en $m_{2\ell+1}$ comparaisons ;
- trouver la médiane de ces médianes
- cette médiane-des-médianes aura dans le tableau d’origine T, un rang loin à la fois de 1 et de n ; on va l’utiliser comme pivot pour le partitionnement, et continuer récursivement.

Un algorithme de type “diviser pour régner”

Idée générale : (choisir un entier ℓ, qui sera un paramètre)

- diviser le tableau T de n valeurs en $n/(2\ell+1)$ “petits” tableaux de taille $2\ell+1$;
- trouver la médiane de chacun des $n/(2\ell+1)$ petits tableaux ; chaque médiane peut se calculer en $m_{2\ell+1}$ comparaisons ;
- trouver la médiane de ces médianes (appel récursif)
- cette médiane-des-médianes aura dans le tableau d’origine T, un rang loin à la fois de 1 et de n ; on va l’utiliser comme pivot pour le partitionnement, et continuer récursivement. (appel récursif)

Rang de la médiane-des-médianes

(On fait comme si n était un multiple de $2\ell + 1$; et même, $n/(2\ell + 1)$ impair ; et on suppose que toutes les valeurs dans T sont distinctes)

- Soit $M = (y^{(1)}, \ldots, y^{(2m+1)})$ l'ensemble des médianes : la médiane-des-médianes est $y^{(m+1)}$.

Rang de la médiane-des-médianes

(On fait comme si n était un multiple de $2\ell + 1$; et même, $n/(2\ell + 1)$ impair ; et on suppose que toutes les valeurs dans T sont distinctes)

- Soit $M = (y^{(1)}, \ldots, y^{(2m+1)})$ l'ensemble des médianes : la médiane-des-médianes est $y^{(m+1)}$.
- On a $m + 1$ médianes $\leq y^{(m+1)}$, chacune ayant $k + 1$ éléments inférieurs ou égaux dans son "petit tableau".

Rang de la médiane-des-médianes

(On fait comme si n était un multiple de $2\ell+1$; et même, $n/(2\ell+1)$ impair ; et on suppose que toutes les valeurs dans T sont distinctes)

- Soit $M = (y^{(1)}, \ldots, y^{(2m+1)})$ l'ensemble des médianes : la médiane-des-médianes est $y^{(m+1)}$.
- On a $m+1$ médianes $\leq y^{(m+1)}$, chacune ayant $k+1$ éléments inférieurs ou égaux dans son "petit tableau".
- Donc on a au moins $(\ell+1)(m+1)$ éléments inférieurs ou égaux à $y^{(m+1)}$ dans le tableau T : le rang de $y^{(m+1)}$ dans T est au moins $(\ell+1)(m+1) \simeq \frac{\ell+1}{2(2\ell+1)} n$

Rang de la médiane-des-médianes

(On fait comme si n était un multiple de $2\ell+1$; et même, $n/(2\ell+1)$ impair ; et on suppose que toutes les valeurs dans T sont distinctes)

- Soit $M = (y^{(1)}, \ldots, y^{(2m+1)})$ l'ensemble des médianes : la médiane-des-médianes est $y^{(m+1)}$.
- On a $m+1$ médianes $\leq y^{(m+1)}$, chacune ayant $k+1$ éléments inférieurs ou égaux dans son "petit tableau".
- Donc on a au moins $(\ell+1)(m+1)$ éléments inférieurs ou égaux à $y^{(m+1)}$ dans le tableau T : le rang de $y^{(m+1)}$ dans T est au moins $(\ell+1)(m+1) \simeq \frac{\ell+1}{2(2\ell+1)} n$
- Même raisonnement pour les "supérieurs ou égaux" : le rang de $y^{(m+1)}$ dans T, est au plus $n - \frac{\ell+1}{2(2\ell+1)} n$.

