Algorithmes de tri

stage IREM - Nov./Déc. 2010

Plan

Plan

Le tri

Problème : étant donné un tableau d'entiers T, trier T dans l'ordre croissant.

- Problème connu
- Grande richesse conceptuelle :
  - ⋆ Des algorithmes basés sur des idées et des structures de données très différentes...
  - ⋆ Des complexités différentes.
  - ⋆ Des algorithmes optimaux.

Plan

Plan

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position
- Trouver le 3^e plus petit élément et le mettre à la 3^e place,

Le tri par **sélection**

- Trouver le plus petit élément et le mettre au début de la liste
- Trouver le 2^e plus petit et le mettre en seconde position
- Trouver le 3^e plus petit élément et le mettre à la 3^e place,
- ...

Le tri par **sélection**

Tri par sélection

Données : Un tableau de n entiers T

Résultat : Le tableau T trié

pour chaque *i allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$

retourner T

`Indice-Min`(T, i, n) : retourne l'indice du plus petit élément de $\{T[i], T[i+1], \ldots, T[n]\}$.

Le tri par **sélection**

Tri par sélection

Données : Un tableau de n entiers T

Résultat : Le tableau T trié

pour chaque i *allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$

retourner T

`Indice-Min`(T, i, n) : retourne l'indice du plus petit élément de $\{T[i], T[i+1], \ldots, T[n]\}$.

Propriété : Après la i^e étape ($i = 1, \ldots, n-1$), les i premières cases sont occupées par les i plus petits entiers de T

Complexité du tri par sélection

Tri par sélection

Données : Un tableau de n entiers T
Résultat : Le tableau T trié

pour chaque i *allant de* 1 *à* $n-1$ **faire**
    $ind \leftarrow$ `Indice-Min`(T, i, n)
    $T[i] \leftrightarrow T[ind]$
retourner T

Dans le pire cas ou en moyenne, la complexité (ici : nombre de comparaisons) du tri par sélection est en $O(n^2)$.

Plan

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...
- **Insérer** le n^e élément à sa place.

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à “sa” place pour que...
- ...
- **Insérer** le n^e élément à sa place.

Le tri par **insertion**

(le tri du joueur de cartes !)

- Ordonner les deux premiers éléments
- **Insérer** le 3^e élément de manière à ce que les 3 premiers éléments soient triés
- **Insérer** le 4^e élément à "sa" place pour que...
- ...
- **Insérer** le n^e élément à sa place.

A la fin de la i^e itération, les i premiers éléments de T sont triés et rangés au début du tableau T'.

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Insérer(T, k)

---

si $k > 1$ **alors**
  si $T[k-1] > T[k]$ **alors**
    $T[k] \leftrightarrow T[k-1]$
    Insérer(T,k-1)

Le tri par insertion

Pour $i = 2 \ldots n$: Insérer(T, i)

Insérer(T, k)

---

si $k > 1$ **alors**
  si $T[k-1] > T[k]$ **alors**
    $T[k] \leftrightarrow T[k-1]$
    Insérer(T,k-1)

Dans le pire cas ou en moyenne, la complexité du tri par sélection est en $O(n^2)$.

Plan

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad\qquad$ $3, 7, 12, 16, 25, 38, 40$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $3, 7, 12, 16, 25, 38, 40$

$3,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38, 40$

Tri **fusion**

Étant donné un tableau (ou une liste) de $T[1, \ldots, n]$:

- si $n = 1$, retourner le tableau T !
- sinon :
  - Trier le sous-tableau $T[1 \ldots \frac{n}{2}]$
  - Trier le sous-tableau $T[\frac{n}{2} + 1 \ldots n]$
  - Fusionner ces deux sous-tableaux...
- Il s'agit d'un algorithme "diviser-pour-régner".
- $O(n \log n)$ opérations (au pire).

Plan

Le tri **rapide**

Un autre tri récursif... plus efficace en pratique.

Étant donné un tableau de $T[1, \ldots, n]$:

- si $n = 1$, retourner le tableau T.
- sinon :
  - Choisir un élément (le "pivot") p dans T
  - Placer les éléments inférieurs à p au début de T
  - Placer p à sa place dans T
  - Placer les éléments supérieurs à p à la fin de T
  - Trier la première partie de T puis la seconde...

(plus de fusion !)

Le tri **rapide**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

Le tri **rapide**

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38

$\underbrace{15,\ 10,\ 19,\ 13,\ 5,\ 3,\ 12,\ 7,\ 16,}_{\text{à trier !}}\ 20,\ \underbrace{35,\ 40,\ 25,\ 38}_{\text{à trier !}}$

Complexité du tri rapide

Dans le pire cas, la complexité du tri rapide est en $O(n^2)$.

Mais en moyenne, elle est en $O(n \cdot \log(n))$.

Plan

Un tri avec des arbres !

A partir d'une liste d'entiers, on va construire un arbre binaire où chaque noeud contiendra un entier de la liste en respectant la propriété suivante :

Tout noeud x doit contenir un entier...

- supérieur (ou égal) aux entiers de son sous-arbre gauche, et
- inférieur strictement aux entiers de son sous-arbre droit.

$\rightarrow$ un "arbre binaire de recherche".

Un tri avec des arbres !

A partir d'une liste d'entiers, on va construire un arbre binaire où chaque noeud contiendra un entier de la liste en respectant la propriété suivante :

Tout noeud x doit contenir un entier...

- supérieur (ou égal) aux entiers de son sous-arbre gauche, et
- inférieur strictement aux entiers de son sous-arbre droit.

$\rightarrow$ un "arbre binaire de recherche".

Comment faire ?

Un tri avec des arbres : exemple...

$20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38$

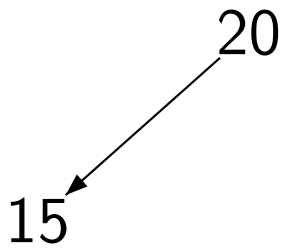
Un tri avec des arbres : exemple...

$20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38$

20

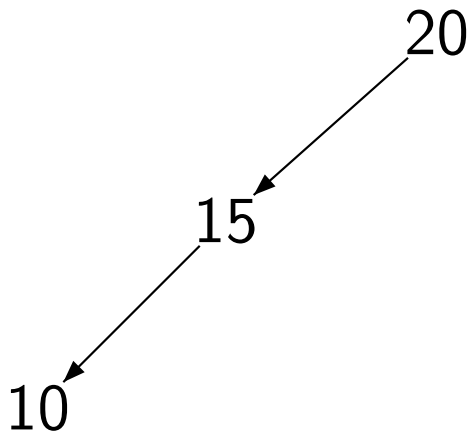
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



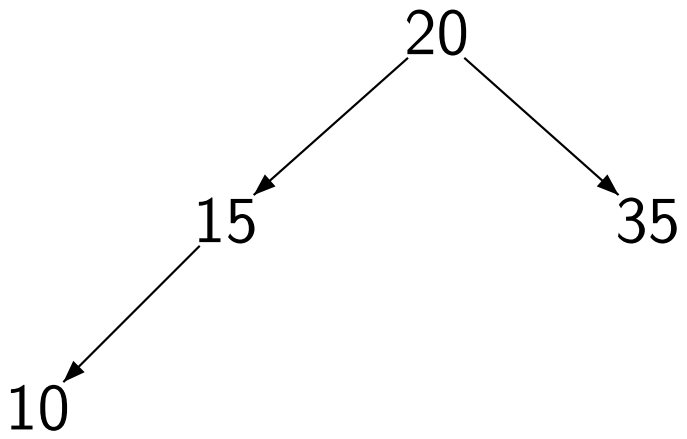
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



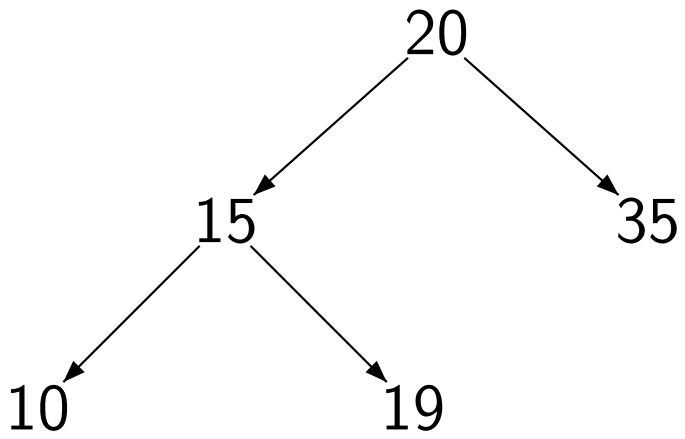
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



