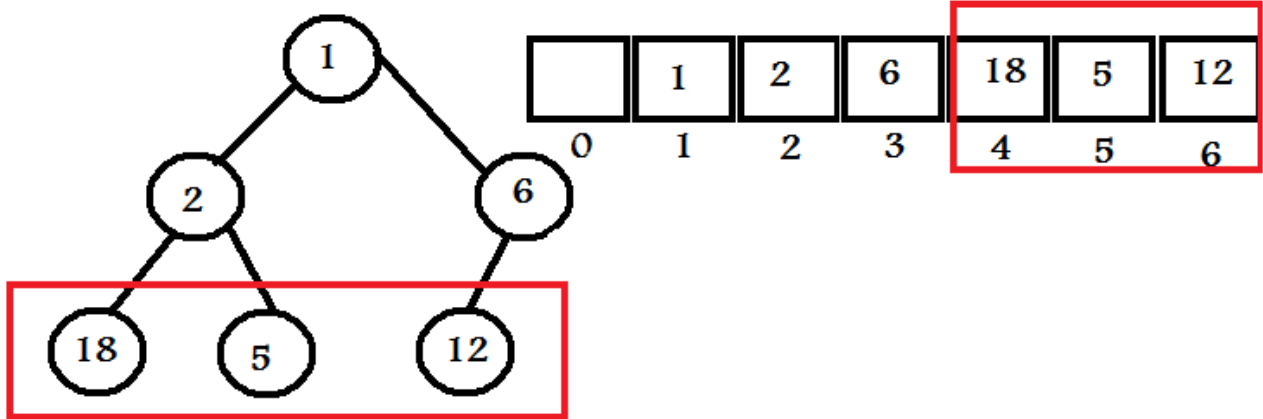
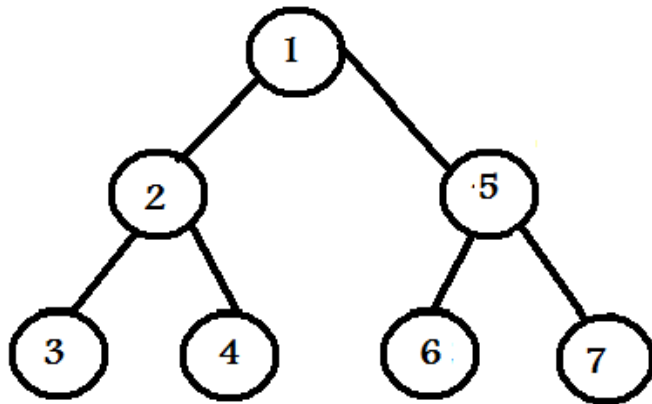


Solutionnaire des travaux pratiques sur les files avec priorités

- Les nœuds où pourrait se retrouver l'entrée avec la plus grande clé sont les nœuds feuilles. Ou bien, si l'on considère une file, ce sont les nœuds aux positions $n/2$ jusqu'à n .
Exemple:

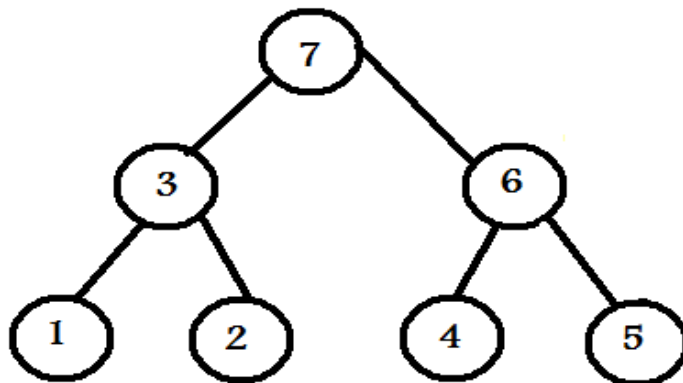


- Parcours préfixe:



