Algorithmique IV (UE-41) - TD 6.

TD 6. Tris empiriques et complexité des tris comparatifs [1]

EXERCICE 1. Si L est une liste de n éléments d'un ensemble $(X, \leqslant)$ totalement ordonné.

(1) Combien y-a-t-il de comparaisons du type $L[i] \leqslant L[j]$ possibles si les indices $(i, j) \in [\![1, n]\!]^2$?

(2) Même question pour les indices $(i, j) \in [\![1, n]\!]^2$ tels que $i < j$?

Solution. **(1)** Comme $\#([\![1, n]\!]^2) = (\#[\![1, n]\!])^2 = n^2$, il a n^2 comparaisons possibles.

(2) Pour tout indice i du couple (i, j), les indices j satisfaisant la condition $i < j$ sont donc des éléments de l'intervalle $[\![i + 1, n]\!]$ dont le cardinal est $n - i$. L'ensemble des couples (i, j) qui satisfont la condition requise est donc la réunion des ensembles deux-à-deux disjoints suivants :

$$\bigsqcup_{i=1}^{n-1} (\{i\} \times [\![i + 1, n]\!]) .$$

Comme ils forment une partition, la formule de sommation nous fournit son cardinal

$$\sum_{i=1}^{n-1} 1 \times (n - i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}.$$

EXERCICE 2. Soit n un entier naturel. On considère l'algorithme de génération d'une permutation aléatoire de S_n suivant : on construit la permutation identité, i.e. la liste $L := [1, 2, \ldots, n]$ et pour toute valeur de i allant de 1 à $n - 1$, on tire un nombre j au hasard dans l'intervalle $[\![i, n]\!]$ et on échange les termes d'indices i et j de la liste, autrement dit $L \leftarrow (L[i], L[j]) \circ L$ où $(L[i], L[j])$ désigne une transposition.

(1) Écrivez un algorithme `GenPerm(n)` qui renvoie une permutation "aléatoire" de S_n sur la base de la description précédente. On utilisera sans l'écrire l'algorithme auxiliaire `Alea(a,b)` qui renvoie une valeur de l'intervalle $[\![a, b]\!]$ avec la probabilité uniforme $\frac{1}{b-a+1}$ (de complexité $\Theta(1)$). Quelle est la complexité de l'algorithme `GenPerm(n)` ?

1. Version du 14 mars 2025, 14 : 04

(2) Démontrez que la liste obtenue après ces $n - 1$ transpositions est une permutation aléatoire de $\mathfrak{S}_n$, au sens où la distribution de probabilités sur les permutations est la distribution uniforme : chaque évènement élémentaire a une probabilité de $1/(n!)$.

Indication : remarquez que l'entier à la position i a été tiré au hasard parmi les $n - i + 1$ valeurs à partir de sa position (lui compris).

Solution. **(1)** L'algorithme de *Fisher-Yates*, ou *mélange de Knuth*, est le suivant :

```
ALGORITHME GenPerm(n):liste
DONNEES
 · n: entier
VARIABLES
 · L: liste
 · i: entier
DEBUT
 · Allouer(L,n)
 · i ← 1
 · TQ (i < n) FAIRE
 ·  · L[i] ← i
 ·  · i ← i + 1
 · FTQ
 · i ← 1
 · TQ (i < n) FAIRE
 ·  · j ← Alea(i,n)
 ·  · Echanger(L,i,j)
 ·  · i ← i + 1
 · FTQ
 · RENVOYER(L)
FIN
```

La création et l'initialisation de la permutation identité a un coût linéaire en la taille n de la liste L. On réalise le test de boucle exactement n fois et on effectue $n - 1$ transpositions à l'issu d'un tirage aléatoire en $\Theta(1)$, l'algorithme de Fisher-Yates est donc de complexité linéaire $T(n) = \Theta(n)$.

(2) Considérons une permutation $\sigma \in S_n$ quelconque et calculons la probabilité que l'algorithme renvoie la liste $L = \sigma$. Au premier passage dans la boucle, la position $j \in [\![1, n]\!]$ où se trouve la valeur $\sigma(1)$ a une probabilité $\frac{1}{n}$ d'être renvoyée par l'appel `Alea(1,n)` et à l'étape $i \in [\![1, n - 1]\!]$ la valeur $\sigma(i)$ est située dans la sous-liste $L[i : n]$, sa position a donc pour probabilité $\frac{1}{n-i+1}$ d'être renvoyée par l'appel `Alea(i,n)`. Chaque tirage étant indépendant des précédents, on a finalement pour probabilité de

choisir une permutation particulière

$$\prod_{i=1}^{n-1} \frac{1}{n-i+1} = \frac{1}{n!}$$

EXERCICE 3. **(1)** Écrivez un algorithme Propager(L, g, d) qui balaie la liste L de l'indice g à l'indice $d-1$ en échangeant les termes adjacents $L[k]$ et $L[k+1]$ si $L[k] > L[k+1]$. L'algorithme devra renvoyer un booléen indiquant s'il y a eu ou non des échanges.