Rang de la médiane-des-médianes

(On fait comme si n était un multiple de $2\ell+1$; et même, $n/(2\ell+1)$ impair ; et on suppose que toutes les valeurs dans T sont distinctes)

- Soit $M = (y^{(1)}, \ldots, y^{(2m+1)})$ l'ensemble des médianes : la médiane-des-médianes est $y^{(m+1)}$.
- On a $m+1$ médianes $\leq y^{(m+1)}$, chacune ayant $k+1$ éléments inférieurs ou égaux dans son "petit tableau".
- Donc on a au moins $(\ell+1)(m+1)$ éléments inférieurs ou égaux à $y^{(m+1)}$ dans le tableau T : le rang de $y^{(m+1)}$ dans T est au moins $(\ell+1)(m+1) \simeq \frac{\ell+1}{2(2\ell+1)} n$
- Même raisonnement pour les "supérieurs ou égaux" : le rang de $y^{(m+1)}$ dans T, est au plus $n - \frac{\ell+1}{2(2\ell+1)} n$.
- **Donc**, lors du deuxième appel récursif après partitionnement, le tableau sur lequel on travaille est de taille au plus $\simeq n(1 - \frac{\ell+1}{4\ell+2}) = n.\frac{3\ell+1}{4\ell+2}$.

Trouver la récurrence

On compte les comparaisons, en arrondissant comme si les fractions étaient entières :

- Trouver les $n/(2\ell+1)$ médianes : $m_{2\ell+1}n/(2\ell+1)$ (les petits tableaux sont de taille $2\ell+1$: on peut trier chaque petit tableau pour trouver sa moyenne.
- Trouver la médiane-des-médianes : $C_{n/(2\ell+1)}$.
- Partitionner une fois la médiane-des-médianes trouvée : $n\frac{\ell}{4\ell+2}$ si on s'arrange pour ne pas faire les comparaisons dont on connaît déjà le résultat après la détermination de la médiane-des-médianes.
- Appel récursif après le partitionnement : C_x, avec x non déterminé, mais on sait d'après le calcul précédent que $x \leq \frac{3\ell+1}{4\ell+2}n$; on peut majorer ce coût par $C_{\frac{3\ell+1}{4\ell+2}n}$, car la complexité dans le pire cas est **croissante**.
- On a donc une récurrence (pour une majoration de la complexité) :

$$C_n = C_{n/(2\ell+1)} + C_{\frac{3\ell+1}{4\ell+2}n} + a.n,$$

Utilisation du théorème maître (version S. Roura)

- On a une récurrence, pour la *majoration* du coût (car on majore la taille du dernier appel récursif) :

$$C_n = \alpha.n + C_{a.n} + C_{b.n}$$

Utilisation du théorème maître (version S. Roura)

- On a une récurrence, pour la *majoration* du coût (car on majore la taille du dernier appel récursif) :

$$C_n = \alpha.n + C_{a.n} + C_{b.n}$$

- (Dans cette récurrence, on néglige des termes additifs "petits" : $a.n$ c'est en fait $a.n \pm f(n)$ avec f bornée – ne serait-ce que pour avoir des indices entiers ; le théorème de Roura est robuste, il "marche" aussi avec ces variations)

Utilisation du théorème maître (version S. Roura)

- On a une récurrence, pour la *majoration* du coût (car on majore la taille du dernier appel récursif) :

$$C_n = \alpha.n + C_{a.n} + C_{b.n}$$

- (Dans cette récurrence, on néglige des termes additifs "petits" : $a.n$ c'est en fait $a.n \pm f(n)$ avec f bornée – ne serait-ce que pour avoir des indices entiers ; le théorème de Roura est robuste, il "marche" aussi avec ces variations)
- Dans le cas où $a + b < 1$ (c'est le cas pour notre algorithme avec $\ell \geq 2$, pas pour $\ell = 1$), il prévoit un comportement asymptotique précis :

$$C_n = \frac{\alpha}{1 - a - b} n + o(n)$$

($o(n)$ désigne un terme d'erreur qui est asymptotiquement négligeable devant n)

Utilisation du théorème maître (version S. Roura)

- On a une récurrence, pour la *majoration* du coût (car on majore la taille du dernier appel récursif) :

$$C_n = \alpha.n + C_{a.n} + C_{b.n}$$

- (Dans cette récurrence, on néglige des termes additifs "petits" : $a.n$ c'est en fait $a.n \pm f(n)$ avec f bornée – ne serait-ce que pour avoir des indices entiers ; le théorème de Roura est robuste, il "marche" aussi avec ces variations)
- Dans le cas où $a + b < 1$ (c'est le cas pour notre algorithme avec $\ell \geq 2$, pas pour $\ell = 1$), il prévoit un comportement asymptotique précis :

$$C_n = \frac{\alpha}{1 - a - b} n + o(n)$$

($o(n)$ désigne un terme d'erreur qui est asymptotiquement négligeable devant n)

- Reste à évaluer α...

