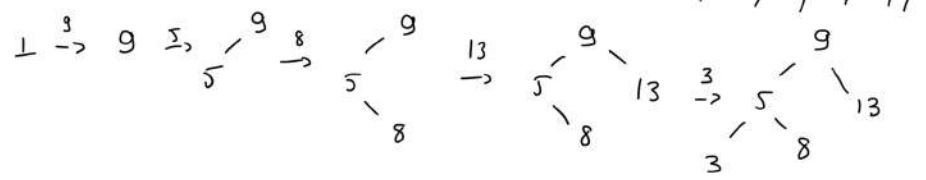
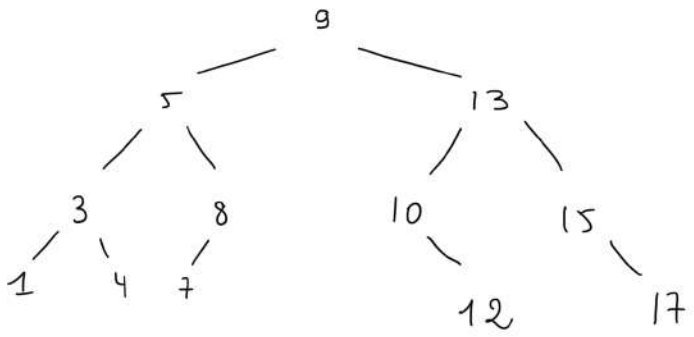
TD - ABR et ARN

1. Ce n'est pas un ABR, le noeud d'étiquette 11 est dans le sous-arbre gauche du noeud d'étiquette 10
2. L'arbre de la question précédente est un parfait exemple
3. Non (sauf si on a 0, 1 ou 2 étiquettes).
4. Non (sauf si on supprime une feuille).
5. Dans l'ABR d'origine, g est le sous-arbre gauche de x et d son sous-arbre droit. Donc toutes les étiquettes de g sont inférieures à x et toutes celles de d sont supérieures à x. Ainsi toutes les étiquettes de g sont inférieures à toutes celles de d, et en particulier à l'étiquette minimale de d. Placer g comme sous-arbre gauche du minimum de d conserve donc bien les propriétés des ABR. Cependant, cette méthode n'est pas efficace car la hauteur de l'arbre résultant est (souvent beaucoup) plus grande que celle de l'arbre d'origine. L'efficacité des algos sur les ABR dépendent pour la plupart de sa hauteur, on n'utilise donc jamais cette méthode.

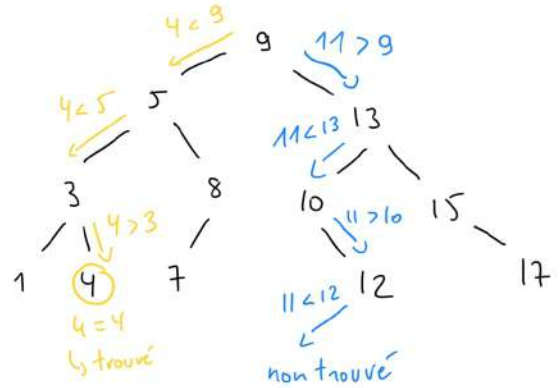
6. • Insertions successives de 9, 5, 8, 13, 3, 4, 15, 7, 10, 17, 1, 6



etc. →

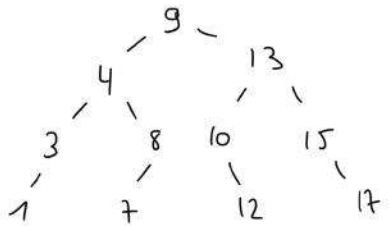


• Recherche de l'étiquette 4 : en jaune
11 : en bleu

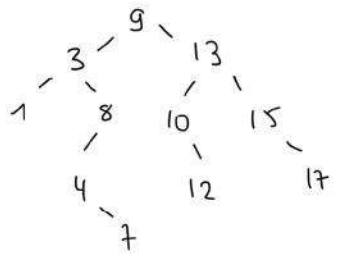


• Suppression de l'étiquette 5 :