(2) Démontrez que votre algorithme s'arrête.

(3) À l'aide d'un invariant de boucle adéquat, démontrez qu'après cette propagation, on a :

$$L[d] = \max\{L[i] \mid i \in [\![g, d]\!]\}. \tag{1}$$

(4) Démontrez qu'après l'exécution de l'algorithme du tri par propagation (le tri à bulles), la liste L est triée.

Solution. **(1)** On suppose que l'algorithme `Echange(@L,i,j)` échange les valeurs de la liste d'indices i et j :

```
ALGORITHME Propager(@L,g,d):booléen
DONNEES
 · L: liste
 · g,d: entiers
VARIABLES
 · EX: booléen
 · k: entiers
DEBUT
 · k ← g
 · EX ← FAUX
 · TQ (k  < d) FAIRE
 ·  · SI (L[k] > L[k+1]) ALORS
 ·  ·  · Echanger(L,k,k+1)
 ·  ·  · EX ← VRAI
 ·  · FSI
 ·  · k ← k + 1
 · FTQ
 · RENVOYER EX
FIN
```

(2) La condition de sortie de la boucle est $k \geqslant d$. Comme d n'est pas modifiée et que k est incrémentée de 1 à chaque passage dans la boucle, la suite des valeurs $k-d$ est strictement croissante et non bornée, la condition sera nécessairement satisfaite.

(3) On considère le prédicat

$$L[k] = \max\{L[i] \mid i \in [\![g, k]\!]\}. \tag{2}$$

Pour $k = g$ la proposition est vraie, supposons qu'elle soit vraie en entrant dans la boucle. Si la valeur $L[k]$ est supérieure à $L[k+1]$, d'après l'hypothèse de récurrence, l'échange assure que la valeur maximale est cette fois en position $k+1$ et l'incrémentation de k montre que la propriété (2) reste vraie pour $k+1$. La sortie de boucle se faisant pour la valeur $k = d$, l'égalité (2) fournit (1).

(4) Le tri à bulles propage la valeur maximale de l'intervalle $[\![1, d]\!]$ de la liste L en dernière position d pour toutes les valeurs d allant de $|L|$ à 2. Après chaque propagation, la valeur maximale est en dernière position de l'intervalle $[\![1, d]\!]$, le prochain maximum étant déplacé dans le sous-intervalle $[1, d-1]$ sa valeur est nécessairement inférieur à $L[d]$ assurant ainsi qu'à la fin des propagations, la liste est triée.

EXERCICE 4. Le *tri cocktail* est une variation autour du tri par propagation. Au lieu de balayer la liste uniquement dans le sens gauche-droite, on alterne un balayage gauche-droite avec un balayage droite-gauche.

(1) Modifiez l'algorithme Propager(L, a, b) pour que la propagation se fasse de gauche à droite si $a < b$ et de droite à gauche sinon. On supposera que $a \neq b$.

(2) Écrivez l'algorithme du tri cocktail.

(3) Soit $L = [2, 3, 4, 5, 1]$. Comparez le nombre d'échanges et le nombre de tests effectués par le tri à bulles et le tri cocktail pour trier L ?

Solution. **(1)** Dans cette version modifiée, la variable `sens` est initialisée à 1 si $a < b$ et -1 sinon. Elle permet d'incrémenter ou de décrémenter la variable k et d'éviter de dupliquer le code selon le sens de parcours de la liste :

```
ALGORITHME Propager(@L,a,b):booléen
DONNEES
 · L: liste
 · a,b: entiers
VARIABLES
 · EX: booléen
```

```
  · k,sens: entiers
DEBUT
  · EX ← FAUX
  · sens ← (a < b) ? +1 : -1
  · k ← a
  · TQ ((sens * (b - k)) > 0) FAIRE
  ·   · SI ((sens * (L[k] - L[k + sens])) > 0) ALORS
  ·   ·   · Echanger(L, k, k + sens)
  ·   ·   · EX ← VRAI
  ·   · FSI
  ·   · k ← k + sens
  · FTQ
  · RENVOYER EX
FIN
```