Médiane des petits tableaux

- On peut toujours trouver la médiane de $2\ell + 1$ valeurs en les triant : $m_{2\ell+1}$, au moins si ℓ est petit (on peut prendre $\ell = 2$ ou $\ell = 3$).

Médiane des petits tableaux

- On peut toujours trouver la médiane de $2\ell + 1$ valeurs en les triant : $m_{2\ell+1}$, au moins si ℓ est petit (on peut prendre $\ell = 2$ ou $\ell = 3$).
- Il faut aussi compter les comparaisons du partitionnement : si on évite de comparer la médiane-des-médianes aux éléments déjà connus pour être plus petits ou plus grands, il reste environ $\frac{\ell}{4\ell+2} n$ comparaisons.

Médiane des petits tableaux

- On peut toujours trouver la médiane de $2\ell+1$ valeurs en les triant : $m_{2\ell+1}$, au moins si ℓ est petit (on peut prendre $\ell=2$ ou $\ell=3$).
- Il faut aussi compter les comparaisons du partitionnement : si on évite de comparer la médiane-des-médianes aux éléments déjà connus pour être plus petits ou plus grands, il reste environ $\frac{\ell}{4\ell+2}n$ comparaisons.
- Pour $\ell=2$, la constante α vaut $7/5+2/10=8/5$; pour $\ell=3$, α vaut $13/7+3/14=29/14$. On divise par $1-a-b=(\ell-1)/(4\ell+2)$; au final la constante dans C_n pour $\ell=2$ vaut 16, celle pour $\ell=3$ vaut 14.5 (mais il faut programmer un tri optimal pour 7 au lieu de 5).

Preuve “à la main”

- La récurrence exacte pour notre algorithme dépend de certains détails : comment on traite les “petits cas”, notamment.

Preuve “à la main”

- La récurrence exacte pour notre algorithme dépend de certains détails : comment on traite les “petits cas”, notamment.
- Par exemple, on peut convenir de ne faire d'appel récursif qu'à partir de la taille 21, et trier complètement pour les tailles jusqu'à 20. On peut utiliser un tri peu efficace comme le tri à bulles pour les petits tableaux : ça ne change pas la complexité asymptotique.

Preuve “à la main”

- La récurrence exacte pour notre algorithme dépend de certains détails : comment on traite les “petits cas”, notamment.
- Par exemple, on peut convenir de ne faire d'appel récursif qu'à partir de la taille 21, et trier complètement pour les tailles jusqu'à 20. On peut utiliser un tri peu efficace comme le tri à bulles pour les petits tableaux : ça ne change pas la complexité asymptotique.
- On va donc compter $C_n = n(n-1)/2$ pour $n \leq 20$.

Preuve “à la main”

- La récurrence exacte pour notre algorithme dépend de certains détails : comment on traite les “petits cas”, notamment.
- Par exemple, on peut convenir de ne faire d'appel récursif qu'à partir de la taille 21, et trier complètement pour les tailles jusqu'à 20. On peut utiliser un tri peu efficace comme le tri à bulles pour les petits tableaux : ça ne change pas la complexité asymptotique.
- On va donc compter $C_n = n(n-1)/2$ pour $n \leq 20$.
- Il est fastidieux d'écrire la récurrence exacte (elle est pleine de parties entières), mais on pourrait trouver des constantes A, B, D, E telles qu'on ait, pour tout $n > 20$,

$$C(n) \leq An + B + C(an + D) + C(bn + E)$$

Preuve “à la main”