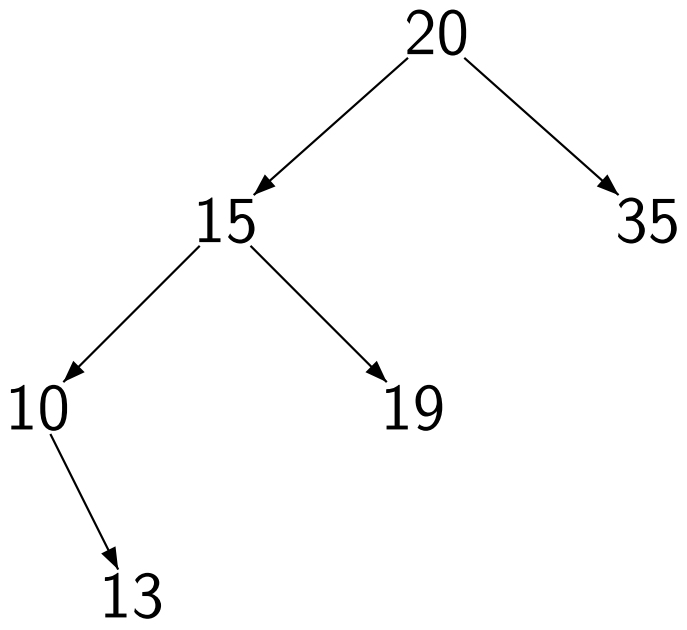
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



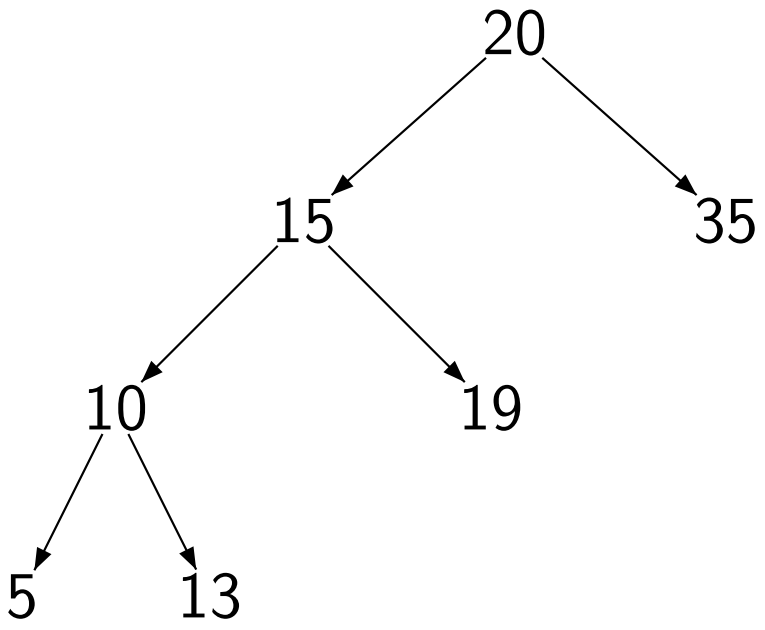
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



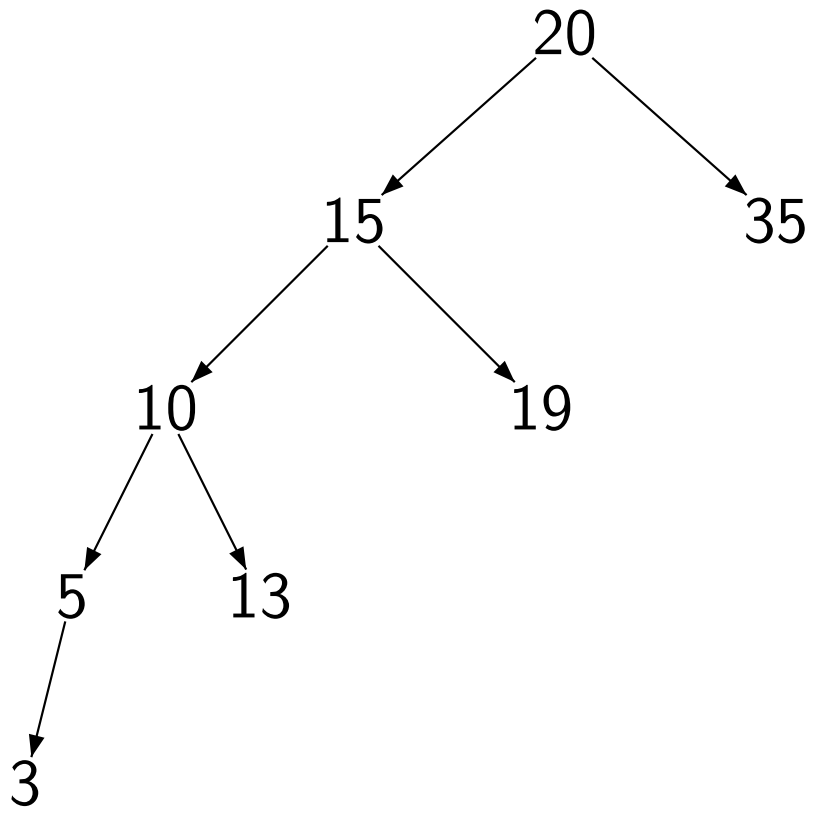
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



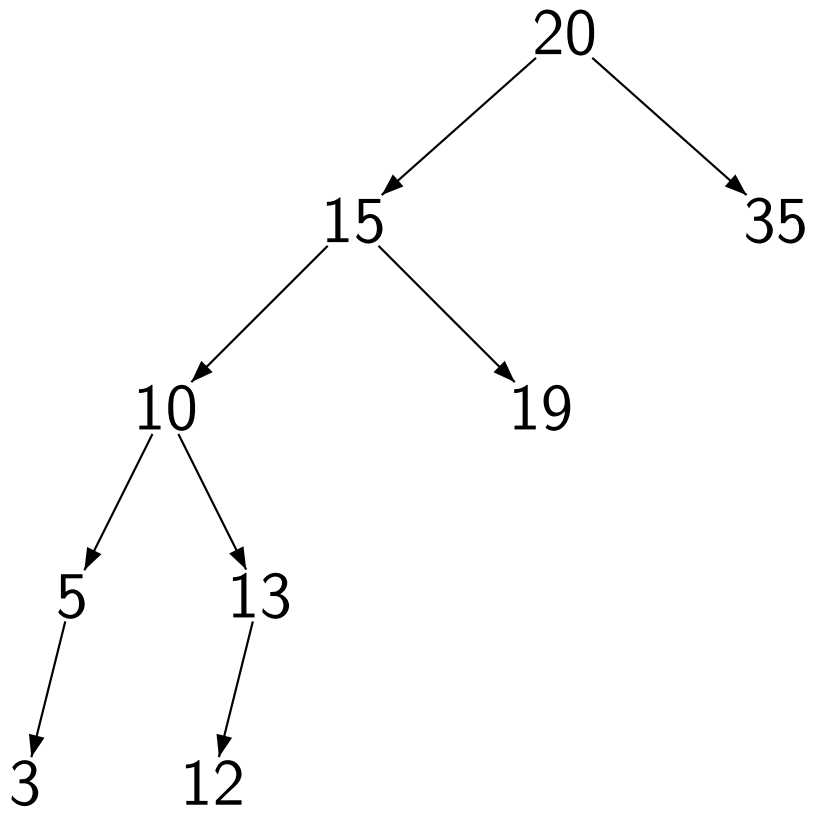
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



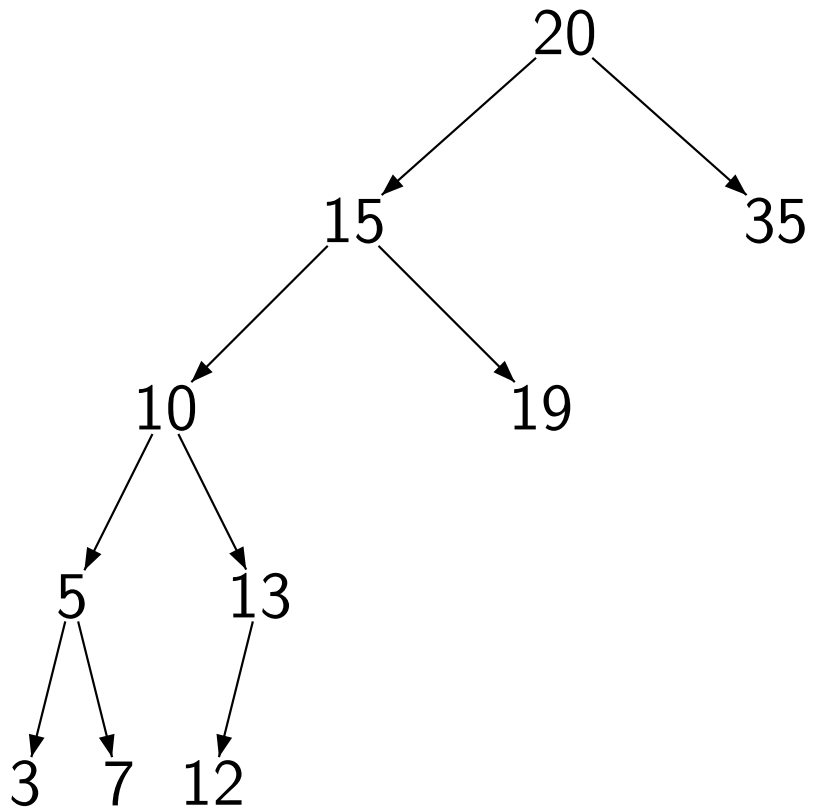
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