(2) L'écriture de l'algorithme ne pose pas de difficultés particulières, il s'agit de la version optimisée, qui ne réalise les balayages que sur la zone qui reste à trier et s'arrête si aucun échange n'a eu lieu. Noter que l'on pourrait se dispenser de renvoyer la liste puisqu'elle est passée par adresse.

```
ALGORITHME TriCocktail(@L):liste
DONNEES
  · L: liste
VARIABLES
  · EX: booléen
  · g,d: entiers
DEBUT
  · EX ← VRAI
  · g ← 1
  · d ← #L
  · TQ ((g  < d) ET EX) FAIRE
  ·   · EX ← Propager(L,g,d)
  ·   · d ← d - 1
  ·   · SI ((g < d) ET EX) ALORS
  ·   ·   · EX ← Propager(L,d,g)
  ·   ·   · g ← g + 1
  ·   · FINSI
  · FINTQ
  · RENVOYER L
FIN
```

(3) Dans la table 1, les cellules sur fond vert (resp. rouge et gris) matérialisent une comparaison sans échange (resp. avec échange et rangées). Le nombre d'échanges reste identique, en revanche la liste est rangée dans l'ordre plus rapidement.

Tri propagation						Tri cocktail					
i	1	2	3	4	5	i	1	2	3	4	5
	2	3	4	5	1		2	3	4	5	1
	2	3	4	5	1		2	3	4	5	1
	2	3	4	5	1		2	3	4	5	1
	2	3	4	5	1		2	3	4	5	1
	2	3	4	1	5		2	3	4	1	5
	2	3	4	1	5		2	3	1	4	5
	2	3	4	1	5		2	1	3	4	5
	2	3	1	4	5		1	2	3	4	5
	2	3	1	4	5		1	2	3	4	5
	2	1	3	4	5		1	2	3	4	5
	1	2	3	4	5						

Table 1. Trace de l'exécution du tri à bulles et du tri cocktail.

EXERCICE 5. Le *tri à peigne* est une autre déclinaison du tri par propagation, plus efficace que le tri cocktail. L'objectif est de faire remonter les bulles plus rapidement en avançant avec un pas plus grand que 1. Ce pas décroît d'un facteur donné après chaque balayage de la liste pour atteindre la valeur 1 et revenir au comportement usuel du tri par propagation. Ainsi, au lieu de comparer les termes contigus $L[i]$ et $L[i+1]$, on compare les termes $L[i]$ et $L[i+pas]$ où le pas est initialisé à $\lfloor n/\rho \rfloor$ avec $1 < \rho < 2$ et mis à jour après chaque passe sur la liste par $pas \leftarrow \max\{1, \lfloor pas/\rho \rfloor\}$.

Une fois que la variable pas a atteint la valeur 1, le tri se comporte comme le tri par propagation optimisé, mais la condition d'entrée dans la boucle se résume à savoir s'il y a eu un échange lors du précédent passage, sans limiter progressivement la propagation en fin de liste. Adaptez l'algorithme `Propager` en conséquence, puis écrivez l'algorithme du tri à peigne avec deux paramètres, la liste à trier et le coefficient de réduction ρ.

Solution. L'algorithme `PropagerPeigne` est plus simple que son ascendant, on n'a pas à se préoccuper de la borne à droite :

```
ALGORITHME PropagerPeigne(@L,pas):booléen
DONNEES
  · L: liste
  · pas: entier
```

```
VARIABLES
 · EX: booléen
 · i: entier
DEBUT
 · EX ← FAUX
 · i ← 1
 · TQ (i + pas  <= #L) FAIRE
 ·  · SI (L[i] > L[i + pas]) ALORS
 ·  ·  · Echanger(L, i, i + pas)
 ·  ·  · EX ← VRAI
 ·  · FSI
 ·  · i ← i + pas
 · FTQ
 · RENVOYER EX
FIN
```

L'algorithme s'en déduit facilement. Il faut noter qu'une fois que le pas `pas` a atteint la valeur 1, le tri se comporte comme le tri par propagation optimisé usuel, à ceci près que l'on ne borne pas le balayage sur la droite. NB. `[x]` désigne la partie entière de `x` et on pourrait se dispenser de renvoyer la liste puisqu'elle est passée par adresse.

```
ALGORITHME TriPeigne(@L, rho):liste
DONNEES
 · L: liste
 · rho: réel
VARIABLES
 · pas: entier
 · EX: booléen
DEBUT
 · pas ← #L
 · EX ← VRAI
 · TQ ((pas > 1) OU EX) FAIRE
 ·  · pas ← max([pas / rho], 1)
 ·  · EX ← PropagerPeigne(L, p)
 · FINTQ
 · RENVOYER L
FIN
```

EXERCICE 6. **(1)** Soit $(X, \preceq)$ un ensemble totalement ordonné fini de cardinal n. Montrez qu'il existe une unique bijection croissante entre l'ensemble X et l'ensemble $[\![1, n]\!]$ muni de l'ordre naturel $\leqslant$.