* méthode de la remontée du maximum

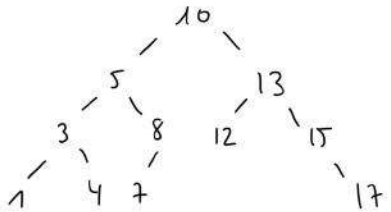


* méthode de la fusion

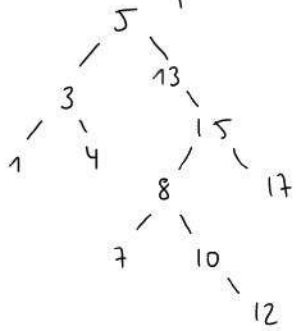


- Suppression de l'étiquette 9 :

* méthode de remontée du minimum



* méthode de la fusion

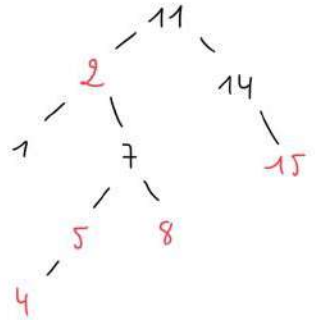


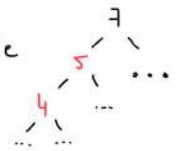
7.

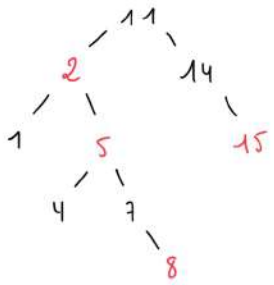
est un ABR mais ne peut pas être coloré en ARN.

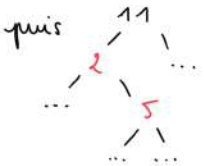
8. • Le premier est un arbre bicolore, on peut vérifier que toutes les propriétés sont vérifiées.
   • Le second n'est pas un arbre bicolore, car le chemin allant de la racine au sous-arbre gauche vide de 1 contient 3 noeuds noirs alors que celui allant de la racine au sous-arbre gauche vide de 14 n'en contient que 2.
   • Le troisième n'est pas un arbre bicolore car le noeud d'étiquette 2 est rouge et son fils droit aussi.
   • Le quatrième n'est pas un arbre bicolore car la racine est rouge.
   • Le cinquième n'est pas un arbre bicolore car ce n'est pas un ABR (0 dans le sous-arbre droit de 2).

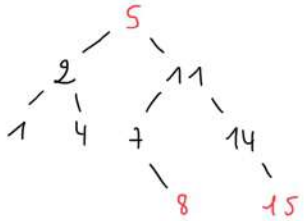
9. Insertion de 4 dans le premier arbre (seul ARN).
   • insertion comme dans un ABR d'une feuille rouge :



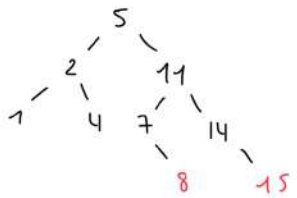
- puis correction des couleurs, par une rotation droite de  qui est un des 4 cas problématiques



puis  qui est aussi un des cas problématiques est corrigé par deux rotations.

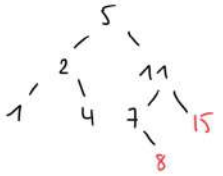


- enfin, la racine est colorée correctement :

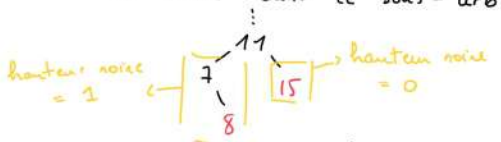


10. Suppression de 14 dans l'arbre obtenu

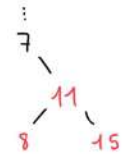
- Suppression de 14 comme la remontée du max dans un ABR :



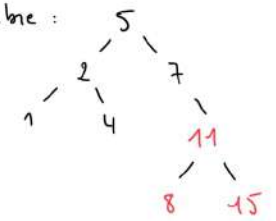
- 14 étant noir, il faut régler le problème de hauteur noire. Le problème se situe dans ce sous-arbre :



On corrige ce problème avec rotation droite et recoloration :

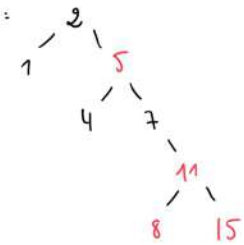


Ce qui donne l'arbre :

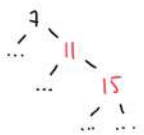


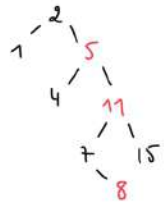
Il y a encore un problème de hauteur noire (la hauteur noire est 2 dans le SAG de 5 et 1 dans son SAD).

