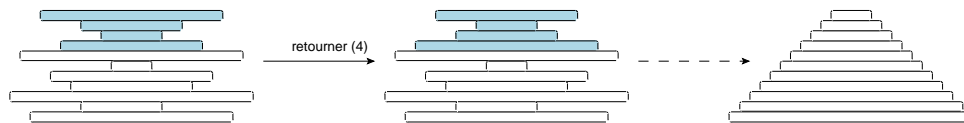


## TD n°6 - Partiel 2007

### 1 Tri crêpes

On veut trier une pile de crêpes de façon à obtenir une pile de crêpes triées par ordre croissant de taille. On dispose pour cela uniquement d'une spatule pour retourner les crêpes. On définit l'opération *retourner*( $n$ ) qui consiste à glisser la spatule sous les  $n$  premières crêpes et à retourner le haut de la pile.



#### Question 1.1

Donner un algorithme permettant de trier une pile de  $n$  crêpes à l'aide de l'opération *retourner*. On donnera sa complexité au pire cas en nombre d'opérations *retourner*.

**Solution :**

```

début
  pour  $k$  de  $n$  à 2 faire
     $i \leftarrow$  indice de la plus grande crêpes parmi les  $k$  premières en partant du haut.;
    /*on fait passer la plus grande des  $k$  premières crêpes sur le dessus. */
    retourner( $i$ ) ;
    /*la plus grande des  $k$  premières crêpes se retrouve en  $k^{eme}$  position dans la pile. */
    retourner( $k$ );
  fin
fin

```

#### Algorithme 1 : tri crêpes

Quoi qu'il arrive, l'algorithme effectuera  $2(n - 1)$  opérations *retourner*. □

#### Question 1.2

Justifiez la correction de votre algorithme.

**Solution :** *Invariant* : Les crêpes  $k + 1$  à  $n$  sont les  $n - k$  plus grandes crêpes triées par ordre de taille croissante.

- L'invariant est initialement vrai car la pile  $k + 1$  à  $n$  est vide pour  $k = n$ .
- Supposons l'invariant vrai pour  $k = j$ . on place la plus grande des  $j$  premières crêpes en position  $j$  dans la pile par retourner( $i$ ) puis retourner( $k$ ). L'invariant est alors vrai pour  $k = j - 1$ .

- L'algorithme s'arrête après l'itération  $k = 2$ , l'invariant est vérifié pour  $k = 1$  donc les crêpes 2 à  $n$  sont les  $n - 1$  plus grande crêpes triées par ordre de taille croissante. La dernière sur le dessus est bien la plus petite.

□

### Question 1.3

Vous devez désormais trier une pile de crêpes ayant chacune une face brûlée. Pour soigner la présentation on vous demande désormais de trier les crêpes tout en vous assurant que chaque crêpe a sa face brûlée tournée vers le bas. Quelle serait alors la complexité de l'algorithme ?

**Solution :** lorsque la plus grande crêpe est sur le dessus de la pile, on s'assure que sa face brûlée est tournée vers le haut. Après le second retournement la crêpe se retrouvera dans le bon sens.

**début**

**pour**  $k$  de  $n$  à 2 **faire**

$i \leftarrow$  indice de la plus grande crêpes parmi les  $k$  premières en partant du haut;

*/\*on fait passer la plus grande des  $k$  premières crêpes sur le dessus. \*/*

;

retourner( $i$ ) ;

**si** dessus non brûlé **alors**

retourner(1) ;

**fin**

*/\*la plus grande des  $k$  premières crêpes se retrouve en  $k^{eme}$  position dans la pile.*

*\*/*

retourner( $k$ ) ;

**fin**

*/\*La dernière crêpe peut elle aussi être brûlée \*/*

;

**si** dessus brûlé **alors**

retourner(1) ;

