Méthodes de tri

- Le problème du tri d'un ensemble est un problème pour lequel on connaît de nombreuses solutions (manuelles) depuis longtemps
  - possibilité de comparaisons entre méthodes
  - support de méthodes plus complexes

Spécification (1)

- tri = construction d'une nouvelle liste
  - vérifiant la propriété d'être triée
  - contenant exactement les mêmes éléments que la liste initiale (c'est-à-dire obtenue par permutation des éléments)

```
opérations
  tri : liste → liste
axiomes
  ∀ l :liste
  est-triée(tri(l)) ∧ est-permut(l, tri(l))
```


Spécification (2)

- Etant donnée une relation d'ordre, une liste triée répond à la spécification suivante :

```
extension type liste
avec
  _ ≤ _ : élément x élément → booléen
opérations
  est-triée : liste → booléen
sémantique
  est-triée(l) =
    si l = listevide alors vrai
    sinon soit l = cons(e, l') ;
      e ≤ premier(l') ∧ est-triée(l')
    fsi
```


Spécification (3)

- Une permutation d'une liste peut être définie de la manière suivante :

```
extension type liste
opérations
  _ ∈ _       : élément x liste → booléen
  est-permut : liste x liste → booléen
sémantique
  e ∈ l =
    si estvide(l) alors faux
    sinon e = premier(l) ∨ e ∈ fin(l) fsi
  est-permut(l, l') =
    si estvide(l) alors estvide(l')
    sinon si ¬ premier(l) ∈ l' alors faux
    sinon soit l' = l1 & [premier(l)] & l2
      est-permut (fin(l), l1 & l2) fsi
```


Tris simples par sélection

- Principe
  - Recherche du plus petit élément de la liste
  - Déplacement de cet élément en tête de liste
  - Tri du reste de la liste
- Famille d'algos en fonction de la manière dont sont effectuées les opérations de
  - sélection
  - déplacement

Spécification

```
extension type liste
opérations
  min           : liste ↛ élément
  supprimer-min : liste ↛ liste
  tri-sélect    : liste → liste
préconditions
  min(l) : ¬ estvide(l)
  supprimer-min(l) : ¬ estvide(l)
axiomes
  ∀ e : élément, l : liste
  e ∈ l ⇒ min(l) ≤ e
  ¬ estvide(l) ⇒ est-permut(cons(min(l), supprimer- min(l)), l)
  tri-sélect (l) =
    si estvide(l) alors listevide
    sinon cons(min(l), tri-sélect(supprimer-min(l))
    fsi
```

Méthodes de tri – tris simples par sélection

Exemple d'implantation

```
void Echanger (int I, int J, int[] tab) {
  int U = tab [J];
  tab [J] = tab [I];
  tab [I] = U;
}

// tri par sélection simple
void Tri (int[] tab, int longueur) {
  int J, K, L;
  for (K = 0; K < longueur - 1; K++) {
    J = K;
    for (L = K + 1; L < longueur; L++) {
      if (tab[L] < tab [J])
          J = L;
    }
    Echanger (J, K, tab);
  }
}
```

Méthodes de tri – tris simples par sélection

Exemple

Rang de tête	Rang + petit	liste
1	5	[] 55, 40, 15, 30, 10
2	3	[10] 40, 15, 30, 55
3	4	[10, 15] 40, 30, 55
4	4	[10, 15, 30] 40, 55
5	Fin	[10, 15, 30, 40] 55

Méthodes de tri – tris simples par sélection

Complexité

- 2 opérations fondamentales :
  - la comparaison de deux éléments
    - Recherche dans une liste de longueur p : p-1
    - Au total : $\sum_{p=2}^{n}(p-1)=\sum_{p=1}^{n-1}p=\frac{n*(n-1)}{2}$
    - Complexité en $O(n^2)$
  - le transfert d'un élément d'un emplacement à un autre
    - n - 1 appels à `Echanger` * 3 transferts = 3 * (n-1)
    - Complexité en O(n)

Méthodes de tri – tris simples par sélection

Tri bulle

Principe

- Recherche du minimum et placement en tête simultanés
- Parcours de la liste depuis la fin en comparant chaque élément avec celui qui précède et échange si pas dans le bon ordre

```
void TriBulle (int[] tab, int longueur) {
  int K, L;

  for (K = 0; K < longueur - 1; K++) {
    for (L = longueur - 1; L >= K + 1; L--) {
      if (tab[L] < tab[L-1])
        Echanger (L-1, L);
    }
  }
}
```

Méthodes de tri – tri bulle

Exemple

Rang de tête	liste
1	[] 55, 40, 15, 30, 10
1	[] 55, 40, 15, 10, 30
1	[] 55, 40, 10, 15, 30
1	[] 55, 10, 40, 15, 30
1	[10] 55, 40, 15, 30
2	[10] 55, 15, 40, 30
2	[10, 15] 55, 40, 30
3	[10, 15] 55, 30, 40
3	[10, 15, 30] 55, 40
4	[10, 15, 30, 40] 55