Parcours symétrique: impossible

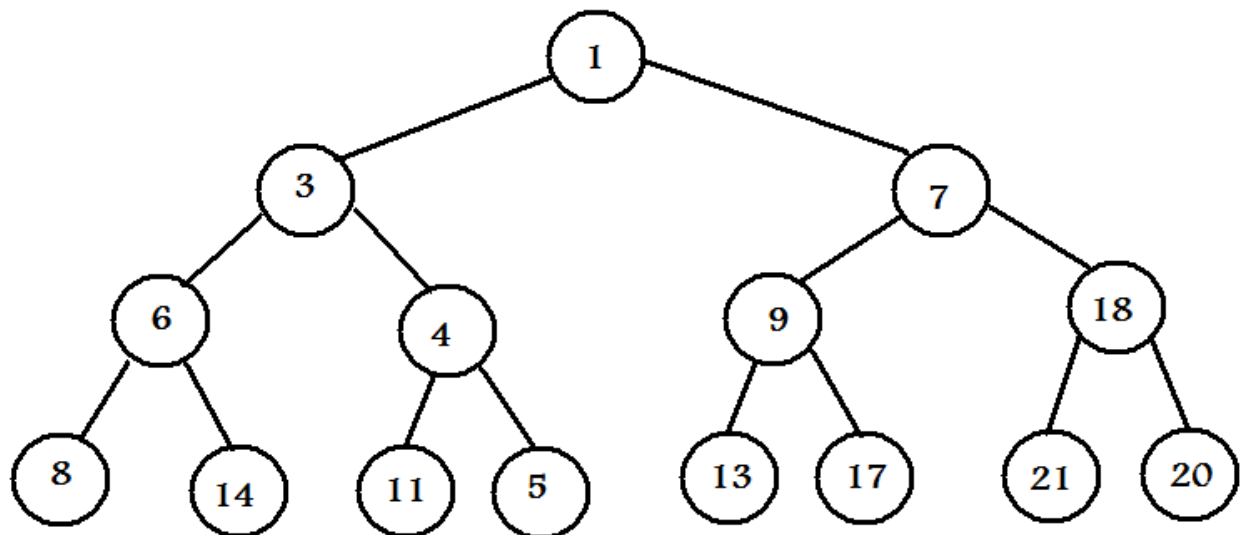
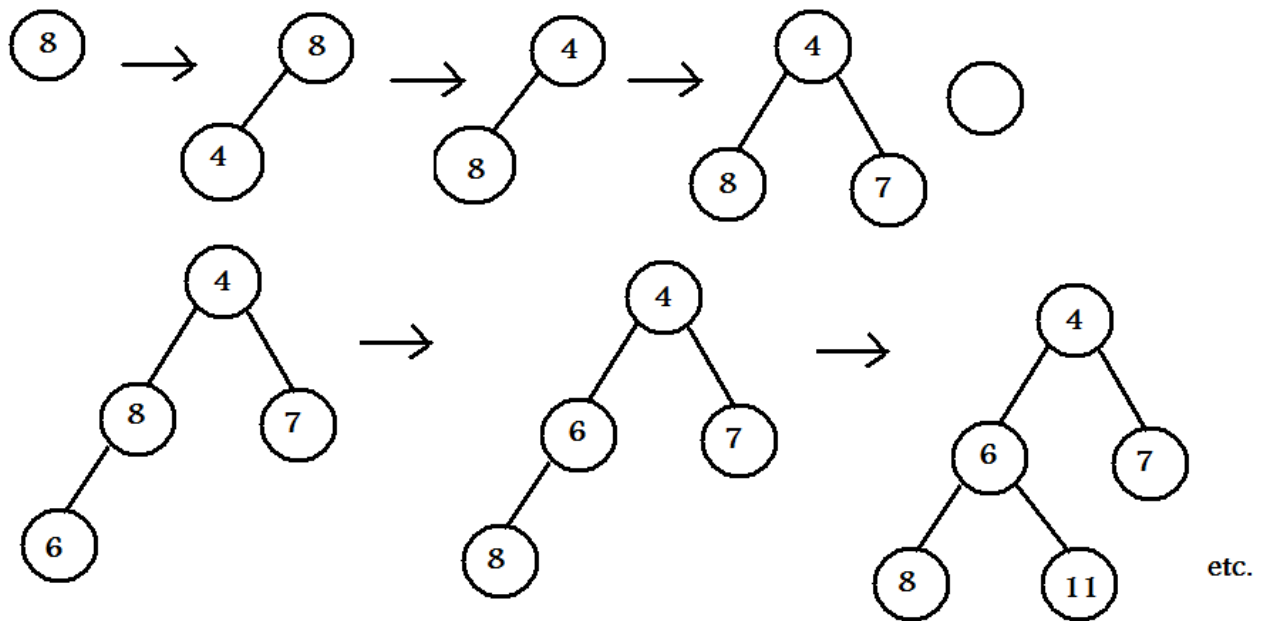
Parcours suffixe:



3. Avec la séquence de clés suivantes: $S = [8, 4, 7, 6, 11, 13, 20, 1, 14, 3, 5, 9, 17, 21, 18]$
 a) Monceau avec l'algorithme de construction du haut vers le bas (top-down)