(2) Démontrez qu'une liste L d'éléments d'un ensemble $(X, \preceq)$ totalement ordonné est triée si et seulement si L est une application croissante.

Solution. **(1)** Commençons par l'existence d'une telle bijection croissante. L'ensemble X étant totalement ordonné et fini, sa relation d'ordre est nécessairement un bon ordre, i.e. toute partie non-vide admet un plus petit élément. Ceci va nous permettre de construire inductivement une bijection croissante $x : X \to [\![1, n]\!]$ grâce à l'algorithme du tri sélection par le minimum. On définit $x_1 := \min X_1$ où $X_1 := X$ et $\forall i \in [\![1, n-1]\!]$, on pose $X_{i+1} := X_i \backslash \{x_i\}$ et $x_{i+1} := \min X_{i+1}$. Par construction, les x_i sont deux-à-deux distincts et au nombre de n, l'application x est bien une bijection. D'autre part, elle est croissante par construction.

Démontrons l'unicité de la bijection x par l'absurde. Supposons qu'il existe une bijection croissante $x' : [\![1, n]\!] \to X$ telle que $x' \neq x$, alors on désigne par $i \in [\![1, n]\!]$ le plus petit de ces entiers tel que $x'_i \neq x_i$, i.e. tel que $x'_i \neq \min X \backslash X_{i-1}$. Puisque x' est surjective, il existe $k \in [\![1, n]\!]$ tel que $x'_k = \min X \backslash X_{i-1}$. Par hypothèse $i < k$, mais on remarque que $x'_i > x'_k$, ce qui est contradictoire puisque x' est supposée croissante.

(2) Si on considère L comme une application de $[\![1, n]\!] \to X$, dire que la liste L est triée équivaut à affirmer que

$$\forall (i, j) \in [\![1, n]\!]^2 \quad i \leqslant j \Rightarrow L[i] \preceq L[j],$$

autrement dit que L est une application croissante.

NB. Ces résultats prouvent que l'étude des algorithmes de tri comparatifs ne perd pas en généralité si l'on se restreint à des permutations de S_n.

EXERCICE 7. Construisez les arbres de décision des trois tris empiriques, i.e., le tri par sélection dans sa version où le min est sélectionné à chaque étape, le tri par propagation (optimisé, cf. algorithme `Propager` plus haut pour le test) et le tri par insertion (on insère en partant de la fin) pour les listes qui décrivent le groupe $\mathfrak{S}_3$.

Solution. Par convention, le fils droit (resp. gauche) correspond à une comparaison entre termes de la liste qui est satisfaite (O) (resp. n'est pas satisfaite (N)).

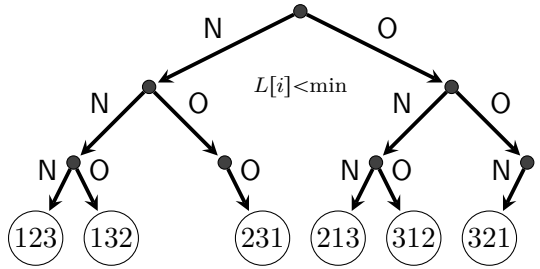

Figure 1. Arbre de décision du tri sélection pour $n = 3$.

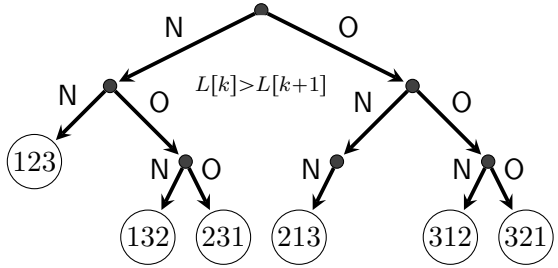

Figure 2. Arbre de décision du tri propagation pour $n = 3$.

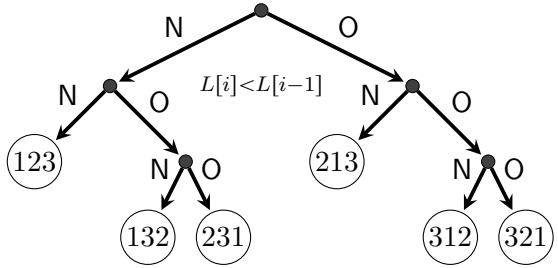

Figure 3. Arbre de décision du tri insertion pour $n = 3$.