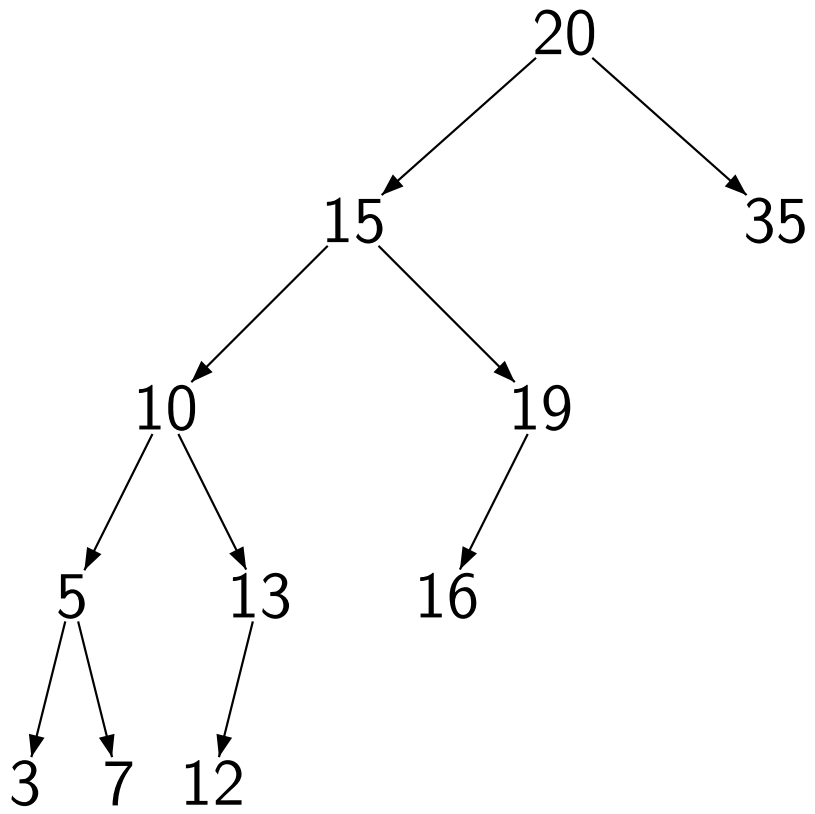
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



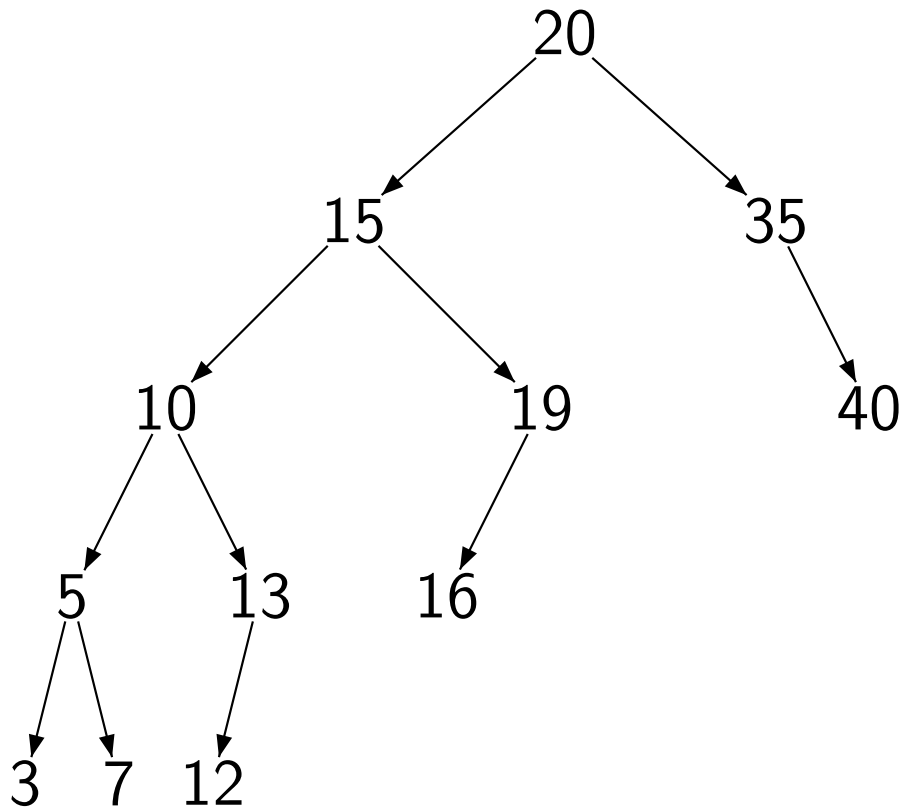
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



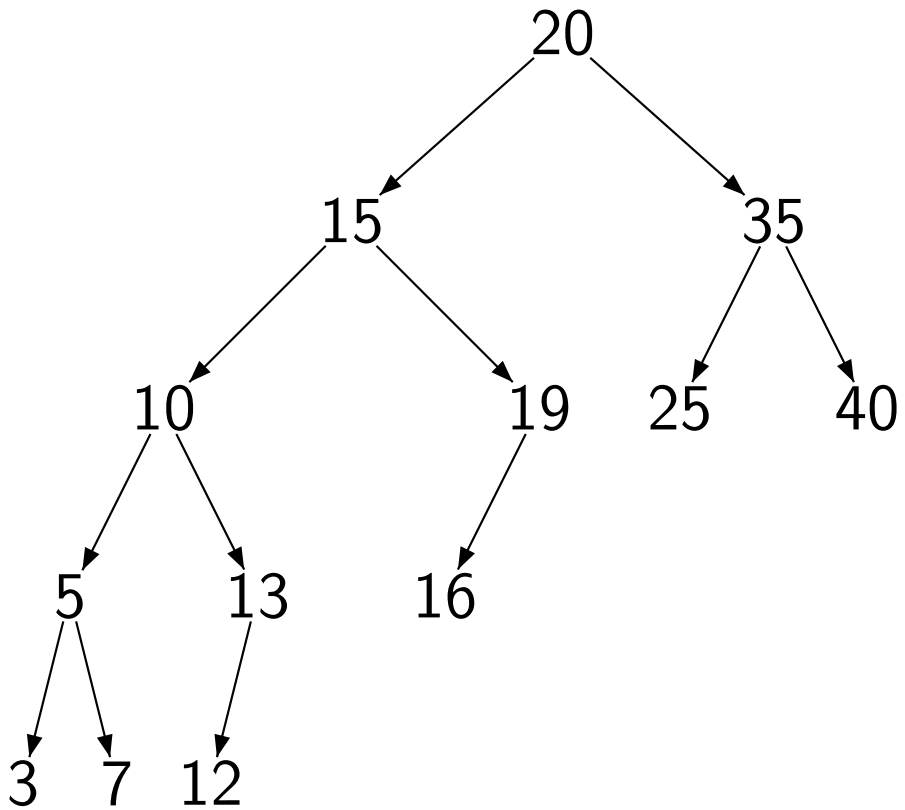
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



