TP : Tri par tas

Comme pour le TP sur les tris, vous devez utiliser le paquetage `paq_permutation` écrit lors des précédents TP. Vous pouvez également charger et utiliser le paquetage compilé fourni si le votre ne fonctionne pas bien. Dans ce cas il vous faudra supprimer les droits en écriture sur les fichiers `paq_permutation.*` par exemple avec la commande `chmod u-w paq_permutation.*`.

L'objectif de ce TP est de (re)programmer l'algorithme du tri par tas et d'étudier sa complexité analytiquement et expérimentalement.

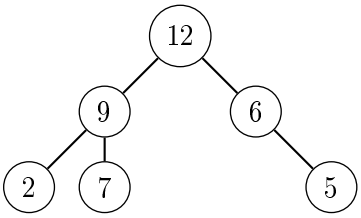
Vous devez créer le paquetage `paq_tri_tas` ainsi qu'un programme principal de test.

Comme pour le TP sur les tris, vous allez utiliser les fonctions `comp` et `echange` lorsque cela sera nécessaire afin de pouvoir compter facilement toutes ces opérations le moment venu. Vous aurez donc sans doute besoin soit de les copier dans votre paquetage `paq_tri_tas`, soit d'utiliser le paquetage `paq_tri` du TP sur les tris.

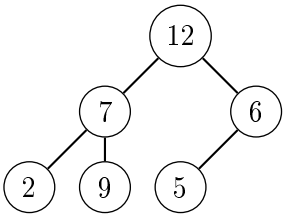
1 Tas binaire

Un tas binaire est un arbre binaire qui possède certaines propriétés supplémentaires :
- la différence de profondeur entre deux feuilles est d'au plus 1,
- les feuilles de profondeurs maximum sont toujours le plus à gauche possible,
- la valeur de chaque nœud doit être supérieure à la valeur de la racine de chacun de ses fils.

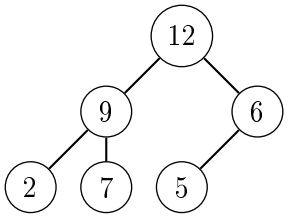
Q1 . Les arbres suivants sont-ils des tas binaires ? Justifiez si ce n'est pas le cas.



-A-



-B-



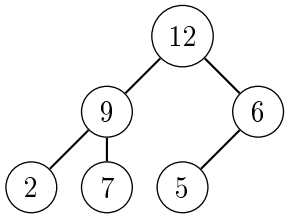
-C-

Q2 . Où se trouve l'élément maximum d'un tas binaire ? Justifiez.

2 Représentation d'un tas binaire par un tableau

Comme tout arbre binaire, un tas binaire peut être représenté dans un tableau unidimensionnel **indicés à partir de 1** : le père d'un nœud en position i a pour enfants un fils gauche en position $2i$ et un fils droit en position $2i + 1$.

Exemple. L'arbre



est codé par le tableau

12	9	6	2	7	5

Les tas binaires sont utilisés pour implanter les files de priorités car ils permettent des insertions en temps logarithmiques et un accès direct au plus grand élément.

Q3 . Ecrire les trois fonctions suivantes retournant respectivement l'indice du père, du fils droit et du fils gauche du nœud d'indice `i` dans le tas `T` :

```
function pere(T : PERMUTATION; i : NATURAL) return NATURAL;

function FD(T : PERMUTATION; i : NATURAL) return NATURAL;

function FG(T : PERMUTATION; i : NATURAL) return NATURAL;
```

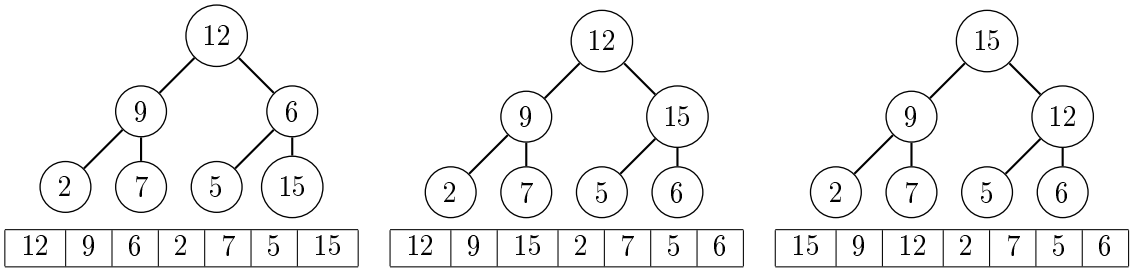
Si l'indice à retourner n'est pas un indice de `T` vous ne devrez pas lever d'exception ni en tenir compte pour l'écriture de ces fonctions.

