Question de cours

Écrire l'algorithme de suppression de la racine d'un tas max.
Vous écrirez aussi l'algorithme de la fonction auxiliaire que vous utilisez.

Outils à utiliser

Vous avez le droit d'utiliser sans les redéfinir les primitives
FeuilleP, Pere, FilsGauche FilsDroit et *Echanger* et l'indice *IndicePremierSommetLibre*.
$Contenu[Som]$ contient la valeur du sommet Som. La racine est le sommet 0.
Pour simplifier l'écriture de la fonction auxiliaire, vous supposerez que lors de
son exécution un sommet qui n'est pas une feuille a exactement deux fils.

Descendre(S)

```
si !FeuilleP(S) alors
    cas où
    Contenu[FilsGauche(S)] > Contenu[S] >Contenu[FilsDroit(S)]
     | Echanger(S,FilsGauche(S)) ; Descendre(FilsGauche(S))
    fin
    cas où
    Contenu[FilsDroit(S)] > Contenu[S] >Contenu[FilsGauche(S)]
     | Echanger(S,FilsDroit(S)) ; Descendre(FilsDroit(S))
    fin
    cas où
    Contenu[FilsGauche(S)] > Contenu[FilsDroit(S)] >Contenu[S]
     | Echanger(S,FilsGauche(S)) ; Descendre(FilsGauche(S))
    fin
    cas où
    Contenu[FilsDroit(S)] > Contenu[FilsGauche(S)] >Contenu[S]
     | Echanger(S,FilsDroit(S)) ; Descendre(FilsDroit(S))
    fin
fin
```

SupRacine

```
IndicePremierSommetLibre−−;
Echanger(IndicePremierSommetLibre,0);
Descendre(0);
```

Exercice.

Un algorithme s'exécute sur une donnée de taille n_1 avec un nombre d'opérations élémentaires o_1.
Le même algorithme s'exécute sur une donnée de taille $n_2 = 2n_1$ avec un nombre d'opérations élémentaires o_2.

Quelle sera le nombre d'opérations élémentaires o_3 de l'exécution de cet algorithme sur une donnée de taille $n_3 = 3n_1$ dans chacun des cas suivants :

- $o_2 = 2o_1$
- $o_2 = o_1^2$
- $o_2 = o_1 + k$ où k est une constante.
- $o_2 = 4o_1$

Justifiez.

- $o_2 = 2o_1 \Rightarrow$ la complexité de l'algo est dans $\theta(n) \Rightarrow o_3 = 3o_1$
- $o_2 = o_1^2 \Rightarrow$ la complexité de l'algo est dans $\theta(e^n) \Rightarrow o_3 = o_1^3$
- $o_2 = o_1 + k \Rightarrow$ la complexité de l'algo est dans $\theta(log(n)) \Rightarrow o_3 = o_1 + 2k$
- $o_2 = 4o_1 \Rightarrow$ la complexité de l'algo est dans $\theta(n^2) \Rightarrow o_3 = 9o_1$

Problème

Préalable

Pour manipuler un ABR A, on utilisera les primitives suivantes (de complexité dans $\theta(1)$) :
- $VideP(A)$ qui renvoie un booléen (vrai si et seulement si A est vide)
- $FeuilleP(A)$ qui renvoie un booléen (vrai si et seulement si A est réduit à une feuille).
- $SAG(A)$ et $SAD(A)$ qui renvoient les sous ABR (éventuellement vides) respectivement gauche et droite de l'ABR (non vide) A.
- $Val(A)$ qui renvoie la valeur stockée à la racine de l'ABR (non vide) A.

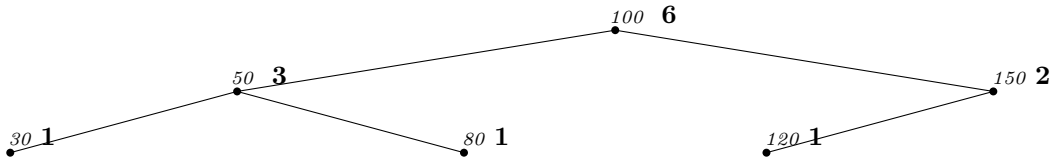
Toutes les complexités demandées devront être fournies si possible en fonction de la hauteur h de l'ABR et sinon en fonction de son nombre n de sommets.