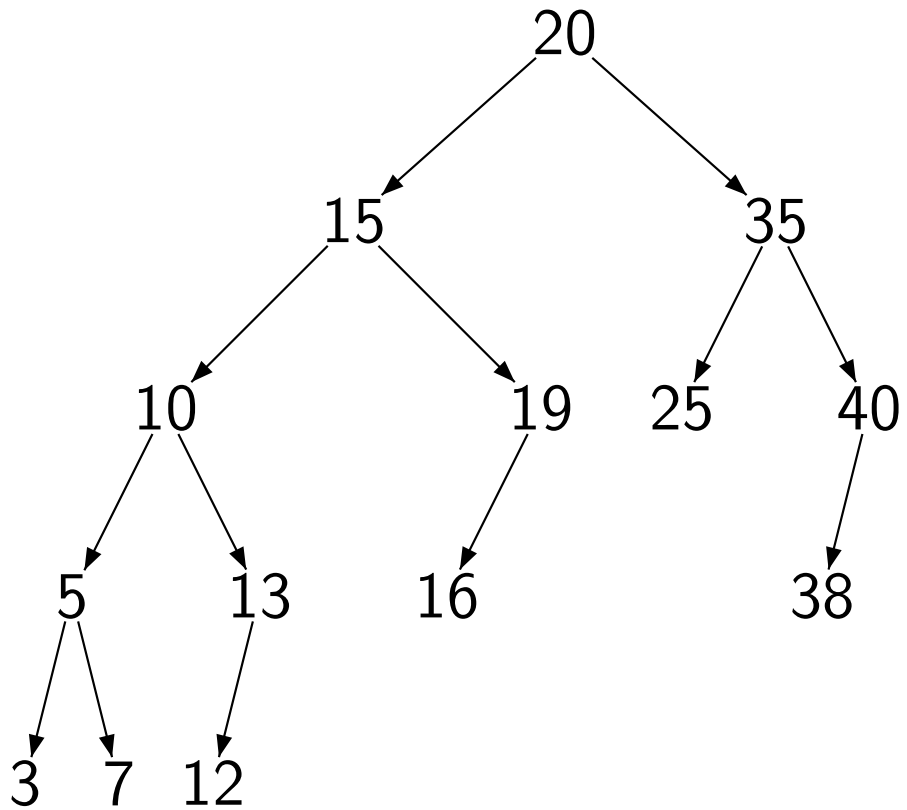
Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



Un tri avec des arbres : exemple...

20, 15, 10, 35, 19, 13, 5, 3, 12, 7, 16, 40, 25, 38



Construire l'arbre

Ajouter (entier x, ABR a)

```
begin
    si EstVide(a) alors
        a = Arbre(x, −, −)
    sinon
        si x ≤ valeur(a) alors
            Ajouter(x,G(a))
        sinon
            Ajouter(x,D(a))
end
```

Et ensuite...

Il reste à parcourir l'arbre construit et à afficher la valeur d'un noeud lorsqu'on le visite pour la deuxième fois (parcours infixe)...

Parcours (noeud a)

```
begin
    si ¬EstVide(a) alors
        [premier passage]
        Parcours(G(a))
        [second passage]
        Parcours(D(a))
        [troisième passage]
end
```

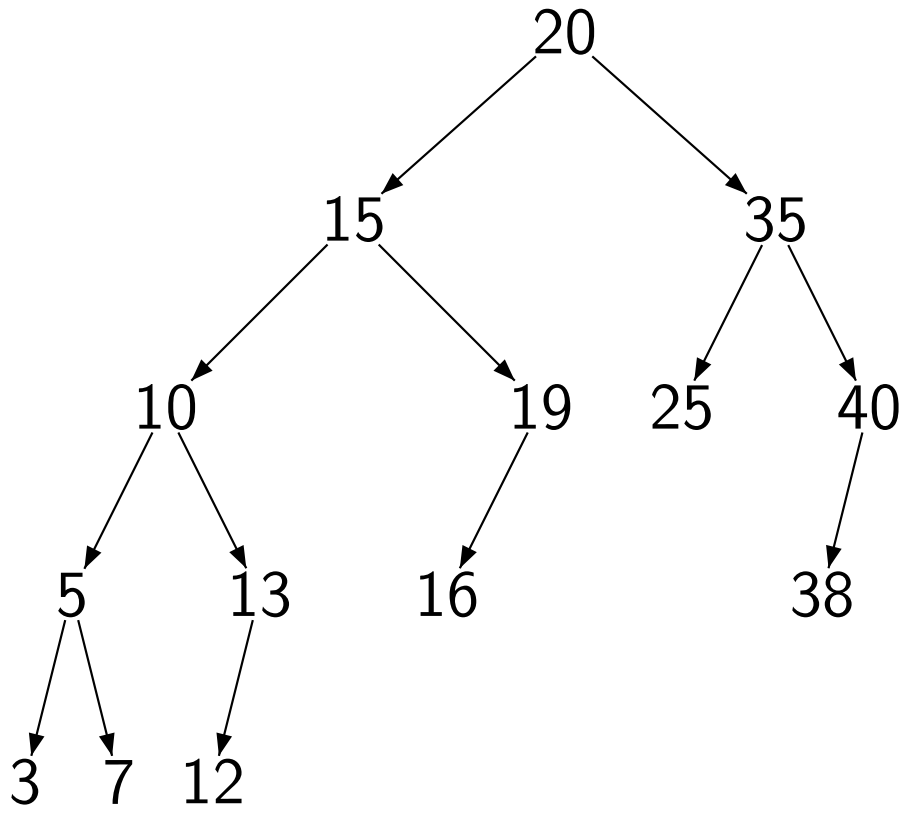
Et ensuite...

Il reste à parcourir l'arbre construit et à afficher la valeur d'un noeud lorsqu'on le visite pour la deuxième fois (parcours infixe)...

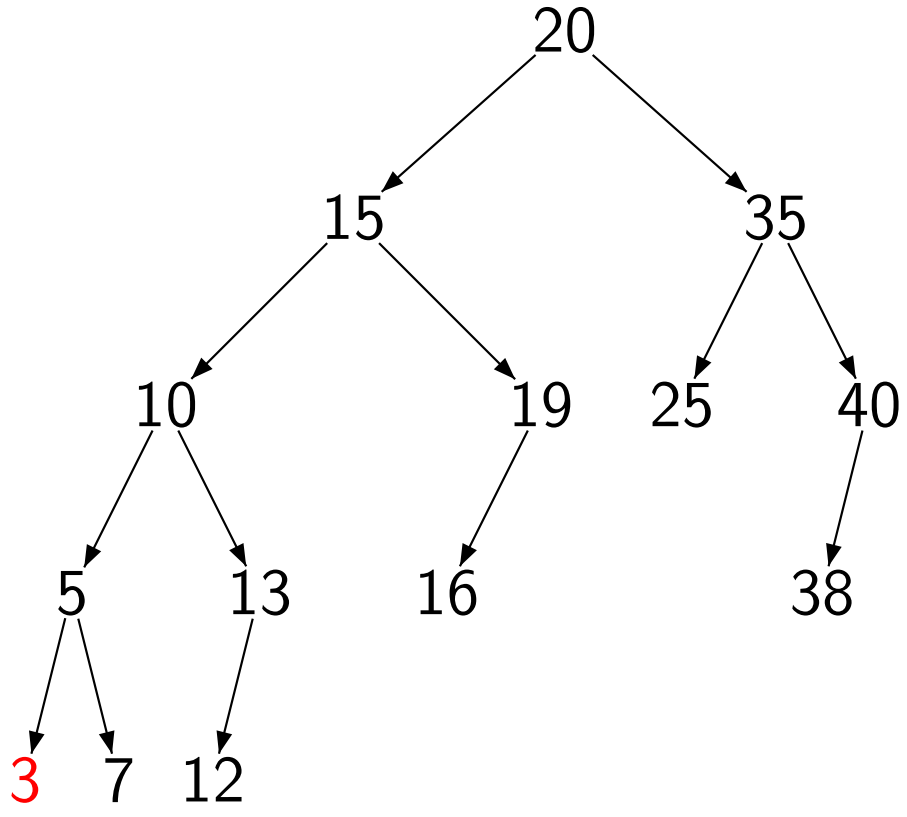
Parcours (noeud a)

```
begin
    si ¬EstVide(a) alors
        [premier passage]
        Parcours(G(a))
        [second passage] Afficher valeur(a)
        Parcours(D(a))
        [troisième passage]
end
```