- La récurrence exacte pour notre algorithme dépend de certains détails : comment on traite les “petits cas”, notamment.
- Par exemple, on peut convenir de ne faire d'appel récursif qu'à partir de la taille 21, et trier complètement pour les tailles jusqu'à 20. On peut utiliser un tri peu efficace comme le tri à bulles pour les petits tableaux : ça ne change pas la complexité asymptotique.
- On va donc compter $C_n = n(n-1)/2$ pour $n \leq 20$.
- Il est fastidieux d'écrire la récurrence exacte (elle est pleine de parties entières), mais on pourrait trouver des constantes A, B, D, E telles qu'on ait, pour tout $n > 20$,

$$C(n) \leq An + B + C(an + D) + C(bn + E)$$

- On va montrer, sans faire référence au théorème maître, qu'on en déduit $C_n = O(n)$ (on n'aura pas forcément la bonne constante...).

Majoration

On a la récurrence : $C_n = n(n-1)/2$ pour $n \leq 20$, et pour $n \geq 21$,

$$C(n) = An + B + C(an + D) + C(bn + E)$$

On procède par **récurrence forte** sur n : l'hypothèse de récurrence, c'est H_n : *pour tout* $m \leq n$, $C(m) \leq Kn + M$ (pour des constantes K et M qu'on explicitera plus tard).

- **Initialisation :** H_{20} est vraie pour peu qu'on prenne $M \geq 0$ et $20K + M \geq 190$. On peut aussi choisir un $n_0 \geq 20$ comme on veut, calculer tous les $C(m)$ jusqu'à n_0 : ça nous donne de nouvelles conditions pour M et K.

Majoration

On a la récurrence : $C_n = n(n-1)/2$ pour $n \leq 20$, et pour $n \geq 21$,

$$C(n) = An + B + C(an + D) + C(bn + E)$$

On procède par **récurrence forte** sur n : l'hypothèse de récurrence, c'est H_n : *pour tout* $m \leq n$, $C(m) \leq Kn + M$ (pour des constantes K et M qu'on explicitera plus tard).

- **Initialisation :** H_{20} est vraie pour peu qu'on prenne $M \geq 0$ et $20K + M \geq 190$. On peut aussi choisir un $n_0 \geq 20$ comme on veut, calculer tous les $C(m)$ jusqu'à n_0 : ça nous donne de nouvelles conditions pour M et K.
- **Hérédité :** soit $n > n_0$ pour lequel H_{n-1} est vraie ; on doit montrer H_n. Il nous faut donc majorer $C(n)$.

Majoration

On a la récurrence : $C_n = n(n-1)/2$ pour $n \leq 20$, et pour $n \geq 21$,

$$C(n) = An + B + C(an + D) + C(bn + E)$$

On procède par **récurrence forte** sur n : l'hypothèse de récurrence, c'est H_n : *pour tout* $m \leq n$, $C(m) \leq Kn + M$ (pour des constantes K et M qu'on explicitera plus tard).

- **Initialisation :** H_{20} est vraie pour peu qu'on prenne $M \geq 0$ et $20K + M \geq 190$. On peut aussi choisir un $n_0 \geq 20$ comme on veut, calculer tous les $C(m)$ jusqu'à n_0 : ça nous donne de nouvelles conditions pour M et K.
- **Hérédité :** soit $n > n_0$ pour lequel H_{n-1} est vraie ; on doit montrer H_n. Il nous faut donc majorer $C(n)$.
- Dès lors que $n \geq n_0 + 1$, $an + D$ et $bn + E$ sont inférieurs à n_0 ($a = 1/5$, $b = 7/10$; éventuellement, en réévaluant n_0 à la hausse, c'est bon) ; on peut donc appliquer H_n à $n_1 = an + D$ et à $n_2 = bn + E$.

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

- Est-ce qu'on peut choisir K et M de telle sorte que cette majoration soit plus petite que $Kn + M$?

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

- Est-ce qu'on peut choisir K et M de telle sorte que cette majoration soit plus petite que $Kn + M$?
- Ça revient à : pour tout $n > n_0$,
$(K(1 - a - b) - A)n \geq M + B + K(D + E)$.

