Algorithmique III. L2 Informatique I41.

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 $\frac{1}{b-a+1}$.

1. Version du 5 avril 2023, 07 : 48

(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 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
```

(2) Considérons une permutation $s \in S_n$ quelconque et calculons la probabilité que l'algorithme renvoie la liste $L = s$. Au premier passage dans la boucle, la position $j \in [\![1, n]\!]$ où se trouve la valeur $s(1)$ 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 $s(i)$ 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 $d-1$, compare les termes adjacents $L[k]$ et $L[k+1]$ en les échangeant 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 `s` est initialisée à 1 si $a < b$ et -1 si $a > b$. Elle permet d'incrémenter ou de décrémenter la variable k selon le sens du parcours et de modifier la nature des tests pour éviter de réécrire 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,s: entiers
DEBUT
 · EX ← FAUX
 · s ← (a < b) ? +1 : -1
 · k ← a
 · TQ ((s * (b - k)) > 0) FAIRE
```

```
  · · SI ((s * (L[k] - L[k + s])) > 0) ALORS
  · · · Echanger(L, k, k + s)
  · · · EX ← VRAI
  · · FSI
  · · k ← k + s
  · 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.

```
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 suivante, 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.

EXERCICE 5. Le *tri à peigne* est une amélioration du tri par propagation. On dispose d'une liste L de n termes. Au lieu de comparer successivement les termes $L[i]$ et $L[i+1]$, on compare les termes $L[i]$ et $L[i+k]$ où k est initialisé à $\lfloor n/\rho \rfloor$ avec $1 < \rho < 2$ et mis à jour après chaque passe sur la liste par $k \leftarrow \max\{1, \lfloor k/\rho \rfloor\}$. Une fois que le pas k a atteint la valeur

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.

1, la condition de boucle se limite à savoir s'il y a eu un échange lors du précédent passage. Écrivez cet algorithme avec deux paramètres en entrée, la liste à trier et le facteur de réduction.

Solution. Voici l'algorithme et on rappelle que, sauf mention du contraire, les listes en algorithmique sont indexées de 1 à leur taille :

```
ALGORITHME TriPeigne(L, coef):liste
DONNEES
 · L: liste
 · coef: réel
VARIABLES
 · k: entier
 · EX: booléen
DEBUT
 · k ← #L
 · EX ← VRAI
 · TQ ((k > 1) OU EX) FAIRE
 · · k ← max([k/coef], 1)
 · · EX ← FAUX
 · · i ← 1
 · · TQ (i + k <= #L)  FAIRE
 · · · SI (L[i] > L[i + k]) ALORS
 · · · · Echanger(L[i], L[i + k])
 · · · · EX ← VRAI
```

```
·  ·  · FINSI
·  ·  · i ← i + 1
·  · FINTQ
· FINTQ
· RENVOYER L
FIN
```

EXERCICE 6. **(1)** Soit $(X, \preceq)$ un ensemble totalement ordonné et 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)** Par définition, si X est fini et de cardinal n, il est en bijection avec l'intervalle $[\![1, n]\!]$. Comme X est fini et totalement ordonné, il admet un minimum x_1 pour la relation d'ordre sur X. On définit alors inductivement pour $i \in [\![1, n]\!]$, une application $x : [\![1, n]\!] \to X$ par $x_i := \min X \backslash X_{i-1}$ où $X_0 := \varnothing$ et $X_i := X_{i-1} \cup \{x_i\}$ (à chaque étape on cherche le minimum de X, que l'on numérote puis que l'on retire de X, c'est l'algorithme du tri sélection !). 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 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.

(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. Les résultats de cet exercice justifient le fait qu'il est inutile d'étudier les algorithmes de tris comparatifs sur d'autres données que les entiers naturels.

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)).

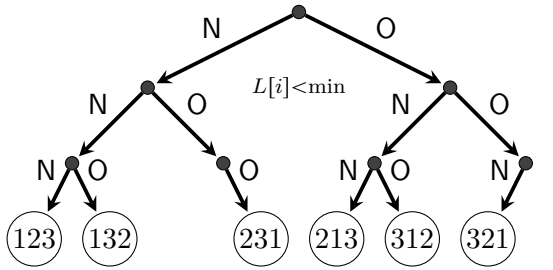

Figure 1. Arbre de décision du tri sélection pour $n = 3$.

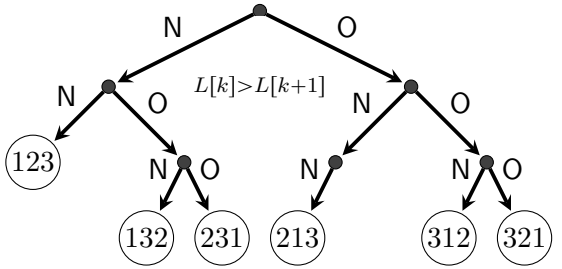

Figure 2. Arbre de décision du tri propagation pour $n = 3$.

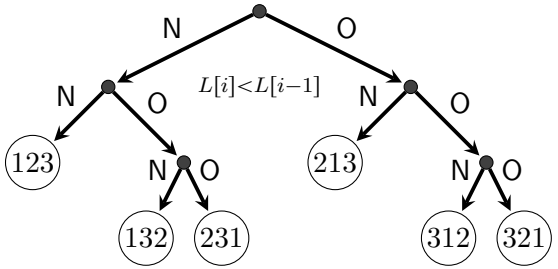

Figure 3. Arbre de décision du tri insertion pour $n = 3$.