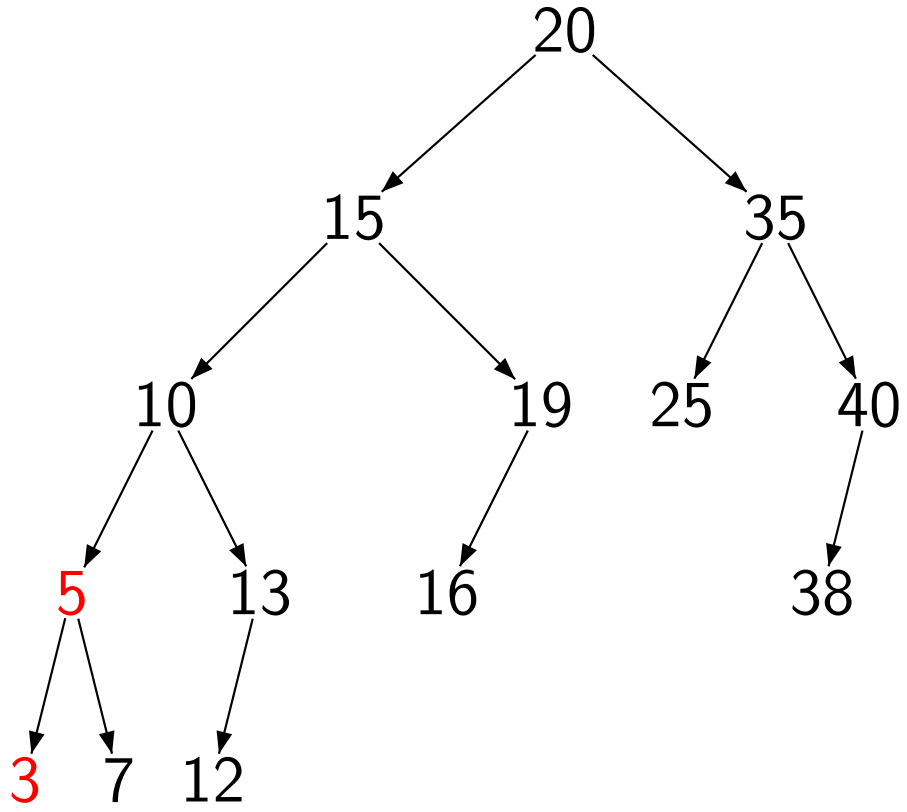
Exemple...



Exemple. . .

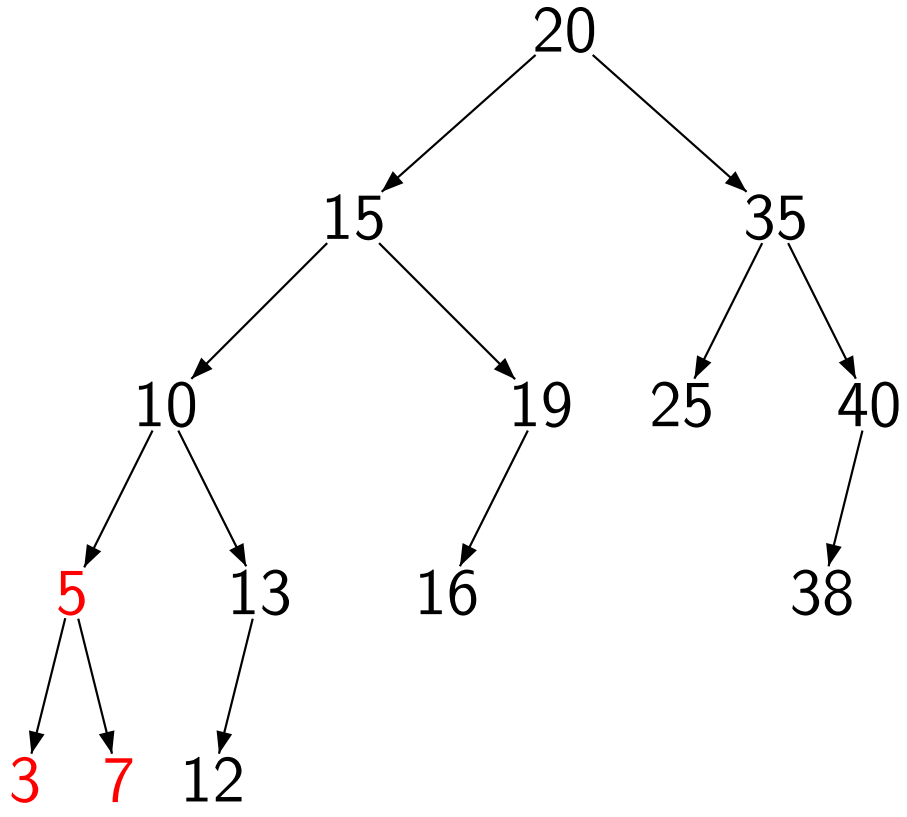


Exemple. . .



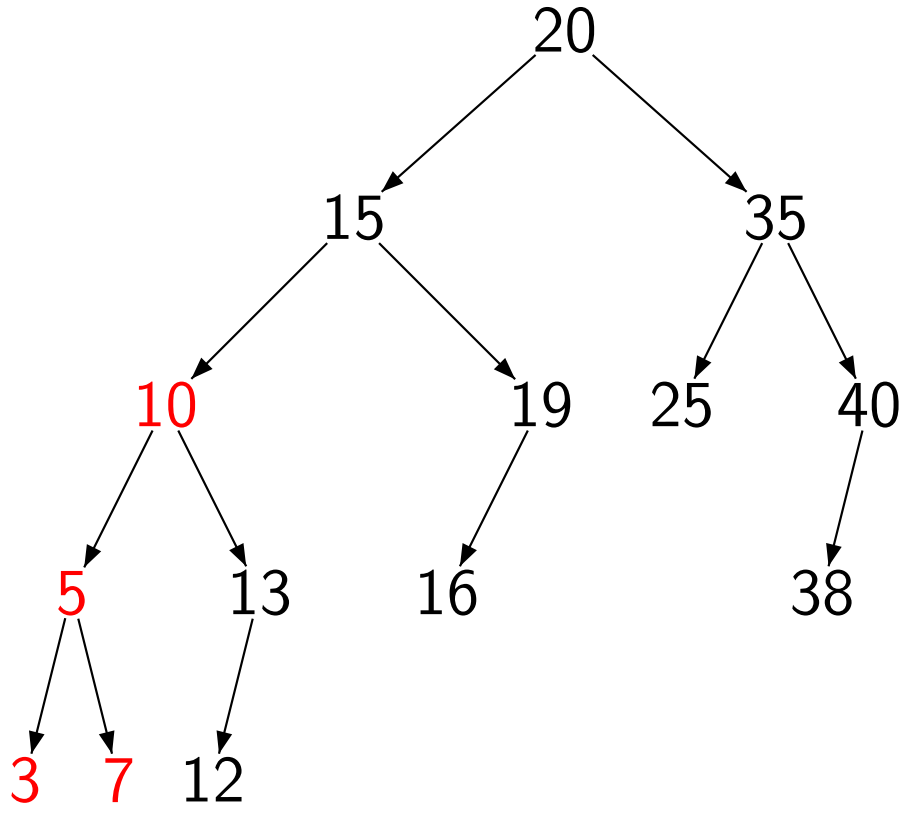
3 5

Exemple. . .



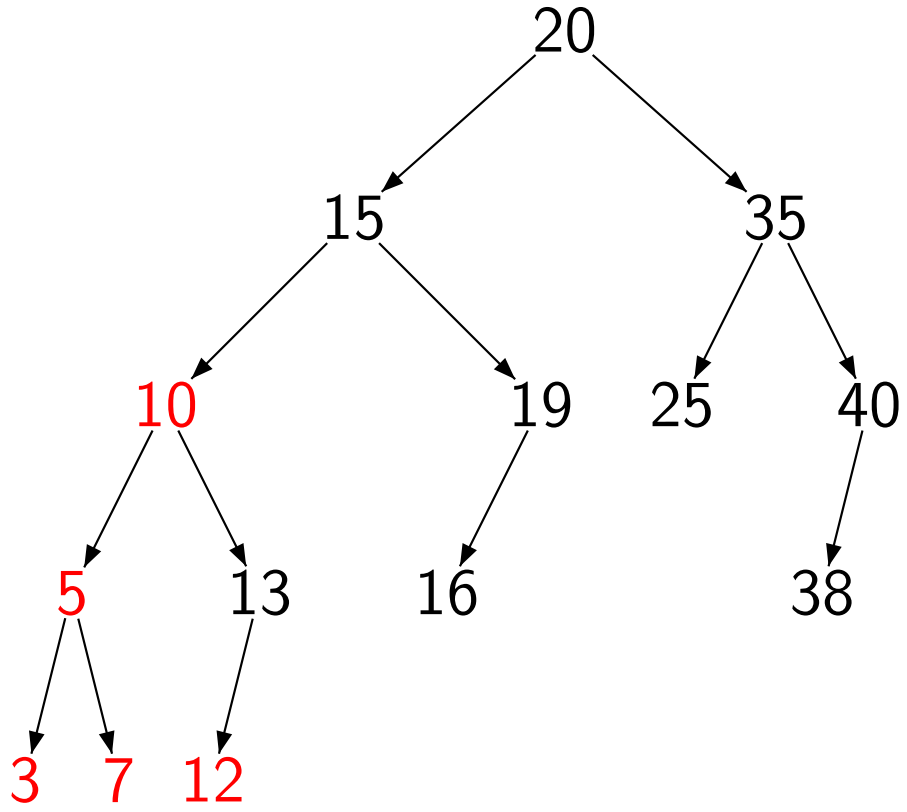
3 5 7

Exemple. . .