Question 1
Écrire l'algorithme récursif `NbreSommet(A)` qui renvoie le nombre de sommets d'un ABR A.

Donner la classe de complexité de cette algorithme avec un minimum de justifications.

Un ABR amélioré

Dans chaque sommet est stocké le compte du nombre de sommets de l'ABR enraciné en ce sommet.



En gras apparaît le compte du nombre de sommets de l'ABR enraciné dans le sommet correspondant. Les petites valeurs en italique sont celles des sommets de l'ABR.

On dispose des primitives (de complexité dans $\theta(1)$)
- `void Incrementer(A)` augmente de 1 le compte du nombre de sommets stocké dans la racine de A
  Cette primitive n'est définie que pour un ABR **non vide** A.
- `Sommet(v)` qui renvoie un ABR réduit à une feuille de valeur v et de compte de sommets à 1.
- `GrefferSAG(Pere,Fils)` et `GrefferSAD(Pere,Fils)` qui greffent l'ABR *Fils* respectivement à gauche et à droite de l'ABR *Pere*

Question 2
Ecrire l'algorithme `InsererValeur (v, A)` qui insère la valeur v dans l'ABR A.
On supposera que la valeur v est absente de A.

La difficulté de cette question est de maintenir en chaque sommet le bon compte du nombre de sommets de l'ABR qui y est enraciné.
En effet, `GrefferSAG` et `GrefferSAD` **ne** modifient **pas** ce compte.

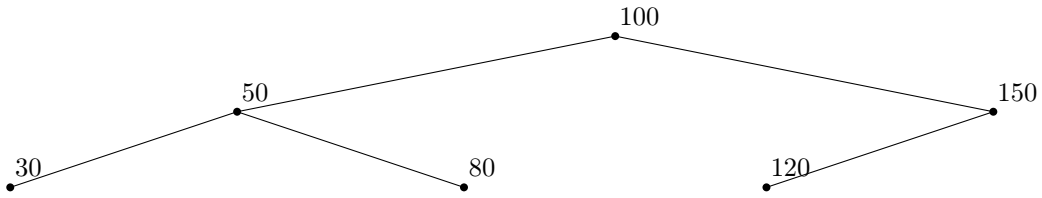
On vous demande d'écrire l' algorithme avec une complexité dans $\theta(h)$.

```
si Val(A) > v alors
|  Incrementer(A);
|  si SAG(A) alors
|  |  InsererValeur(v,SAG(A))
|  sinon
|  |  GrefferSAG(A,Sommet(v))
|  fin
sinon
|  même chose à droite
fin
```

Utilisation de l'ABR amélioré

On dispose maintenant de la primitive [1] `NbreSomPrim(A)` qui renvoie le compte du nombre de sommets de l'ABR A (et 0 si A est vide).

On veut trouver la $k^{\text{ième}}$ (avec $1 \leq k \leq n$) des valeurs (classées par ordre croissant) d'un ABR A de n sommets .



La cinquième des valeurs classées par ordre croissant de l'ABR ci dessus est 120.

Question 3
Écrire un algorithme récursif `KiemeSommet(A,k)` de complexité dans $\theta(h)$ qui renvoie l'ABR enraciné dans le sommet qui a pour valeur la $k^{\text{ème}}$ des valeurs (classées par ordre croissant) qui apparaîssent dans A
(il sera inutile de vérifier dans l'algorithme que k est compris entre 1 et n).
Justifier brièvement sa complexité.

Complications

Question 4
Que devient la complexité de votre algorithme `KiemeSommet` quand vous ne pouvez plus utiliser `NbreSomPrim` mais que vous devez recalculer à chaque fois le nombre de sommets (avec la fonction `NbreSommet` que vous avez écrite au début de ce problème) ?

1. de complexité dans $\theta(1)$