**fin**

**fin**

#### **Algorithme 2 : tri crêpes**

L'algorithme effectuera  $2(n - 1)$  opérations retourner pour trier la pile plus, au pire, 1 opération retourner par crêpe pour cacher la face brûlée, soit  $3n - 2$  retournements au total dans le pire des cas. □

## 2 Fanfare

Une fanfare est composée de  $n$  musiciens de tailles  $t_1, t_2, \dots, t_n$ . Pour les jours de fête l'orchestre dispose de  $m$  uniformes ( $m \geq n$ ) de tailles  $u_1, u_2, \dots, u_m$ . Chaque année certains mu-

siciens s'en vont et sont remplacés par d'autres, il faut alors ré-attribuer à chaque musicien le costume qui lui sied le mieux.

René, le percussionniste, pense qu'il faut chercher à minimiser la différence moyenne entre la taille d'un musicien et celle de son costume :

$$\frac{1}{n} \sum_{i=1}^n |t_i - u_{\alpha(i)}|$$

où  $\alpha(i)$  est l'indice du costume attribué au musicien de taille  $t_i$ . Il propose pour cela un algorithme glouton qui consiste à chercher  $i$  et  $j$  minimisant  $|t_i - u_j|$ . On attribue alors l'uniforme  $j$  au musicien  $i$  et on itère jusqu'à ce que tout le monde ait reçu un uniforme.

**Question 2.1** L'algorithme de René est-il optimal ?

**Solution :** *non, par exemple avec  $t_1 = 1, t_2 = 4, u_1 = 3, u_2 = 6$ , l'algorithme glouton associe  $t_2$  à  $u_1$  puis  $t_1$  à  $u_2$  quand la solution optimale est d'associer  $t_1$  à  $u_1$  et  $t_2$  à  $u_2$ .*  $\square$

Anne, la corniste, trouve plus équitable de chercher à minimiser le carré moyen des écarts :

$$\frac{1}{n} \sum_{i=1}^n (t_i - u_{\alpha(i)})^2$$

**Question 2.2** Montrez par un exemple l'avantage de cette fonction objective par rapport à la précédente. L'algorithme glouton est-il optimal pour la nouvelle fonction objective ?

**Solution :** *C'est fonction défavorise les grand écarts entre la taille d'un musicien et celle de son uniforme. l'algo glouton n'est pas optimal, il donne les mêmes solutions avec les deux fonctions objectives.*  $\square$

**Question 2.3** Montrer que si il y a autant de costumes que de musiciens, l'algorithme consistant à classer les musiciens et le costumes par taille croissante et à attribuer au musicien  $i$  le costume  $i$  est optimal pour la seconde fonction objective.

**Solution :** *On trie les musiciens et les costumes par ordre de taille croissante Soit  $S$  le score obtenu en assignant le costume  $i$  au musicien  $i$  et  $S'$  le score d'une solution optimale où un musicien  $i$  recoit un costume  $j$  et vice-versa ( $i \leq j$ ).*

$$\begin{aligned} S' - S &= (t_i - u_j)^2 + (t_j - u_i)^2 - ((t_i - u_i)^2 + (t_j - u_j)^2) \\ \equiv S' - S &= 2(u_i - u_j)(t_i - t_j) \end{aligned}$$

*Or  $(u_i - u_j) \leq 0$  et  $(t_i - t_j) \leq 0$  donc  $S' - S \geq 0$*   $\square$

**Question 2.4** Donner un algorithme donnant une solution optimale dans le cas  $m \geq n$ .

Solution :

```
début
  pour j de 1 à m faire
    M0j ← ∞;
  fin
  pour i de 1 à n faire
    pour j de 1 à m faire
      si Mi(j-1) < M(i-1)(j-1) + (ui - ti)2 alors
        Mij ← Mi(j-1);
      sinon
        Mij ← M(i-1)(j-1) + (ui - ti)2;
      /*On note les assignations d'uniforme dans A */
      Ai ← j;
    fin
  fin
fin
```

*Algorithme 3 : Fanfare()*

□

### 3 Sous-vecteur de somme maximale

Étant donné un tableau  $T$  de  $n$  entiers relatifs, on cherche  $\max\{\forall i, j \in \{1 \dots n\} \mid \sum_{k=i}^j T[k]\}$ .