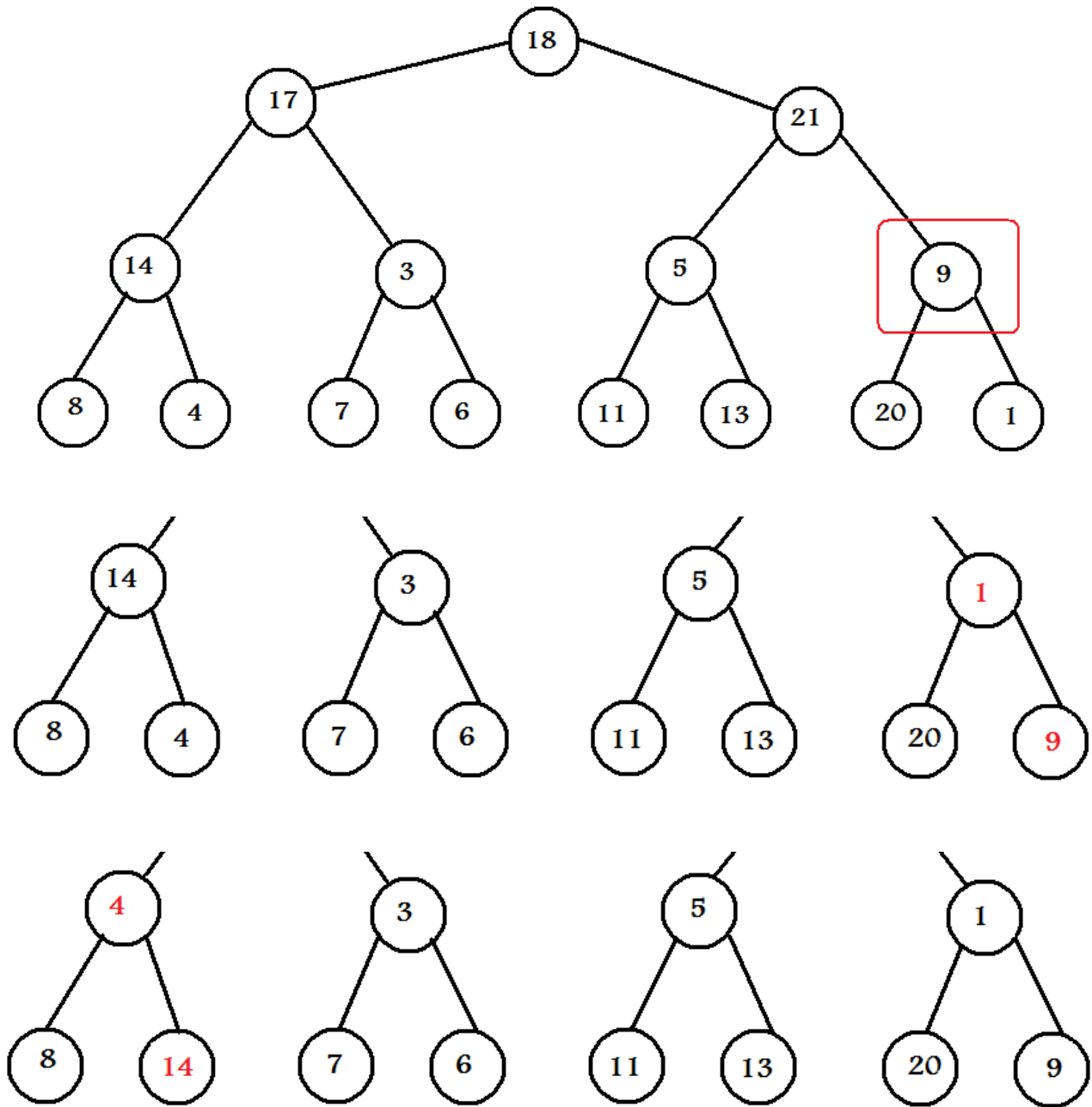
À partir d'un monceau vide, insérer successivement les éléments. À chaque insertion, utiliser l'algorithme d'insertion suivant:

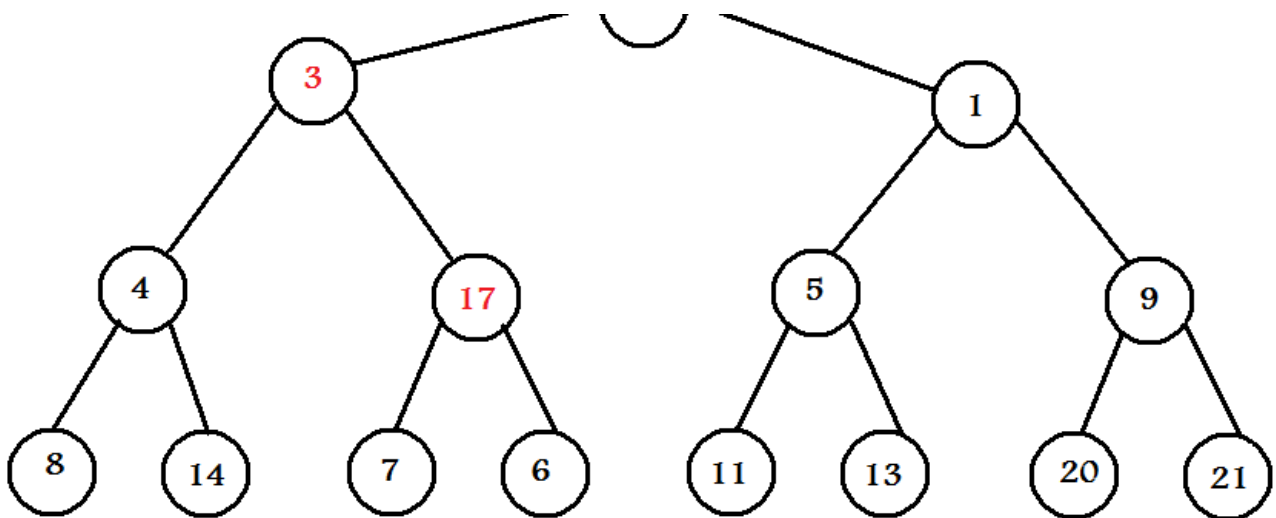
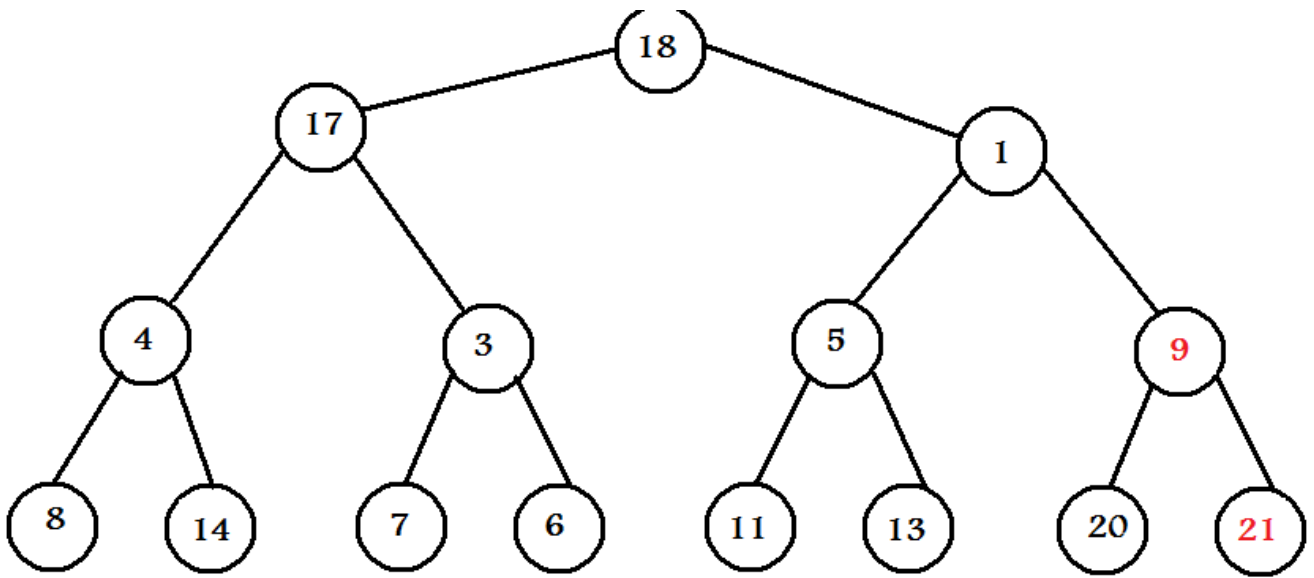
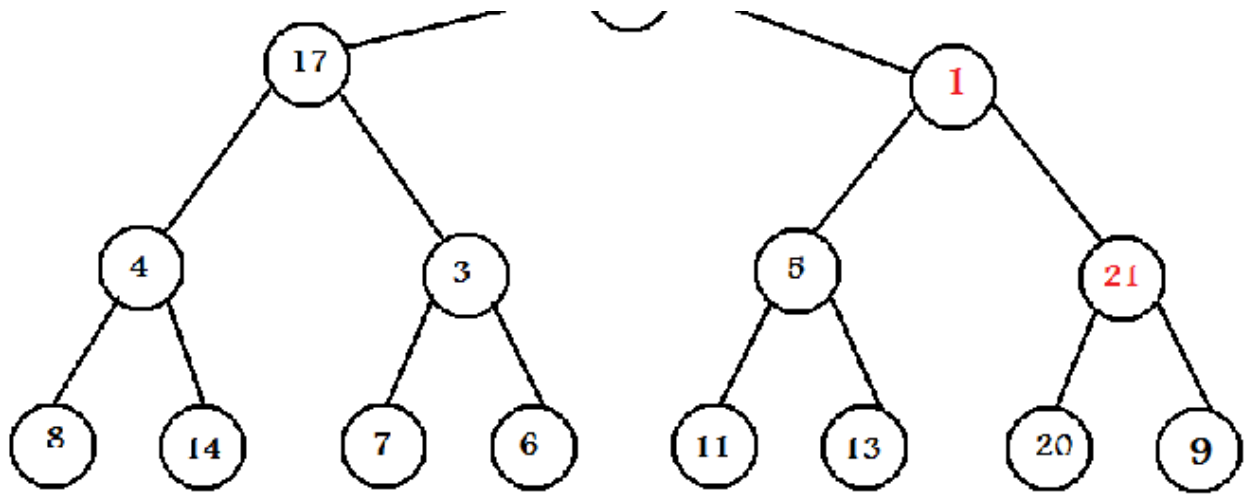
- Insérer l'élément à la dernière position du monceau
- Comparer avec son parent, et si les conditions de priorités ne sont pas respectées, échangez les deux nœuds.
- Continuer la comparaison du nouvel élément avec les nœuds au dessus dans l'arbre, jusqu'à ce que les conditions de priorités soient satisfaites.