3 Insertion dans un tas binaire

L'insertion d'un élément dans un tas binaire se ramène à 2 type d'opérations :

1. l'insertion de l'élément dans première cellule vide du tableau codant l'arbre.
2. la *percolation* de cet élément depuis une feuille de l'arbre jusqu'à la racine (si nécessaire). Ces opérations consistent à échanger autant que nécessaire l'élément qui *percole* avec son père *courant* si ce dernier est plus petit que lui.

Dans l'exemple suivant, on ajoute l'élément 15 au tas en 3 étapes :



Q4 . Ecrire la procédure `percoler` qui insère l'élément se trouvant à l'indice `i` à la place appropriée dans `T` pour que les éléments entre l'indice `1` et `i` forment un tas, sachant que les éléments d'indices `1` à `i-1` forment déjà une structure de tas dans `T` :

```
procedure percoler(T : in out PERMUTATION; i : in NATURAL);
```

Q5 . Quelle(s) propriété(s) sur les données doit/doivent être vérifiée(s) pour que votre algorithme effectue respectivement un maximum et un minimum de comparaisons entre éléments de `T` ?

Q6 . Combien de comparaisons sont effectuées dans chacun de ces cas ? Vous donnerez un nombre précis de comparaisons pour pouvoir utiliser ce résultat ultérieurement, ainsi qu'un ordre de grandeur asymptotique.

4 Créer une structure de tas binaire dans un tableau quelconque

Pour ordonner les éléments d'un tableau quelconque en une structure de tas binaire il suffit de procéder par insertions successives comme dans un tri par insertion.

En effet l'élément d'indice `1` pris seul forme un tas car n'ayant aucun fils il respecte toutes les contraintes. Il est donc possible d'utiliser la procédure `percoler` pour insérer dans ce tas de taille 1 l'élément d'indice `2` du tableau, et ainsi de suite jusqu'à ce que le tableau complet forme un tas.

Q7 . Ecrire la procédure `entasser` qui ordonne les éléments du tableau `T` pour former un tas dans `T` :

```
procedure entasser(T : in out PERMUTATION);
```

Q8 . Quelle(s) propriété(s) sur les données doit/doivent être vérifiée(s) pour que votre algorithme effectue respectivement un maximum et un minimum de comparaisons entre éléments de `T` ?

Q9 . Combien de comparaisons sont effectuées dans chacun de ces cas ? Vous donnerez un nombre précis de comparaisons pour pouvoir utiliser ce résultat ultérieurement, ainsi qu'un ordre de grandeur asymptotique.

5 Une première tentative de tri

Vous allez étudier une première idée pour exploiter les avantages de la structure de tas dans le tri d'un tableau.

En effet, sachant où se trouve le plus grand élément du tas, et sachant reformer un tas à partir d'un tableau ou d'une tranche de tableau quelconque, vous pouvez appliquer une stratégie comparable à celle du tri par séléction : trouver le maximum, l'échanger avec la dernière case du tableau etc.

Comme dans le tri par séléction, le tableau pourra être vu comme étant divisé en une partie triée en fin de tableau et une partie non triée en début de tableau.

Q10 . Ecrire la procédure `tritri` s'inspirant du tri par séléction mais utilisant la structure de tas.

```
procedure tritri(T : in out PERMUTATION);
```

Q11 . D'après les calculs de complexité effectués précédemment sur `percoler` et `entasser` donnez une majoration et une minoration du nombre de comparaisons effectuées par `tritri`.

Q12 . Par rapport aux algorithmes de tri que vous connaissez déjà, ce tri vous paraît-il intéressant ?

6 Une autre idée

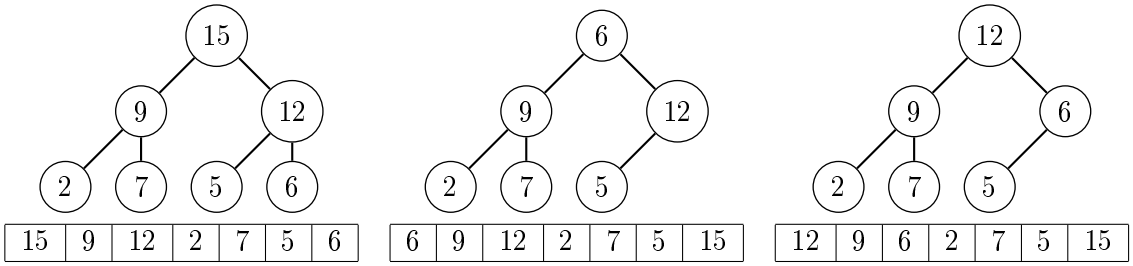
Dans l'algorithme `tritri` précédent, lorsque l'élément maximum courant du tas est évacué vers sa place définitive à la fin du tableau, les éléments restants ne forment plus un tas, mais *presque* un tas, ce qui n'est pas exploité par `tritri`.