Par exemple pour le tableau suivant :

2	18	-22	20	8	-6	10	-24	13	3
---	----	-----	----	---	----	----	-----	----	---

l'algorithme retournerait la somme des éléments 4 à 7 soit 32.

**Question 3.1** Donner un algorithme retournant la somme maximale d'éléments contigus par une approche *diviser pour régner*.

Solution :

```
début
  si  $g > d$  alors
    retourner 0;
  sinon
     $Sig = d$  retourner  $\max\{T[d], 0\}$ ;
  fin
   $m \leftarrow (g + d)/2$ ;
   $gmax \leftarrow 0$  ;
   $s \leftarrow 0$  ;
  pour  $i$  de  $m$  à  $g$  faire
     $s \leftarrow s + T[i]$ ;
     $gmax \leftarrow \max\{gmax, sum\}$ ;
  fin
   $m \leftarrow (g + d)/2$ ;
   $dmax \leftarrow 0$  ;
   $s \leftarrow 0$  ;
  pour  $i$  de  $m + 1$  à  $d$  faire
     $s \leftarrow s + T[i]$ ;
     $dmax \leftarrow \max\{dmax, sum\}$ ;
  fin
  retourner  $\max\{gmax + rmax, SommeMax(g, m), SommeMax(m + 1, d)\}$ ;
fin
```

*Algorithme 4 : SommeMax(g, d)*

□

**Question 3.2** Donner un algorithme retournant la somme maximale d'éléments contigus par programmation dynamique.

Solution :

```
début
   $max \leftarrow 0$ ;
   $maxlocal \leftarrow 0$ ;
  pour  $i$  de 1 à  $n$  faire
     $maxlocal \leftarrow \max\{maxlocal + T[i], 0\}$ ;
     $max \leftarrow \max\{max, maxlocal\}$  ;
  fin
  retourner  $max$ ;
fin
```

*Algorithme 5 : SommeMax()*

□

**Question 3.3** Comparer la complexité Asymptotique au pire cas des deux approches.

**Solution :**  $O(n \log(n))$  pour la solution par diviser pour régner,  $O(n)$  pour la solution par programmation dynamique. La complexité de l'approche diviser pour régner peut être ramenée en  $O(n)$  en mémorisant les résultats d'une étape à l'autre pour ne pas avoir à recalculer de somme sur tout le vecteur à chaque fois. On obtient ainsi un coup constant par itération.  $\square$

## 4 La Bibliothèque

La bibliothèque planifie son déménagement. Elle comprend une collection de  $n$  livres  $b_1, b_2, \dots, b_n$ . Le livre  $b_i$  est de largeur  $w_i$  et de hauteur  $h_i$ . Les livres doivent être rangés dans l'ordre donné (par valeur de  $i$  croissante) sur des étagères identiques de largeur  $L$ .

**Question 4.1** On suppose que tous les livres ont la même hauteur  $h = h_i, 1 \leq i \leq n$ . Montrer que l'algorithme glouton qui range les livres côte à côte tant que c'est possible minimise le nombre d'étagères utilisées.

**Solution :** Pour un livre la solution gloutonne est optimale (une étagère). Si on a  $n$  livres rangés de manière optimale, il y a deux façon de ranger le  $n + 1^{\text{eme}}$  livre : soit on le met sur l'étagère courante soit sur une nouvelle étagère. Le choix optimal est de le mettre sur l'étagère courante si on peut car cela n'augmente pas le coût de la solution. ce choix donne une solution optimale pour  $n + 1$  livres. Il correspond au choix fait par l'algorithme glouton.  $\square$

**Question 4.2** Maintenant les livres ont des hauteurs différentes, mais la hauteur entre les étagères peut se régler. Le critère à minimiser est alors l'encombrement, défini comme la somme des hauteurs du plus grand livre de chaque étagère utilisée.