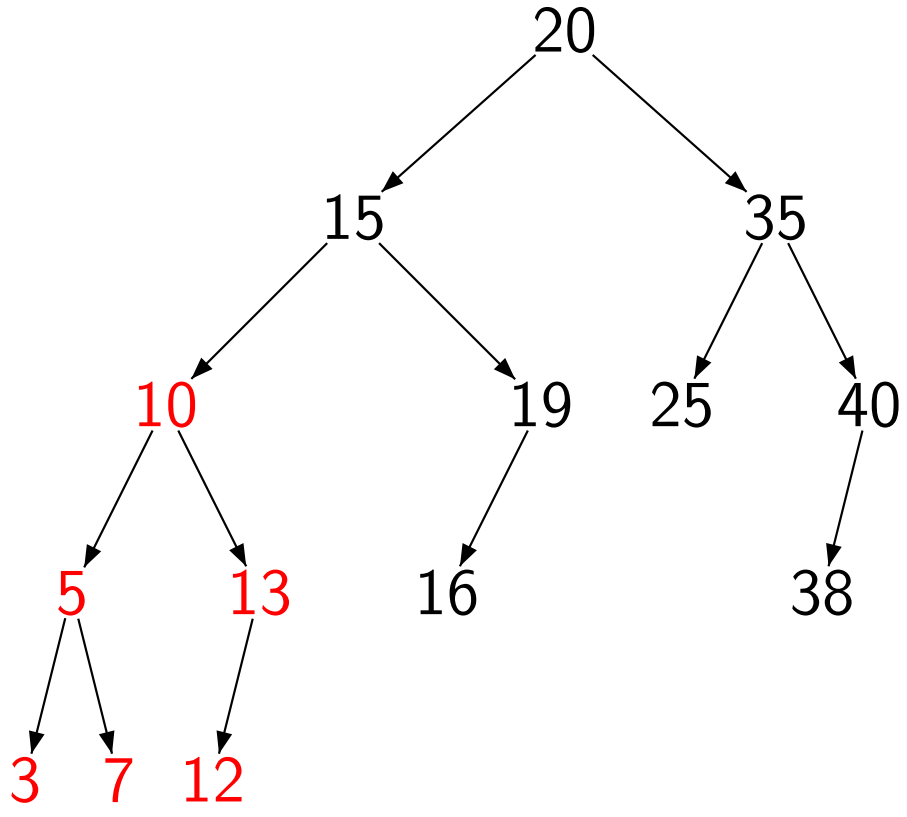
3 5 7 10

Exemple. . .



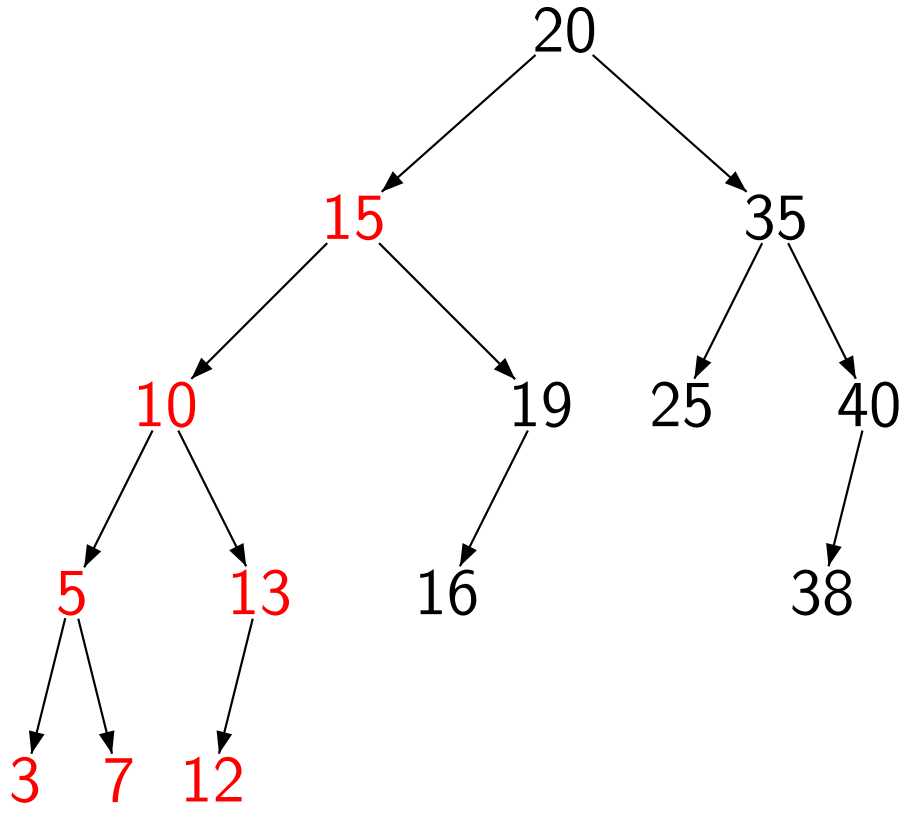
3 5 7 10 12

Exemple...



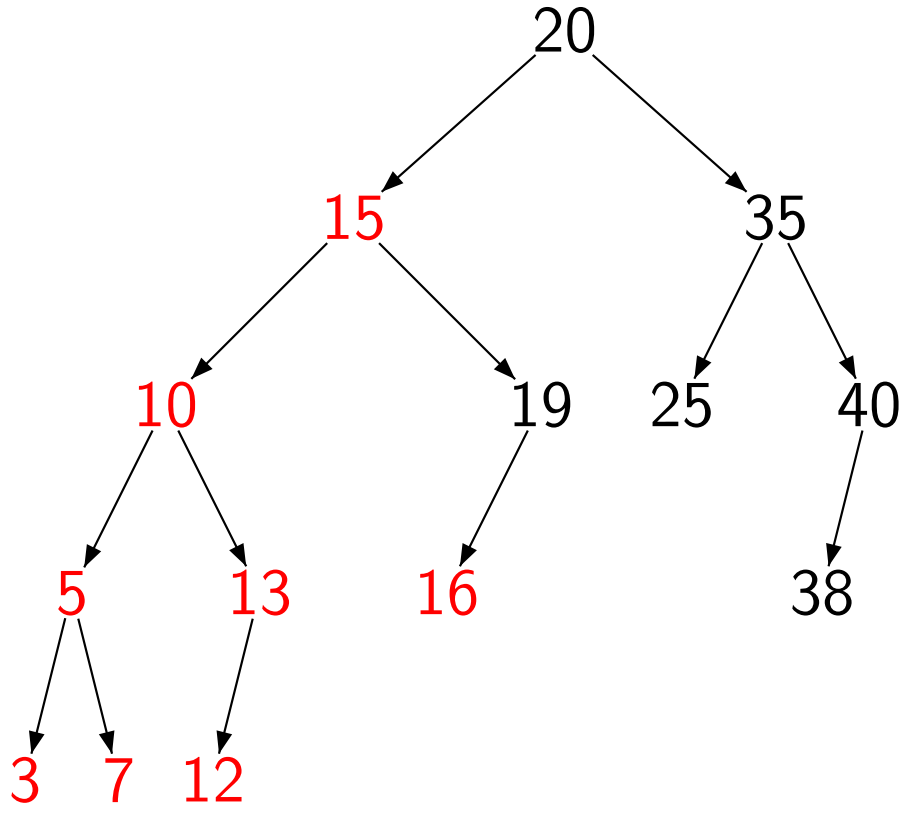
3 5 7 10 12 13

Exemple...



