(Tri par) Tas

Les internautes sont invités à aller visiter la page
`http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/heapsort.html`
pour de bien belles simulations de tri par tas...

EXERCICE 1 *Codons le tri par tas*
On va coder les deux versions du tri par tas vues en cours.
On utilisera des tas au sens usuel : le plus gros élément en haut. On arrivera donc à un tableau trié dans l'ordre croissant à la fin.

1. Tracer "à la main" l'exécution du tri par tas sur le tableau $[1; 2; 10; 3; 4; 5; 6; 7]$.
2. Ecrire trois fonctions fournissant les indices des fils droit, gauche et du père d'un nœud d'indice donné dans un tableau Caml.
   Vérifier qu'elles marchent ET PAS SEULEMENT QU'ELLES MARCHENT A UNE UNITE PRES...
3. Ecrire un programme de tri par tas permettant de trier en place un tableau d'entiers. Ce tri aura la structure suivante :

```
let heap v=let n=vect_length v
    and swap i j=let x=v.(i) in
      begin v.(i)<-v.(j); v.(j)<-x end in
    let build_again i=......
    and sort i=......
    in
    for k=1 to (n-1) do build_again k done;
    for k=0 to (n-2) do sort k done;
   v;;
```

   La sous-procédure `build_again` prend un argument i, et doit faire en sorte que $v[0..i]$ redevienne un tas, sachant que $v[0..i-1]$ est un tas avant l'appel. La sous procédure `sort` a pour rôle de placer la racine du tas $v[0..n-i]$ à sa place définitive dans le tableau trié (c'est-à-dire en position $n-1-i$), puis `sort` reconstitue le tas $v[0..n-i-2]$.
4. Faire des essais...
5. Que se passe-t-il lors du tri par tas d'un tableau déjà trié dans l'ordre croissant (resp. décroissant) ?
6. On passe à la version alternative pour la phase de construction du tas : une procédure `percole` prend en argument deux entiers $i \in [\![0, n-1]\!]$ (en fait $i \leq n/2$) et $t \in [\![i, n-1]\!]$, et fait en sorte que l'arbre de racine $v.[i]$ et dont les étiquettes ont un indice $\leq t$ devienne un tas, sachant qu'avant l'appel, ses deux fils sont des tas. On réalise alors les percolations "du bas vers le haut" pour constituer le tas :

```
let heap2 v=let n=vect_length v
    and swap i j=let x=v.(i) in
      begin v.(i)<-v.(j); v.(j)<-x end in
    let percole i t=......
    in let sort i=......
    in
    for k=(pere (n-1)) downto 0 do percole k (n-1) done;
    for k=0 to (n-2) do sort k done;
   v;;
```

   Compléter... et tester.
7. Tracer à la main la construction du tas sur le tableau $[1; 2; 10; 3; 4; 5; 6; 7]$.
8. Vérifier qu'avec cette deuxième version, la construction du tas ne demande plus que $O(n)$ comparaisons. Faire des essais comparatifs sur les mêmes tableaux entre les deux versions.

Exercice 2 *Tri linéaire d'une partie d'un tableau*

On cherche un algorithme permettant de trier les $n^{4/5}$ plus grands éléments d'un ensemble de n entiers en utilisant $O(n)$ comparaisons.

1. Quel est le coût d'un algorithme naïf ?
2. L'améliorer !

Exercice 3 *Tas ternaires*

Pour réduire la hauteur du tas donc la complexité du tri associé, on se propose d'utiliser des tas "ternaires" : chaque nœud a jusqu'à trois fils.

1. Expliquer comment stocker un tas ternaire dans un tableau en place.
2. Montrer que la hauteur d'un tas ternaire avec n nœuds est majorée par $\dfrac{\ln(2n-1)}{\ln 3}$.
3. Montrer que cela n'améliore pas la complexité du tri par tas (en terme de comparaisons), même avec un facteur multiplicatif constant.

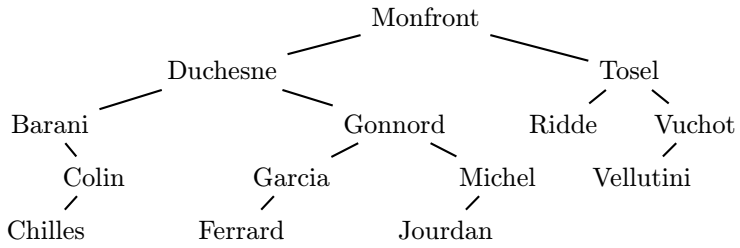
Un petit retour sur les ABR...

Exercice 4

1. Ecrire une procédure prenant en argument un ABR et retournant la liste triée de ses étiquettes en temps $O(n)$. On écrira une procédure itérative utilisant une pile (module `heap`, ou bien fonctions données dans le TD précédent).
2. L'éminent professeur Duchmule prétend pouvoir constituer un ABR à partir d'un tableau de n éléments en temps $O(n)$ (comme pour les tas). Qu'en pensez-vous ?

Exercice 5 *Reprise du dernier exo du TD précédent...*

Le but du jeu était de déterminer le nombre de listes l telles que les insertions successives de ses éléments dans un ABR donne :



On note $insert(l)$ l'ABR obtenu par insertions successives à la racine d'un ABR (initialement vide) des éléments de l.

$(E, \leq)$ est un ensemble ordonné (par exemple A^* muni de l'ordre lexicographique...)

1. Soient l_1 et l_2 sont deux listes disjointes d'éléments de E et $x \in E$ avec "$l_1 < x < l_2$" et l_3 un mélange de l_1 et l_2 au sens des mots : les éléments de l_3 sont ceux de l_1 et l_2, et on ne change pas les positions relatives des éléments de l_1 (resp. l_2). Par exemple, $[4;2;3;7;5;1]$ est un mélange de $[4;7;5]$ et $[2;3;1]$
   Que dire de $insert(x :: l_3)$?
2. Conclure !