**4.2.1** Donner un exemple où l'algorithme glouton précédent n'est pas optimal.

**Solution :**  $L = 2$

$$w_1 = w_2 = w_3 = 1$$

$$h_1 = 1, h_2 = 2, h_3 = 3$$

$\square$

**4.2.2** Proposer un algorithme optimal pour résoudre le problème, et donner son coût. **Solution :**

**début**

*/\*Chaque case  $i$  de  $E$  contient l'encombrement optimal pour ranger les livres  $i$  à  $n$ . \*/*

*/\*On remplit  $E$  en partant de la fin. \*/*

$E[n] \leftarrow h_n;$

**pour  $i$  de  $n - 1$  à  $1$  faire**

$l \leftarrow w_i;$

**pour  $j$  de  $i + 1$  à  $n$  faire**

$l \leftarrow l + w_j;$

**si  $l \leq L$  et  $\max\{h_i, \dots, h_j\} + E[j + 1] \leq E[i]$  alors**

$E[i] \leftarrow \max\{h_i, \dots, h_j\} + E[j + 1];$

*/\*au final,  $et[i]$  contient le prochain changement d'étagère suivant  $i$ . \*/*

$et[i] \leftarrow j + 1;$

**fin**

**fin**

**fin**

**fin**

**Algorithme 6 : Bibliothèque()**

□

**Question 4.3** On revient au cas où tous les livres ont la même hauteur  $h = h_i, 1 \leq i \leq n$ . On veut désormais ranger les  $n$  livres sur  $k$  étagères de même longueur  $L$  à minimiser, où  $k$  est un paramètre du problème. Il s'agit donc de partitionner les  $n$  livres en  $k$  tranches, de telle sorte que la largeur de la plus large des  $k$  tranches soit la plus petite possible. Proposer un algorithme pour résoudre le problème, et donner son coût en fonction de  $n$  et  $k$ .

**Solution :** On utilise une approche récursive pour évaluer tous les cas possibles. On remplit pour cela une matrice  $M$  où chaque case  $M_{nk}$  contient le coût minimal pour caser  $n$  livres sur  $k$  étagères.

$$\forall j \in [1..k], M_{1j} = w_1$$

$$\forall i \in [1..n], M_{i1} = \sum_{l=1}^i w_l$$

$$\forall i \in [1..n], \forall j \in [1..k], M_{ij} = \min_{t=1}^i \max(M_{t(j-1)}, \sum_{u=t+1}^i w_u)$$

Cette définition donne un algorithme en  $O(kn^3)$  ( $n^2$  pour calculer chaque case,  $kn$  cases) mais on peut diminuer sa complexité en calculant à priori le tableau  $W$ ,  $W_i = \sum_{j=1}^i w_j$ . On obtient alors une complexité en  $O(kn^2)$

```

début
  /*calcul de W */
   $W_0 = 0;$ 
  pour  $i$  de 1 à  $n$  faire
     $W_i = W_{i-1} + w_i;$ 
  fin
  /*initialisation de la matrice */
  pour  $i$  de 1 à  $n$  faire
     $M_{i1} = W_i;$ 
  fin
  pour  $j$  de 1 à  $k$  faire
     $M_{1j} = w_1;$ 
  fin
  pour  $i$  de 2 à  $n$  faire
    pour  $j$  de 2 à  $k$  faire
       $M_{ij} = \infty;$ 
      pour  $x$  de 1 à  $i - 1$  faire
         $e \leftarrow \max(M_{x(j-1)}, W_i - W_x);$ 
        si  $M_{ij} > e$  alors
           $M_{ij} = e;$ 
          /*et on enregistre dans  $D$  le dernier changement d'étagère pour pouvoir
          reconstruire la solution optimale. */
           $E_{ij} = x;$ 
        fin
      fin
    fin
  fin