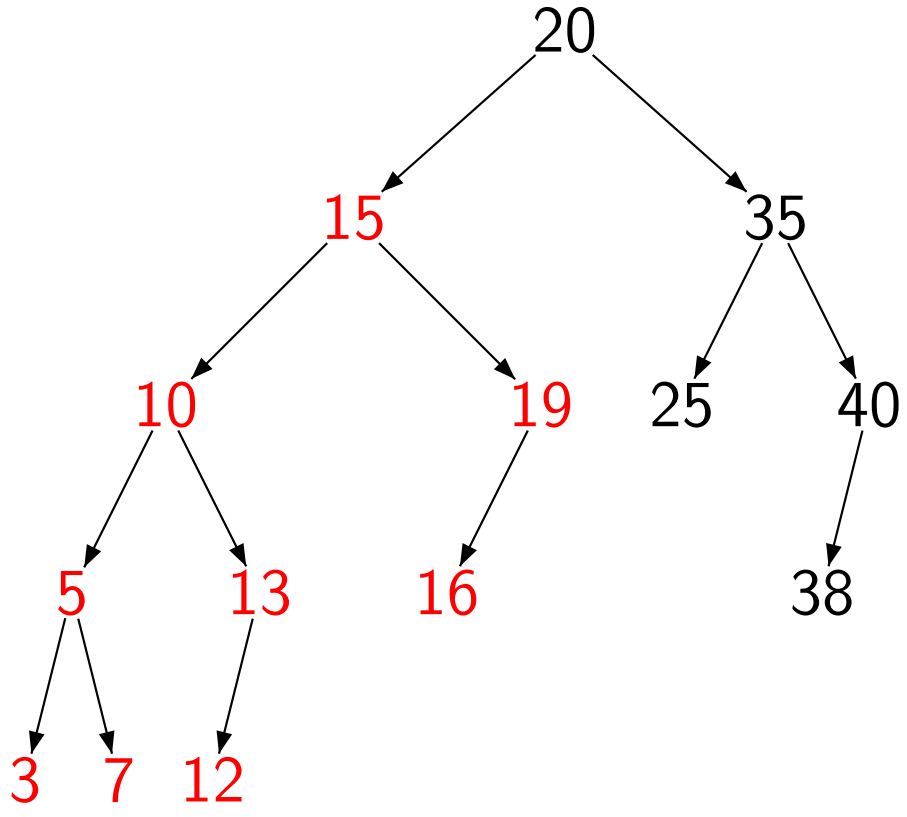
3 5 7 10 12 13 15

Exemple...



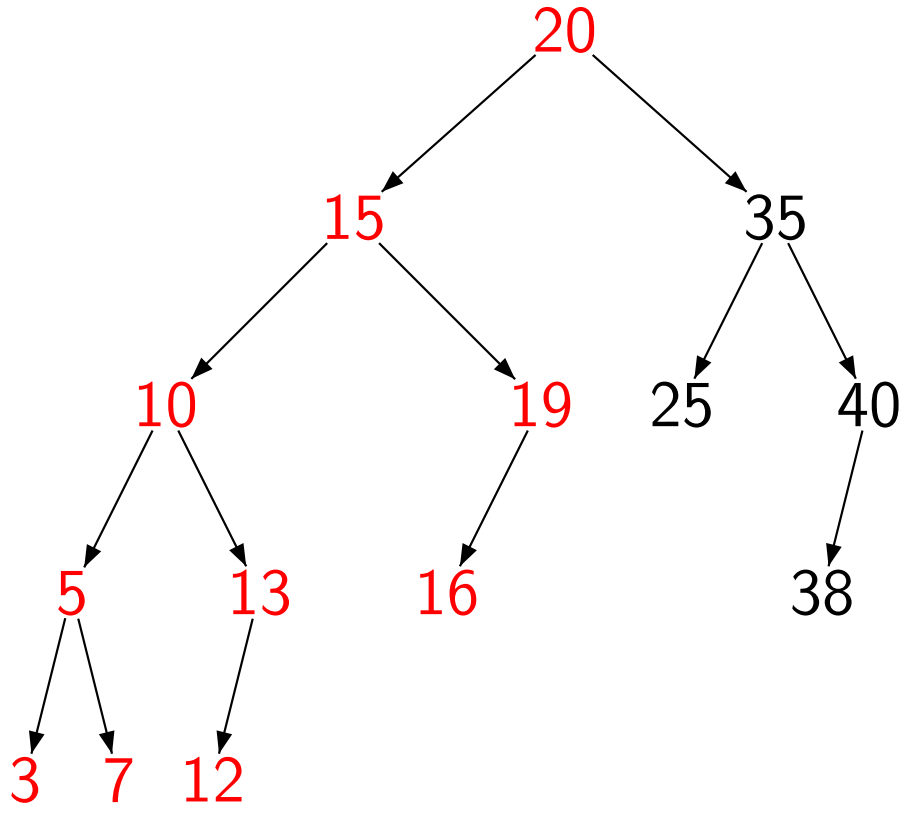
3 5 7 10 12 13 15 16

Exemple...



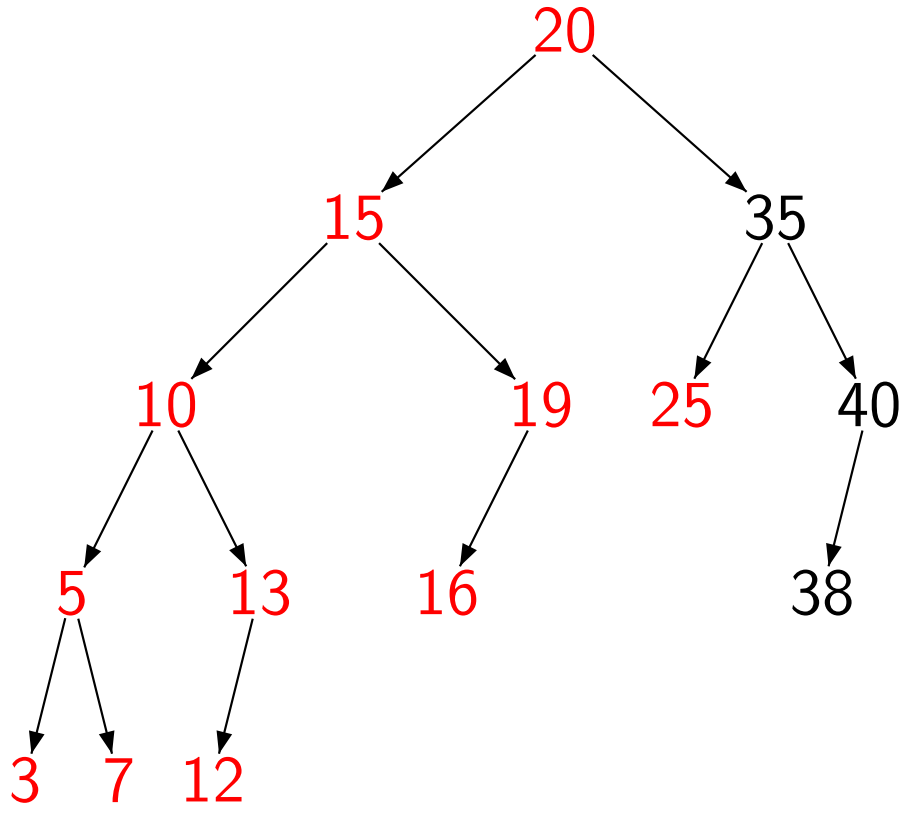
3 5 7 10 12 13 15 16 19

Exemple...



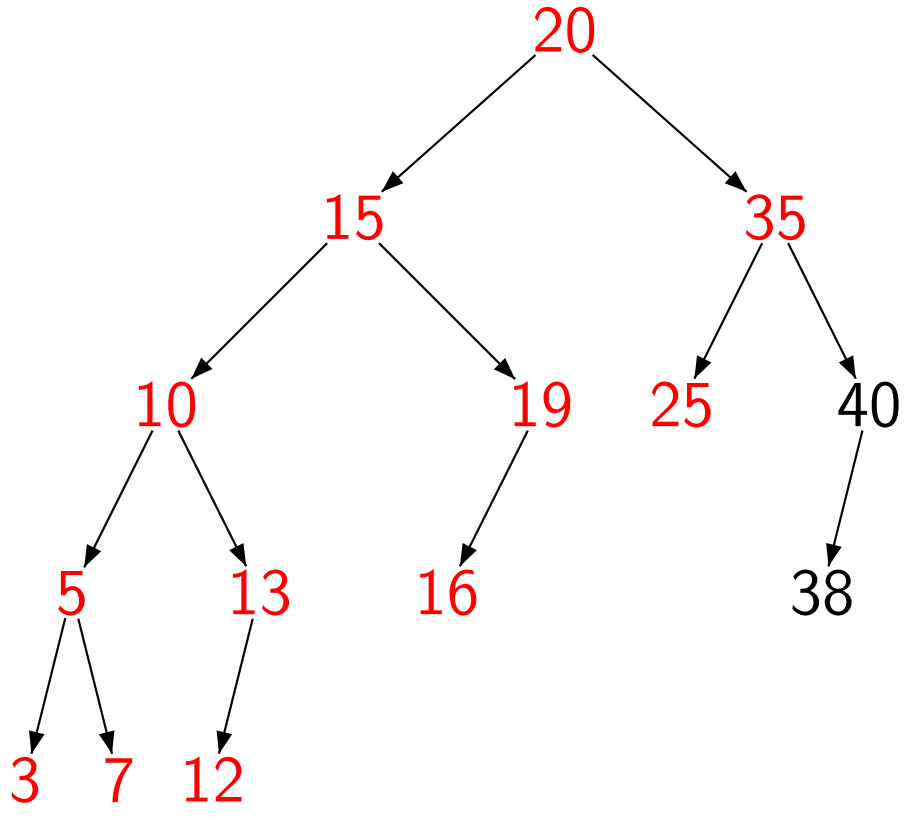
3 5 7 10 12 13 15 16 19 20

Exemple...