Méthodes de tri – tri bulle

Complexité

- Comparaisons : idem sélection ordinaire
- Transferts : nb d'appels à `Echanger`
  - au pire :
    - égal au nb de comparaisons (liste en ordre inverse)
    - complexité au pire en $O(n^2)$
  - au mieux :
    - aucun échange (liste triée)
    - complexité au mieux en O(1)
  - en moyenne :
    - égal à n(n-1)/4
    - complexité en moyenne en $O(n^2)$
- Arrêt si pas d'échanges
- Peu d'intérêt dans le cas général

Méthodes de tri

Tris simples par insertion

- Principe
  - Sélection du premier élément de la liste
  - Recherche de la position d'insertion
  - Tri du reste de la liste
- Famille d'algos en fonction de la manière dont l'élément est placé

Méthodes de tri - Tris simples par insertion

Spécification

```
extension type liste
opérations
   placer          : élément x liste ↛ liste
   tri-insert-iter : liste x liste ↛ liste
   tri-insert      : liste → liste
préconditions
   placer(e, l) : est-triée(l)
   tri-insert-iter(l', l) : est-triée(l')
sémantique
   ∀ e : élément, l : liste
   est-permut(cons(e, l), placer(e, l)) ∧ est-triée(placer(e, l))
   tri-insert-iter(l', l) =
        si estvide(l) alors l'
        sinon tri-insert-iter(placer(premier(l), l'), fin(l))
        fsi
   tri-insert(l) = tri-insert-iter(listevide, l)
```

Méthodes de tri - Tris simples par insertion

Insertion séquentielle

- Parcours séquentiel de la liste triée jusqu'à trouver la bonne place
- En général, parcours depuis la fin de la liste et décalage des éléments au fur et à mesure

```
void TriInsertSéq (int[] tab, int longueur) {
  int K, L, Val;

  for (K = 1; K < longueur; K++) {
    Val = tab[K];
    L = K;
    while (L != 0 && Val < tab[L - 1]) {
      tab[L] = tab[L-1];
      L = L - 1;
    }
    tab[L] = Val;
  }
}
```

Méthodes de tri – Insertion séquentielle

Exemple

Rang d'él[t]	Où le placer	liste
		[55] 40, 15, 30, 10
2	1	[40, 55] 15, 30, 10
3	1	[15, 40, 55] 30, 10
4	2	[15, 30, 40, 55] 10
5	1	[10, 15, 30, 40, 55]

Méthodes de tri – Insertion séquentielle

Complexité - Comparaisons

- au mieux (liste déjà triée) :
  - n-1
  - complexité au mieux en O(n)
- au pire (chaque élément est placé en tête) :
  - k-1 comparaisons pour placer le $k^{ième}$
  - Au total : $\sum_{k=1}^{n} (k-1) = \sum_{k=1}^{n-1} k = \frac{n*(n-1)}{2}$
  - complexité en $O(n^2)$
- en moyenne :
  - n(n+3)/4
  - complexité en $O(n^2)$

Méthodes de tri – Insertion séquentielle

Complexité - Transferts

- pour un élément particulier :
  - nb de transferts = nb de comparaisons + 1
- au total
  - nb de transferts = nb de comparaisons + n-1
- mêmes complexités que pour le nb de comparaisons

Méthodes de tri – Insertion séquentielle

Bilan

- en moyenne moins bon que le tri par sélection ordinaire
- plus performant si liste presque triée
- on peut éviter des transferts avec une liste chaînée, au prix d'un espace mémoire pour les pointeurs en O(n)

Méthodes de tri - Tris simples par insertion

Insertion dichotomique

- recherche dichotomique sur la liste en cours de construction pour trouver la bonne place
- impose une représentation contiguë de la liste
  - n'évite pas le déplacement des éléments supérieurs pour insérer le nouvel élément

Méthodes de tri – Insertion dichotomique

Exemple d'implantation

```
public void TriInsertDicho (int[] tab, int longeur) {
  int J, K, Debut, Milieu, Fin, Val;

  for (K = 1; K < longueur; K++) {
    if (tab[K] < tab[K - 1]) {
      Val = tab[K];
      Debut = 0;
      Fin = K - 1;
      while (Debut < Fin) {
        Milieu = (Debut + Fin) / 2;
        if (tab[Milieu] < Val) Debut = Milieu + 1;
        else Fin = Milieu;
      }
      for (J = K; J >= Debut + 1; J--)
        tab[J] = tab[J-1];
      tab[Debut] = Val;
    }
  }
}
```