```

*Algorithme 7 : Bibliothèque2()*

□

## 5 Bonus

On considère un ensemble  $S$  de  $n \geq 2$  entiers distincts stockés dans un tableau ( $S$  n'est pas supposé trié). Résoudre les questions suivantes :

**Question 5.1** Proposer un algorithme en  $\mathcal{O}(n)$  pour trouver deux éléments  $x$  et  $y$  de  $S$  tels que  $|x - y| \geq |u - v|$  pour tout  $u, v \in S$ . **Solution :** Pour que  $|a - b|$  soit maximum il faut que

$a$  (Resp  $b$ ) soit maximum et que  $b$  (Resp  $a$ ) soit minimum. Le problème se résume donc à une recherche du min et du max qui se fait en  $\mathcal{O}(n)$ . □

**Question 5.2** Proposer un algorithme en  $\mathcal{O}(n \log n)$  pour trouver deux éléments  $x$  et  $y$  de  $S$  tels que  $x \neq y$  et  $|x - y| \leq |u - v|$  pour tout  $u, v \in S, u \neq v$ . **Solution :** Si  $|c - d|$ , est

minimum pour  $c$  et  $d$  appartenant à  $S$  alors il n'existe pas  $e$  appartenant à  $S$  tel que  $a > e > b$  ou  $a < e < b$  : sinon  $|c - e| < |c - d|$ . On trie donc  $S$  en  $\mathcal{O}(n \times \log n)$  et on cherche ensuite le min des  $s_i - s_{i+1}$ , ( $s_i$  et  $s_{i+1}$  étant deux entiers consécutifs de  $S$ ) en  $\mathcal{O}(n)$ . On obtient donc un algorithme en  $\mathcal{O}(n \times \log n)$ .  $\square$

**Question 5.3** Soit  $m$  un entier arbitraire (pas nécessairement dans  $S$ ), proposer un algorithme en  $\mathcal{O}(n \log n)$  pour déterminer s'il existe deux éléments  $x$  et  $y$  de  $S$  tels que  $x + y = m$ . **Solution :**

On commence par trier  $S$  en  $\mathcal{O}(n \times \log n)$ . Ensuite pour tout  $y$  appartenant à  $S$  on cherche si  $m - y$  appartient à  $S$  par dichotomie ( $S$  est trié), la dichotomie se faisant en  $\mathcal{O}(\log n)$  cette opération ce fait en  $\mathcal{O}(n \times \log n)$ . A la fin on obtient bien un algorithme en  $\mathcal{O}(n \times \log n)$ .  $\square$

**Question 5.4** Proposer un algorithme en  $\mathcal{O}(n)$  pour trouver deux éléments  $x$  et  $y$  de  $S$  tels que  $|x - y| \leq \frac{1}{n-1}(\max(S) - \min(S))$ . **Solution :** Il existe deux méthodes pour résoudre cette question.

**Première méthode** Cette méthode est basée sur la médiane. On note  $S_1$  l'ensemble des éléments de  $S$  inférieurs à la médiane et  $S_2$  les éléments de  $S$  supérieurs à la médiane. Il est à noter que  $\|S_1\| = \|S_2\| = \lceil \frac{\|S\|}{2} \rceil$ .