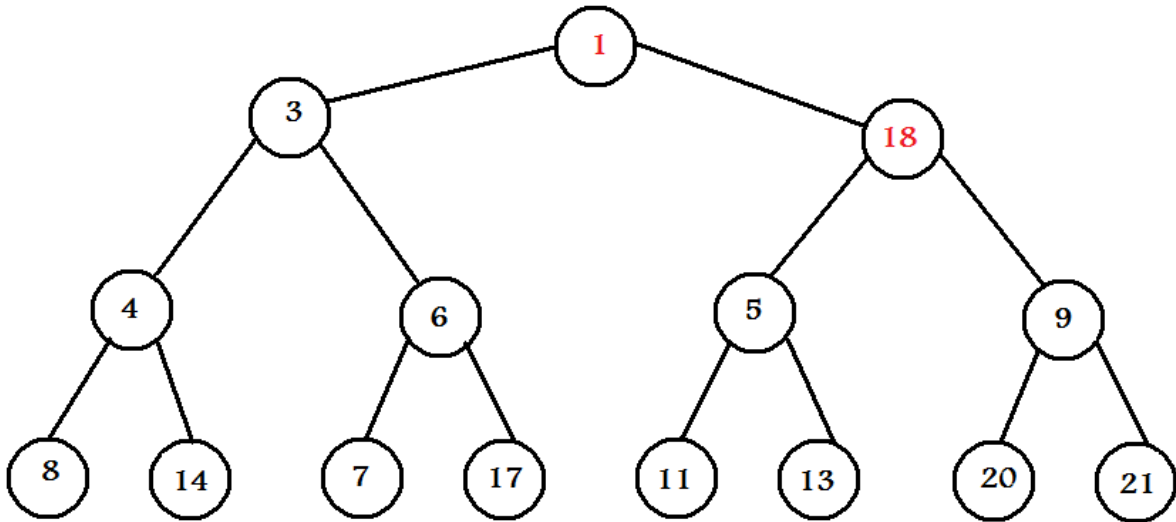
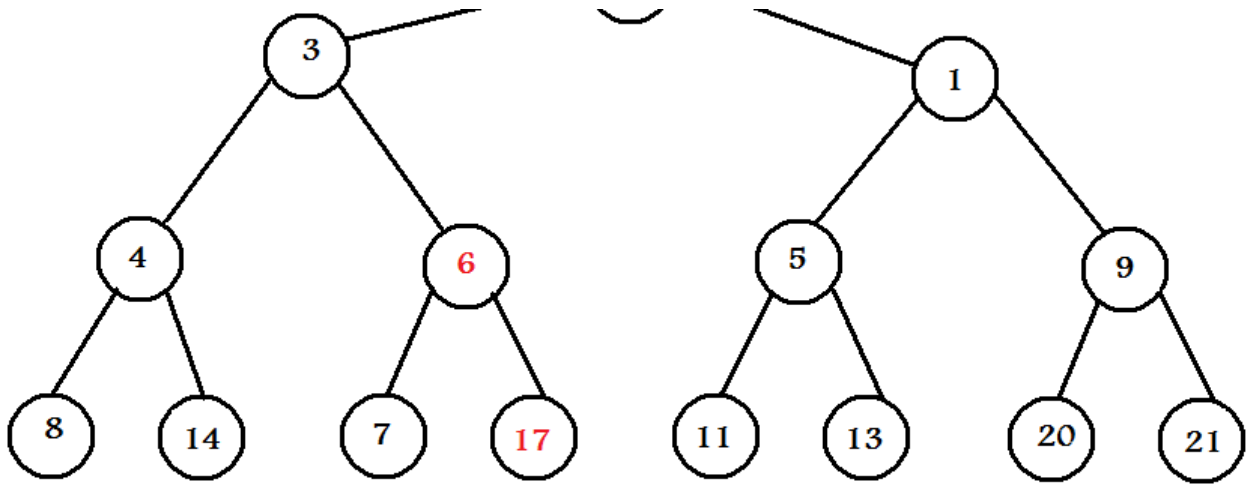