Méthodes de tri – Insertion dichotomique

Exemple

Rang d'él[t]	Où le placer	liste
		[55] 40, 15, 30, 10
2	1	[40, 55] 15, 30, 10
3	1	[15, 40, 55] 30, 10
4	2	[15, 30, 40, 55] 10
5	1	[10, 15, 30, 40, 55]

Méthodes de tri – Insertion dichotomique

Complexité - Comparaisons

- au mieux (liste déjà triée)
  - n-1
  - complexité au mieux en O(n)
- au pire
  - recherche du kième demande $\lceil \log_2(k-1) \rceil$
  - au total : $n-1+\sum_{k=1}^{n-1} \lceil \log_2(k-1) \rceil$
  - complexité en $O(n.\log_2 n)$
- complexité en moyenne identique à la complexité dans le pire des cas

Méthodes de tri – Insertion dichotomique

Complexité - Transferts

- comme pour le tri par insertion séquentielle, sauf que les éléments déjà à leur place ne sont pas déplacés
  - complexité au mieux en O(1)
  - complexités en moyenne et au pire en $O(n^2)$

Méthodes de tri – Insertion dichotomique

Bilan

- excellent pour le nb de comparaisons
  - ($O(n.\log_2 n)$ contre $O(n^2)$ pour les autres)
- pénalisé par le nb de transferts
  - $O(n^2)$
  - inévitable (représentation contiguë)

Méthodes de tri

Tri rapide (quicksort)

- Principe
  - Choix d'un élément pivot
  - Partage de la liste en 2 sous-listes contenant
    - Les éléments inférieurs au pivot
    - les éléments supérieurs au pivot
  - Tri des 2 sous-listes
- Famille d'algorithmes en fonction
  - du choix du pivot
  - de la construction des deux sous-listes

Méthodes de tri - tri rapide

Choix du pivot

- But = séparer la liste initiale en 2 sous-listes de taille identique
- Exemple de choix du pivot :
  - Examiner le 1er, le dernier, et l'élément du milieu
  - Choisir l'élément médian comme pivot
  - Ordonner les éléments
    - le plus petit en tête de liste
    - le plus grand en fin de liste
    - le pivot au milieu de la liste

Méthodes de tri - tri rapide

Tri des 2 sous-listes

- on ne connaît pas la place finale du pivot
  - on construit la liste des éléments inférieurs à partir du début de la liste
  - on construit la liste des éléments supérieurs à partir de la fin de la liste
- Les éléments qui sont
  - en début de liste et inférieurs au pivot
  - en fin de liste et supérieurs au pivot

  … restent en place
- On échange deux éléments qui ne sont pas à leur place
- A l'issue du tri des sous-listes, le pivot est à sa place définitive

Méthodes de tri - tri rapide

Exemple

Opération	liste
Liste initiale	[55 40 15 30 10 5 25 35]
Choix pivot	[30 40 15 35 10 5 25 55]
Echange	[30 25 15 35 10 5 40 55]
Echange	[30 25 15 5 10 35 40 55]
Listes obtenues	[30 25 15 5 10] 35 [40 55]
Liste à 2 éléments	[30 25 15 5 10] 35 40 55
Choix pivot	[10 25 15 5 30] 35 40 55
Echange	[10 5 15 25 30] 35 40 55
Listes obtenues	[10 5] 15 [25 30] 35 40 55
Liste à 2 éléments	5 10 15 [25 30] 35 40 55
Liste à 2 éléments	5 10 15 25 30 35 40 55

Méthodes de tri - tri rapide

Complexité

- Comparaisons :
  - Au mieux et en moyenne : $n.\log_2 n$
  - Au pire : max = n^2
- Transferts :
  - Au mieux : n
  - En moyenne : $n.\log_2 n$
  - Au pire : n^2
- Mémoire :
  - Mémorisation des bornes des sous-listes restant à traiter
  - Pile dont la taille est bornée par $\log_2 n$
  - Complexité en mémoire en O(n)

Méthodes de tri

Tri par tas (heapsort)

- Constat :
  - dans le tri par sélection, on obtient des infos qui sont perdues par la suite
- Principe :
  - structurer les éléments dans la liste restant à trier de manière à rendre la recherche et l'extraction la plus efficace possible
  - structuration sous la forme d'un arbre binaire parfait de telle sorte que tout élément soit plus petit que tous ses descendants
  - liste contiguë = représentation par niveau d'un arbre binaire parfait

Méthodes de tri – tri par tas

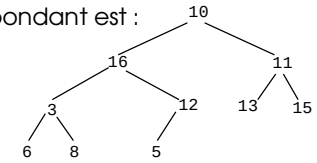
Construction d'un tas

- **Tas** = arbre binaire partiellement ordonné de manière croissante (décroissante)
  - chaque élément est plus petit (plus grand) que tous ses descendants
  -
```
ordonner(a) =
  si max(a, filsg, filsd) == a alors rien
  sinon si max(a, filsg, filsd) == filsg alors
      échanger(a, filsg)
      ordonner(filsg)
  sinon si max(a, filsg, filsd) == filsd alors
      échanger(a, filsd)
      ordonner(filsd)
```
  - Exécuter `ordonner` sur tous les nœuds en partant des feuilles (on part du milieu du tableau)