Pour essayer de réduire la complexité, l'idée est donc maintenant de tenir compte du fait que la partie non triée du tableau forme presque un tas.

Q13 . Pourquoi la partie non triée ne forme-t-elle pas un tas ?

Pour remédier à ce problème et ré-ordonner la partie non triée du tableau en structure de tas il faut une nouvelle procédure, `tamiser` qui fait un peu la même chose que `percoler` mais en sens inverse. Cette opération consiste à faire descendre de nœud en nœud l'élément se trouvant à la racine jusqu'à la place apropriée dans le tas en l'échangeant autant que nécessaire avec le plus grand fils de sa position courante.

Dans l'exemple suivant, le plus grand élément du tas construit précédemment est échangé avec l'élément se trouvant en dernière case du tableau, puis cet élément qui se retrouve à la racine est tamisé jusqu'à sa place :



L'élément 15 est toujours présent dans le tableau mais dans la partie triée, il n'apparaît donc plus dans le tas.

Q14 . Ecrire la procédure `tamiser` qui reforme un tas dans le tableau `T` sachant que les éléments de `T` sont presque ordonnés en tas comme détaillé précédemment.

```
procedure tamiser(T : in out PERMUTATION);
```

Q15 . Quelle(s) propriété(s) sur les données doit/doivent être vérifiée(s) pour que votre algorithme effectue respectivement un maximum et un minimum de comparaisons entre éléments de `T` ?

Q16 . Combien de comparaisons sont effectuées dans chacun de ces cas ? Vous donnerez un nombre précis de comparaisons pour pouvoir utiliser ce résultat ultérieurement, ainsi qu'un ordre de grandeur asymptotique.

Tout est maintenant prêt pour écrire la procédure de tri par tas classique.

Q17 . Ecrire la procédure `tritas` qui tri le tableau `T` en utilisant correctement la structure de tas binaire :

```
procedure tritas(T : in out PERMUTATION);
```

Q18 . D'après les calculs de complexité effectués précédemment, donnez une majoration et une minoration asymptotiques du nombre de comparaisons effectuées par `tritas`.

Q19 . Par rapport aux algorithmes de tri que vous connaissez déjà, ce tri vous paraît-il intéressant ?

7 Complexité en moyenne - expérimentations

Les questions suivantes sont identiques aux questions sur l'analyse de complexité expérimentale du TP sur les tris.

7.1 Des valeurs exactes

Pour établir expérimentalement la valeur exacte du nombre moyen d'opérations, on peut engendrer les $n!$ tableaux possibles contenant n éléments distincts, évaluer le nombre d'opérations effectuées lors du tri de chacun, et faire la moyenne arithmétique.

Q20 . Programmez une procédure qui traitera tous les tableaux de taille n et calculera les nombres moyens de comparaisons et d'échanges de manière exacte et l'affichera sous la forme `<taille de donnée> <comparaisons> <échanges>`.

Q21 . Affichez le nombre moyen d'opérations nécessaires au tri par tas d'une permutation de taille $n = 2, 3, \ldots, 10$. Vous redirigerez l'affichage dans un fichier de données qui sera ensuite utilisé par gnuplot.

7.2 Des estimations

Malheureusement $n! = \Omega(e^n)$: en pratique, trier tous les tableaux de taille n devient rapidement impossible. Pour des valeurs de n plus grandes, il faut se contenter de valeurs établies au vu d'échantillons. Vous calculerez donc des valeurs approchées du nombre moyen d'opérations en considérant n tableaux de taille n choisis aléatoirement.

Q22 . Programmez une procédure par tri qui traitera tous les tableaux de taille n et calculera les nombres moyens de comparaisons et d'échanges de manière approchée.

Q23 . Affichez le nombre moyen d'opérations nécessaires au tri par tas d'une permutation de taille $n < 15$, $n = 25, 50, 100, 250, 500, 1000, 5000$. Vous redirigerez l'affichage dans un fichier de données qui sera ensuite utilisé par gnuplot.

7.3 Tracé des courbes

Q24 . Tracez les courbes donnant le nombre moyen de comparaisons d'une part et le nombre moyen d'échanges d'autres part sur deux graphiques rassemblant le tri par tas, le tri rapide et le tri par fusion.

Vous pouvez utiliser le paquetage `paq_tri` du TP sur les tris et copier les lignes de code nécessaires à la génération des nombres d'opérations à tracer sur votre TP tris. Mais vous pouvez aussi réutiliser les résultats obtenus lors du TP sur les tris car les tests à effectuer sont identiques.

Q25 . Quelles sont vos conclusions ?