b) Monceau avec l'algorithme de construction du bas vers le haut (bottom-up)

1. Remplir l'arbre avec les éléments dans l'ordre d'arrivée, en suivant les niveaux (remplir les niveaux en commençant par le bas);
2. En commençant par le dernier nœud parental (celui le plus à droite), arranger le monceau accroché a ce nœud, s'il ne satisfait pas la condition de priorité.
 - L'échanger avec son plus petit enfant;
 - Arranger le sous-arbre accroché à ce nœud (maintenant l'enfant).

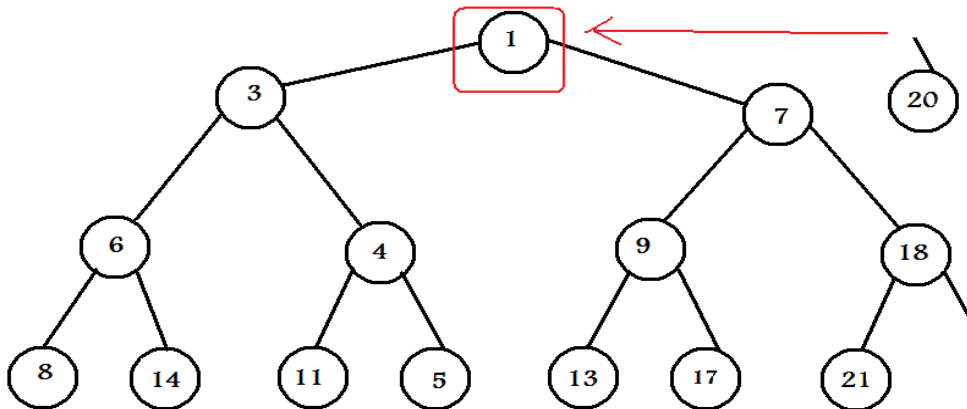


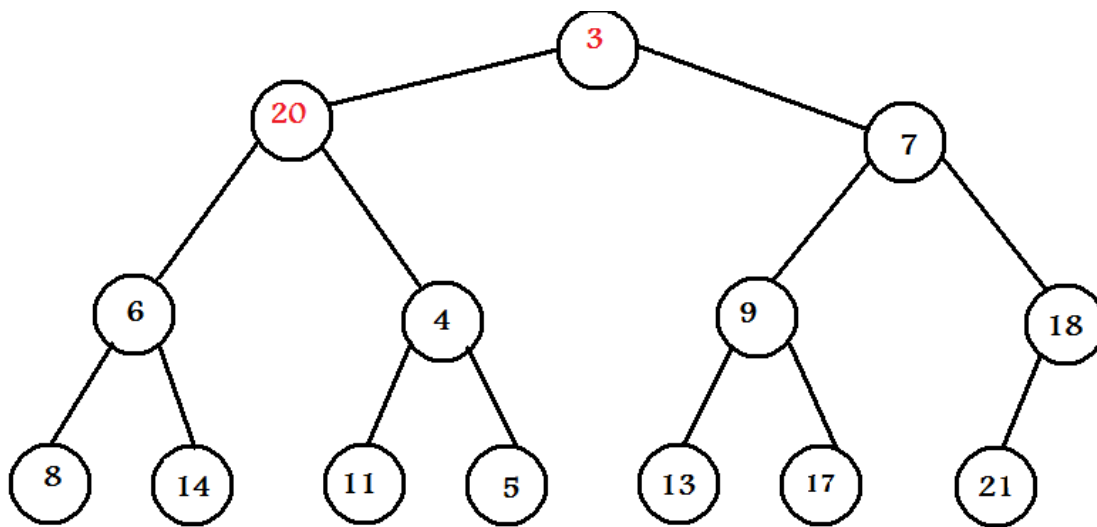
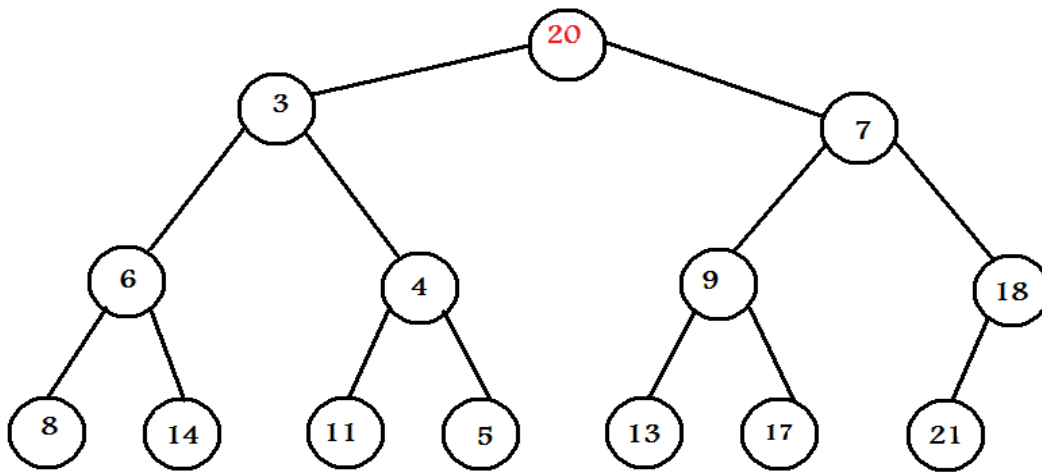


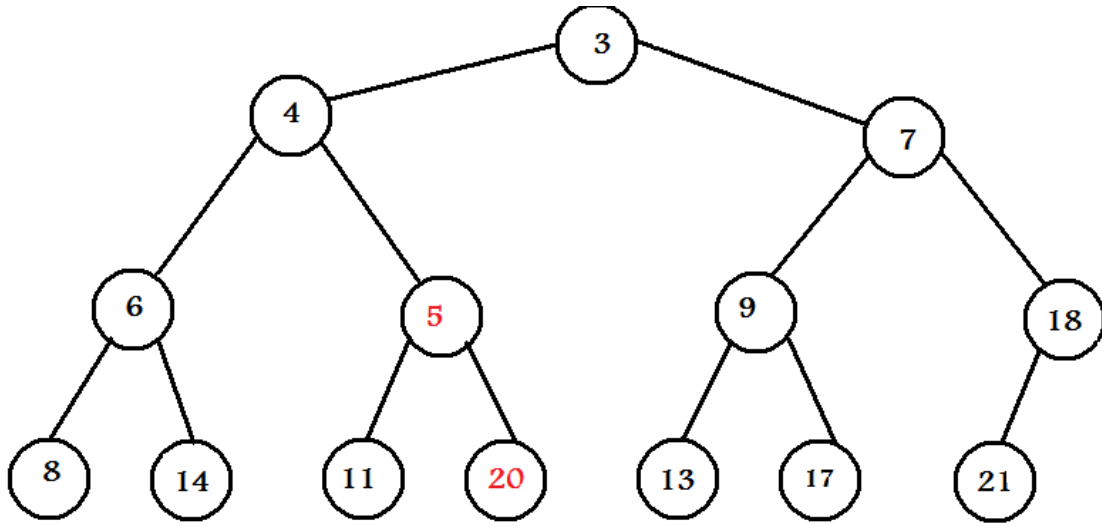
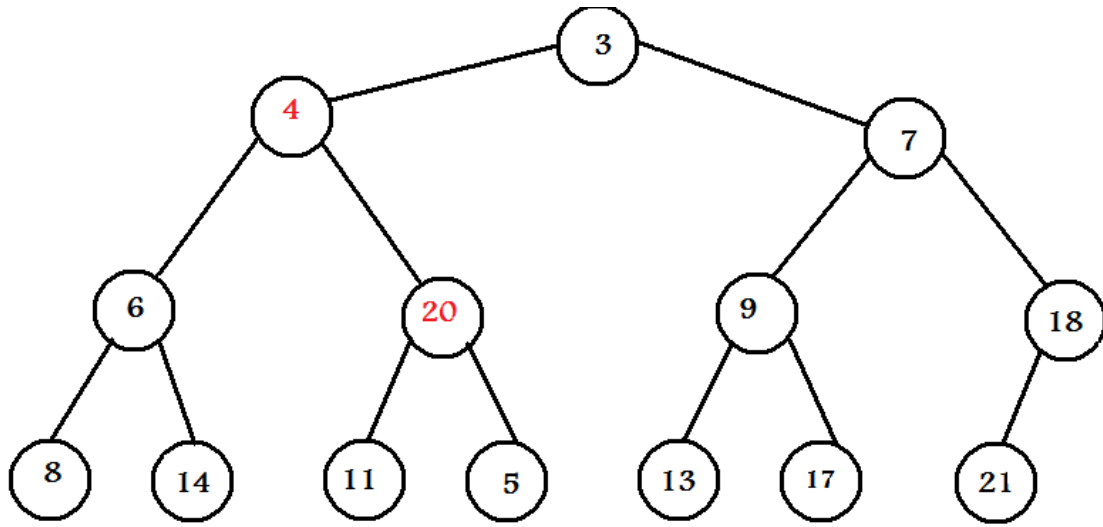