On corrige à nouveau, on obtient :

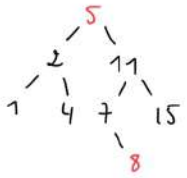


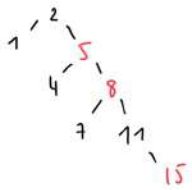
- les hauteurs noires sont corrigées, il reste à corriger les conflits père rouge / fils rouge.

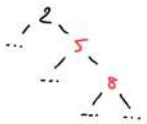
→ correction de  :

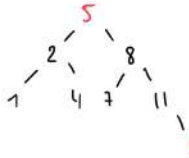


→ puis correction de  :

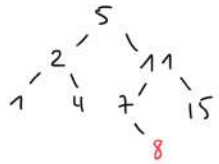


Remarque : on peut aussi choisir de corriger  au début, et on trouverait 

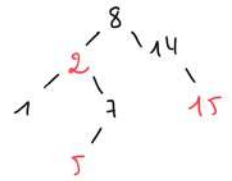
ce qui donne après dernière correction de 

l'arbre  . Le choix n'a pas d'importance, tant que vous préciser bien ce que vous avez choisi de corriger.

- pour finir, on n'oublie pas de remettre la racine en noir :



11. Suppression de 11 dans le premier arbre de la question 8.
    - suppression par remontée du max comme dans un ABR :



    - c'est la feuille 8 qui a été supprimée, elle était rouge donc les hauteurs noires sont correctes.
    - aucun conflit rouge/rouge à corriger
    - la racine est déjà noire : Fini !

12. Le chemin le moins long possible dans un ARN a ne contient que des noeuds noirs et est de longueur $h_N(a)$.
Le plus long possible alterne noeuds rouges et noirs et est de longueur $2h_N(a)$. Ainsi le chemin le plus long ne peut être que 2 fois plus long que le chemin le court.

13.
- Création de la file de priorité (FP)
  - ↳ ABR vide, qui sera organisé selon la relation d'ordre des priorités
  - ↳ $O(1)$
- Ajout d'un élément de priorité p dans la FP
  - ↳ Insertion de p dans l'ABR, on conserve aussi l'élément dans le même noeud
  - ↳ $O(n)$ avec n le nb d'éléments de la FP
- Extraction de l'élément le plus prioritaire de la FP
  - ↳ recherche du min/max (selon le type de FP) puis suppression du noeud et renvoi de l'élément associé
  - ↳ $O(n)$ avec n le nombre d'éléments de la FP

- Test de vacuité
  - ↳ regarder si ABR est vide
  - ↳ $O(1)$

C'est moins efficace que les tas, sauf si l'ABR est équilibré (on passerait en $O(\log n)$ pour l'ajout et l'extraction).

14. Non, car la comparaison de deux étiquettes n'est pas en $O(1)$ mais dépend de la taille des chaînes. Donc la complexité de toutes les opérations augmente. On peut utiliser un trie.

15. Il faut effectuer un parcours infixe.

```
Infixe (ABR a) :
    si a est vide :
        renvoyer liste vide
    sinon :
        renvoyer Infixe (SAG(a))
                 + // concaténation
                 [racine (a)]
                 + // idem
                 Infixe (SAD(a))
```

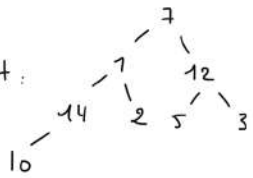
```
Tri (liste l) :
    a ← ABR vide
    Pour chaque élément de l :
        a ← insérer e
    Renvoyer Infixe (a)
```