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

- Est-ce qu'on peut choisir K et M de telle sorte que cette majoration soit plus petite que $Kn + M$?
- Ça revient à : pour tout $n > n_0$,
  $(K(1 - a - b) - A)n \geq M + B + K(D + E)$.
- Si on prend $K > A/(1 - a - b)$, le coefficient de n est positif, et il sufit que ce soit vrai pour $n = n_0$, soit :

$$K((1 - a - b)n_0 - (D + E)) \geq B + 2M - An_0.$$

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

- Est-ce qu'on peut choisir K et M de telle sorte que cette majoration soit plus petite que $Kn + M$?
- Ça revient à : pour tout $n > n_0$, $(K(1 - a - b) - A)n \geq M + B + K(D + E)$.
- Si on prend $K > A/(1 - a - b)$, le coefficient de n est positif, et il sufit que ce soit vrai pour $n = n_0$, soit :

$$K((1 - a - b)n_0 - (D + E)) \geq B + 2M - An_0.$$

- On peut se contenter de prendre $M = 0$ (c'est compatible avec les conditions initiales), et ensuite fixer n_0 pour que $(1 - a - b)n_0 > D + E$; puis, $K \geq \frac{B - An_0}{(1-a-b)n_0-(D+E)}$ suffit.

Hérédité : suite

- On obtient, en majorant :

$$\begin{aligned} C(n) &\leq An + B + K(an + D) + M + K(bn + E) + M \\ &= (A + K(a + b))n + B + K(D + E) + 2M. \end{aligned}$$

- Est-ce qu'on peut choisir K et M de telle sorte que cette majoration soit plus petite que $Kn + M$?
- Ça revient à : pour tout $n > n_0$, $(K(1 - a - b) - A)n \geq M + B + K(D + E)$.
- Si on prend $K > A/(1 - a - b)$, le coefficient de n est positif, et il sufit que ce soit vrai pour $n = n_0$, soit :

$$K((1 - a - b)n_0 - (D + E)) \geq B + 2M - An_0.$$

- On peut se contenter de prendre $M = 0$ (c'est compatible avec les conditions initiales), et ensuite fixer n_0 pour que $(1 - a - b)n_0 > D + E$; puis, $K \geq \frac{B - An_0}{(1-a-b)n_0-(D+E)}$ suffit.
- (Ça ne donne pas la "bonne" constante K, mais ça prouve bien $C(n) = O(n)$; pour obtenir la bonne constante, c'est encore un peu plus technique)

Conclusion...

- Sur notre problème : on a bien un algorithme qui, dans tous les cas, utlise $O(n)$ comparaisons.
- D'après le théorème de Roura, le bon ordre de grandeur est $16n + o(n)$ avec $k = 2$.
- L'algorithme directement inspiré du tri rapide est beaucoup plus efficace en pratique : $2(1 + \ln(2)) \simeq 3.38$, mais c'est seulement "en moyenne" (de la même manière que le tri rapide est plus rapide "en moyenne" que la plupart des tris classiques).
- Utilisation du théorème maître : dans la pratique, c'est un outil très puissant qui évite des calculs fastidieux...

Trier 5 éléments en 7 comparaisons ?

Remarque : quand on a déjà k valeurs triées, et qu'on veut trouver où s'insère une $k+1$-ème valeur parmi eux, il suffit de $\lceil \log_2(k+1) \rceil$ comparaisons (par dichotomie).
On cherche à trier 5 valeurs a, b, c, d, e.

- Comparer a à b, c à d (2 comparaisons)
- Comparer le minimum de (a, b) au minimum de (c, d) : on obtient $a' < b' < c'$ et $a' < d'$, en renommant (3-ème comparaison)
- En 2 comparaisons, trouver la position de e dans la séquence (a', b', c') (5 comparaisons au total)
- Il reste à trouver la position de d' parmi les valeurs plus grandes que a' ; comme il y en a au plus trois, il suffit de 2 comparaisons.
- (Sous forme de programme, ça donne des **if** imbriqués, c'est illisible ; mais c'est optimal !)