Méthodes de tri – tri par tas

Ex. de construction d'un tas

- Soit la liste :
  - `10 16 11 3 12 13 15 6 8 5`
- L'arbre correspondant est :



- Le tas correspondant est :
  - `16 12 15 8 10 13 11 6 3 5`

Méthodes de tri – tri par tas

Principe de l'algo

- Après avoir transformé l'arbre en arbre partiellement ordonné, l'élément maximum se retrouve à la racine (tête de liste)
- Echange de l'élément max avec la feuille du dernier niveau la plus à droite
- Application de `ordonner(a)` sur la racine du nouvel arbre (les deux sous-arbres sont déjà partiellement ordonnés)

Méthodes de tri – tri par tas

Exemple

Tri de la liste

`55 40 15 30 10 5`

Méthodes de tri – tri par tas

Complexité - `ordonner`

- Hauteur d'un arbre binaire parfait
  - $h(A) = \lfloor \log_2 \text{taille}(A) \rfloor$
- Nb de comparaisons de l'opération ordonner :
  - sur un nœud : 2 comparaisons
  - au pire $2\log_2 p$ avec p la taille de l'arbre sur lequel on l'applique

Méthodes de tri – tri par tas

Complexité - Construction

- nb de comparaisons < 4n
  - complexité en nb de comparaisons en O(n)
- en moyenne 1 transfert pour 2 comparaisons
  - complexité en nb de transferts en O(n)

Méthodes de tri – tri par tas

Complexité - Extraction

- Extraction de l'élément à la K[ième] place de la liste résultat
  - au plus $2\lfloor \log_2 (K-1) \rfloor$ comparaisons
  - nb de comparaisons au pire : $2\sum_{K=2}^{n} \lfloor \log_2 (K-1) \rfloor$
  - complexité au pire en nb de comparaisons en $O(n.\log_2 n)$
  - complexité au mieux et en moyenne identiques
- Dans `ordonner`
  - nb transferts = moitié des comparaisons et 2(n-1) dans la boucle principale
  - complexité en terme de transferts aussi en $O(n.\log_2 n)$
- Complexité globale en $O(n.\log_2 n)$

Méthodes de tri - Comparaison des méthodes

Complexité

	comparaisons			transferts		
	min	moy	max	min	moy	max
Sélection ordinaire	n^2			n		
Tri bulle	n^2			0	n^2	
Insertion séquentielle	n	n^2		n	n^2	
Insertion dichotomique	n	$n.\log_2 n$		0	n^2	
Tri par tas	$n.\log_2 n$			$n.\log_2 n$		
Tri rapide	$n.\log_2 n$		n^2	n	$n.\log_2 n$	n^2

- Longueur de la liste
  - Méthodes simples → évolution en n^2
  - Méthodes efficaces → évolution en $n.\log_2 n$

Méthodes de tri - Comparaison des méthodes

Temps d'exécution

	500 éléments		2000 éléments		8000 éléments	
	Non triés	triés	Non triés	triés	Non triés	triés
Sélection ordinaire	0,34 s.	0,33 s.	5,4 s.	5,5 s.	106 s.	105 s.
Tri bulle	1,5 s.	0,36 s.	26 s.	6,2 s.	600 s.	129 s.
Insertion séquentielle	0,73 s.	0,0 s.	13 s.	0,06 s.	290 s.	0,22 s.
Insertion dichotomique	0,52 s.	0,0 s.	9,2 s.	0,01 s.	232 s.	0,04 s.
Tri par tas	0,05 s.	0,07s.	0,35 s.	0,38 s.	2,0 s.	1,9 s.
Tri rapide	0,03 s.	0,02 s.	0,24 s.	0,07 s.	1,2 s.	0,5 s.
$n^2/500^2$	1		16		256	
$n.\log_2 n/500.\log_2 500$	1		5		24	

Méthodes de tri - Comparaison des méthodes

Facteurs déterminants

- Sensibilité à l'ordre initial des éléments
  - Insensible
    - Tri séquentiel ordinaire
    - Tri par tas
  - Amélioration
    - Tri bulle → amélioration d'un facteur 4
    - Tri rapide → amélioration d'un facteur 2 (même nb de comparaisons mais plus de transferts)
    - Tri par insertion → amélioration d'un facteur voisin de la taille de la liste
- Temps de comparaison contre temps de transfert
  - Tris par sélection et par insertion sont en n^2 en moyenne mais temps d'exécutions différents