16. Les comparaisons interviennent lors des insertions, pour la recherche de l'emplacement où mettre la feuille. L'insertion effectue un nombre logarithmique de comparaisons dans le meilleur cas (ABR équilibré) et linéaire dans le pire des cas.
On a donc un algo de tri quasi-linéaire dans le meilleur des cas et quadratique dans le pire des cas.
Un pire des cas est obtenu quand l'ABR est filiforme, ce qui survient par exemple avec une liste déjà triée (dans l'ordre croissant ou non). Le meilleur des cas est obtenu quand l'ABR est équilibré, ce qui survient lorsque le premier élément à insérer est la médiane m, et ainsi de suite le second élément est la médiane des éléments inférieurs à m (ou supérieurs), etc.
On peut améliorer ce tri en maintenant l'ABR équilibré (avec un ARN par exemple).

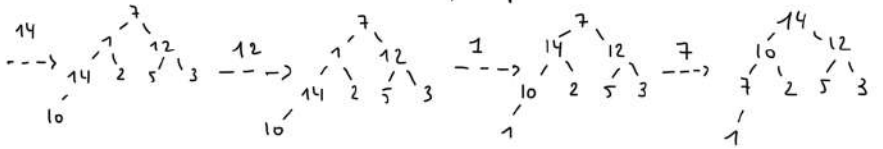
17. Tri de 7, 1, 12, 14, 2, 5, 3, 10.

* Tri par tas

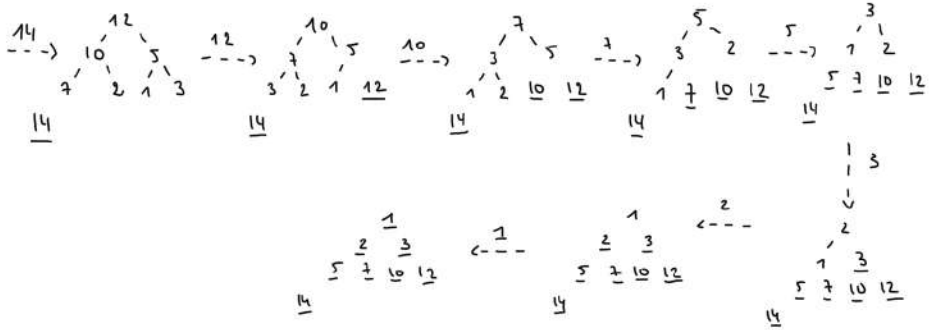
↳ arbre correspondant :



↳ transformation en tas-max par percolations vers le bas successives :

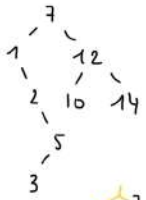


↳ extractions successives du maximum :

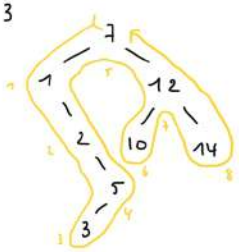


* Tri avec un ABR

↳ insertions successives :

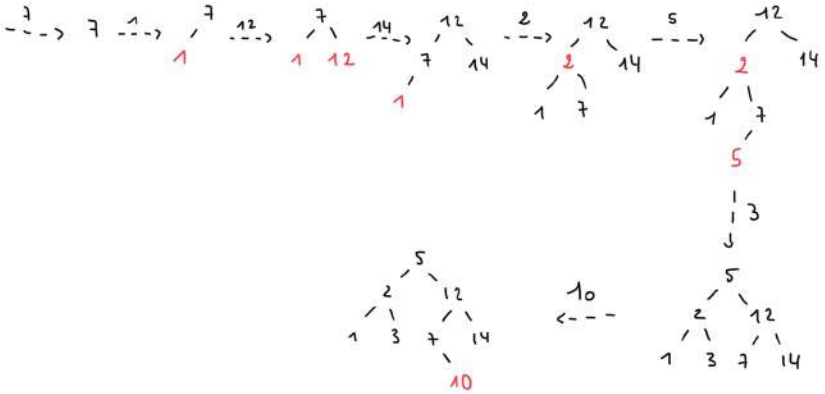


↳ parcours infixe :



* Tri avec un ARN

↳ insertions successives :



↳ parcours infixe :