**début**

Calculer le min et le max de  $S$  ;

**si**  $n = 2$  **alors**

retourner  $(\text{Min}, \text{Max})$  ;

**sinon**

extraire la médiane de  $S$  ; calculer  $S_1, S_2$  ;

relancer l'algo sur  $S_1$  ou  $S_2$  suivant le cas ;

**fin**

**fin**

Preuve de l'algorithme : on pose  $M(S)$  la moyenne pondérée de  $S$ ,  $n = \|S\|$ ,  $a$  le min de  $S$ ,  $b$  le max de  $S$ , et  $m$  la médiane de  $S$ , deux cas sont alors possibles :

-  $n$  impair :

$$n = 2k + 1, \text{ alors } \|S_1\| = \frac{n+1}{2} = k+1$$

$$\text{donc : } M(S_1) = \frac{m-a}{k+1-1} = \frac{m-a}{k}$$

$$\text{et : } \|S_2\| = \frac{n+1}{2} = k+1$$

$$\text{donc : } M(S_2) = \frac{b-m}{k+1-1} = \frac{b-m}{k}$$

$$\text{donc : } \frac{M(S_1) + M(S_2)}{2} = \left( \frac{m-a}{k} + \frac{b-m}{k} \right) \times \frac{1}{2} = \frac{b-a}{2k}$$

$$\text{or : } \frac{b-a}{2k} = \frac{b-a}{n-1} = M(S)$$

$$\text{donc : } 2 \times M(S) = M(S_1) + M(S_2)$$

$$\text{finalement : } M(S_1) \leq M(S) \text{ ou } M(S_2) \leq M(S).$$

- *n pair* :

$$n = 2k, \text{ alors } \|S_1\| = \frac{n+1}{2} = k$$

$$\text{donc : } M(S_1) = \frac{m-a}{k-1}$$

$$\text{et : } \|S_2\| = \frac{n+1}{2}$$

$$\text{donc : } M(S_2) = \frac{b-m}{k-1}$$

$$\text{donc : } \frac{M(S_1) + M(S_2)}{2} = \left( \frac{m-a}{k-1} + \frac{b-m}{k-1} \right) \times \frac{1}{2} = \frac{b-a}{2k-2}$$

$$\text{or : } \frac{b-a}{2k-2} \leq \frac{b-a}{n-1} = M(S)$$

$$\text{donc : } 2 \times M(S) \geq M(S_1) + M(S_2)$$

$$\text{finalement : } M(S_1) \leq M(S) \text{ ou } M(S_2) \leq M(S).$$

Comme on sait que  $M(S_1) \leq M(S)$  ou  $M(S_2) \leq M(S)$  cela nous permet d'enclencher l'induction : soit sur  $S_1$ , soit sur  $S_2$ . Rechercher la moyenne se fait en  $O(n)$ , chercher le min et le max d'un ensemble aussi, donc :

$$C(n) = C(\lceil \frac{n}{2} \rceil) + O(n)$$

Ce qui donne par master théorème

$$C(n) = O(n)$$

**Deuxième méthode** On divise  $S$  en  $(n-1)$  boites.

**début**

*si* il n'y a qu'une boite **alors**

*on renvoie deux de ses éléments;*

**sinon**

*si* le nombre d'éléments des  $\lfloor \frac{n-1}{2} \rfloor$  premières boites est supérieur à  $\lfloor \frac{n-1}{2} \rfloor$ . **alors**  
*la moyenne pondérée sur les  $\lfloor \frac{n-1}{2} \rfloor$  premières boites est inférieure à celle de  $S$ ,*  
*et on relance l'algorithme sur ces boites;*

**sinon**

*le nombre d'éléments des  $\lceil \frac{n+1}{2} \rceil$  dernières boites est supérieur à  $\lceil \frac{n+1}{2} \rceil$ , et alors*  
*la moyenne pondérée sur les  $\lceil \frac{n+1}{2} \rceil$  dernières boites est inférieure à celle de  $S$ ,*  
*et on relance l'algorithme sur celles-ci;*

**fin**

**fin**

**fin**

Quant à la complexité elle est en  $O(n)$  :

$$C(n) = C(\lceil \frac{n}{2} \rceil + 1) + O(n)$$

Ce qui donne par master théorème

$$C(n) = O(n)$$

□