c) Non ce n'est pas le même arbre.

d) Nous allons exécuter EnleverMin sur le monceau obtenu en a. Nous enlevons la racine et nous la remplaçons par le le dernier nœud, puis nous arrangeons l'arbre.







4. Voici un algorithme qui calcule le k ième plus petit élément d'un ensemble de n entiers distincts en temps $O(n + k \log n)$

1. On construit un monceau avec les n éléments en temps $O(n)$
2. On explore les nœuds pour trouver le k ième plus prioritaire. On compare le fils gauche et le fils droit du nœud courant. Le fils avec la plus grande priorité devient le nœud courant et on met l'autre fils en réserve. À chaque fois qu'on change de courant, on diminue k. Au nouveau courant, on vérifie que ses fils ne sont pas moins prioritaires que le nœud en réserve, si oui, la réserve devient le nœud courant. Si $k = 1$, on retourne le courant. Au plus, on parcourt k fois tous les niveaux ($\log n$), soit $k \cdot \log(n)$.

RetournerKieme* (courant, réserve, k) avec courant = racine, réserve = null et k un entier

```
Si k = 1 faire
    Retourner courant
Fin si
Si courant.FilsG > courant.FilsD
    Si réserve = null
        réserve = courant.FilsG
    Fin si
    Si courant.FilsD > réserve
        RetournerKieme(réserve, courant.FilsD, k-1)
    Sinon
        RetournerKieme(courant.FilsD, réserve, k-1)
    Fin si
Sinon
    Si réserve = null
        réserve = courant.FilsD
    Fin si
    Si courant.FilsG > réserve
        RetournerKieme(réserve, courant.FilsG, k-1)
    Sinon
        RetournerKieme(courant.FilsG, réserve, k-1)
    Fin si
Fin si
```

Évidemment, on doit s'assurer que chaque nœud est différent de NULL, sinon il est automatiquement moins prioritaire que l'autre nœud auquel on le compare.

On peut aussi faire une version avec un tableau au lieu d'un arbre binaire. Sachant que le monceau est implémenté avec une file prioritaire, le nœud courant pourrait être représenté par $File[i]$, son fils gauche par $File[2 \cdot i]$, son fils droit par $File[2 \cdot i + 1]$. On peut alors refaire le même algorithme. On n'oublie pas vérifier si la position i ou celle réservée est plus petite ou égale au nombre total d'éléments, soit n.

• Pour ceux qui étaient à la séance de TP, j'ai trouvé un moyen d'éviter de répéter « si $k=1$, alors retourner courant »

RetournerKieme(File, posReserve, i, k) avec File, le tableau d'éléments avec la première position vide, posReserve = 0, i = 1

Si k = 1 faire

Retourner courant

Fin si

Si File[2*i] > File[2*i+1]

Si posReserve = 0

posReserve = 2*i

Fin si

Si File[2*i+1] > File[posReserve]

RetournerKieme(posReserve, 2*i+1, k-1)

Sinon

RetournerKieme(2*i+1, posReserve, k-1)

Fin si

Sinon

Si posReserve = 0

posReserve = 2*i+1

Fin si

Si File[2*i] > File[posReserve]

RetournerKieme(réserve, 2*i, k-1)

Sinon

RetournerKieme(2*i, réserve, k-1)

Fin si

Fin si