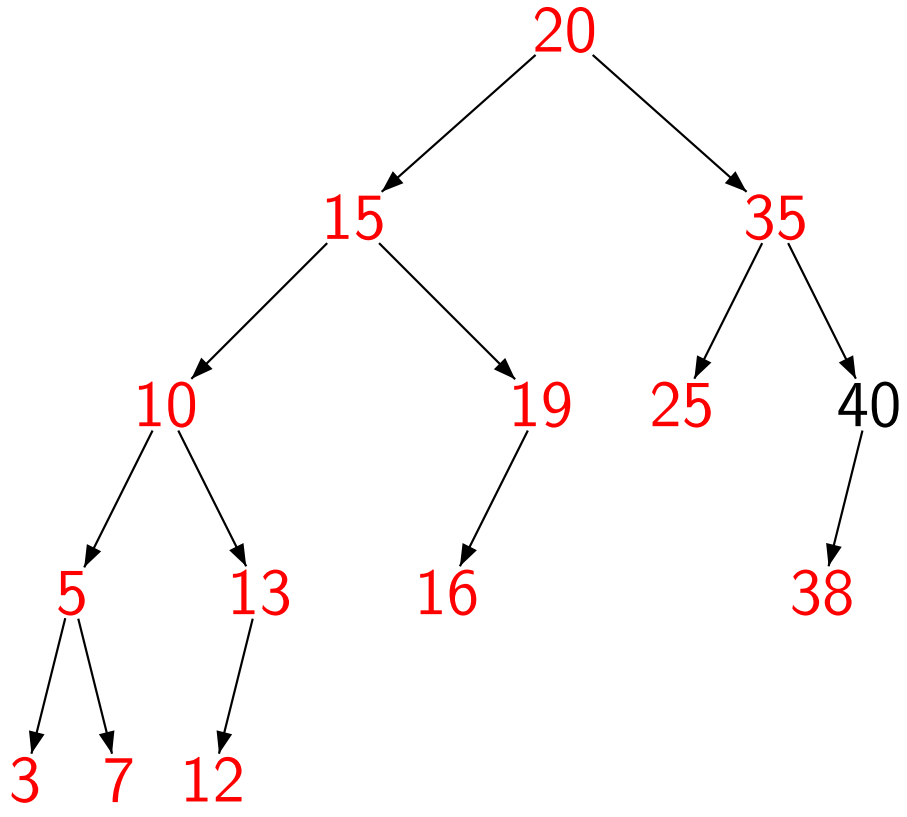
3 5 7 10 12 13 15 16 19 20 25

Exemple...



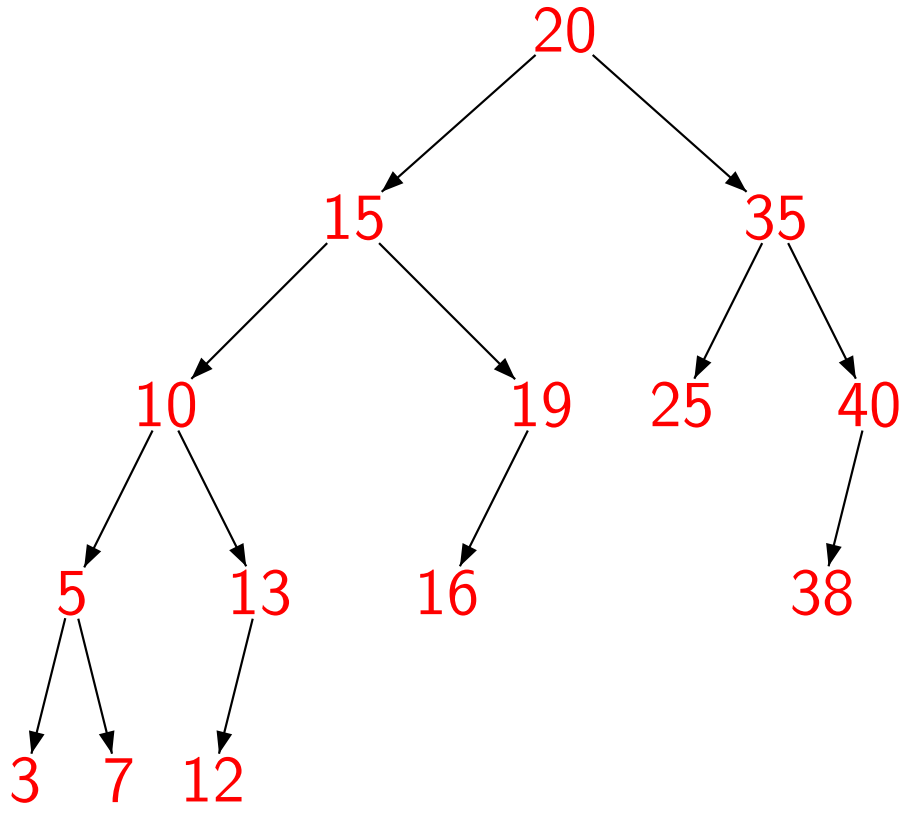
3 5 7 10 12 13 15 16 19 20 25 35

Exemple...



3 5 7 10 12 13 15 16 19 20 25 35 38

Exemple...



3 5 7 10 12 13 15 16 19 20 25 35 38 40

Complexité du tri par ABR

Dans le pire cas, la complexité est en $O(n^2)$.

En moyenne, la complexité est en $O(n \cdot \log(n))$:

Le nombre moyen de comparaisons de clés effectuées pour construire un ABR en insérant n clés distinctes dans un ordre aléatoire à partir d'un ABR vide est :

$$2(n+1)(H_{n+1} - 1) - 2n$$

Ce nombre est donc en $O(n \cdot \log(n))$.

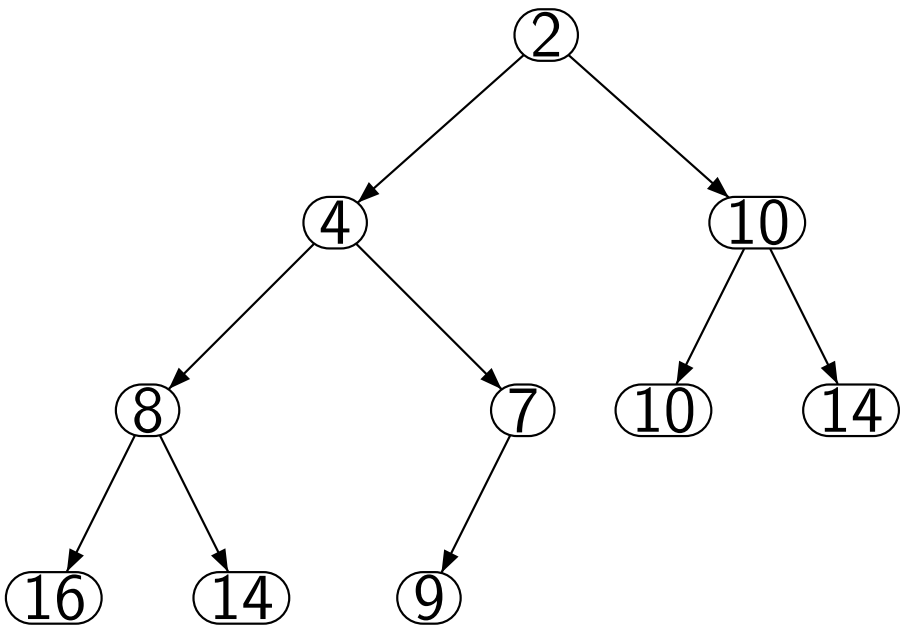
Plan

Tri par tas

Un tas est un arbre binaire...

- parfait (tous les niveaux sont remplis sauf éventuellement le dernier et les feuilles sont regroupées à gauche), et
- chaque noeud contient une valeur inférieure (ou égale) à celles stockées dans ses sous-arbres.

Exemple :



Représentation des tas

Un tas se représente facilement avec une paire (T, n) :

- un entier n donnant le nombre d'éléments du tas, et
- un tableau T (de taille $\geq n$).

$T[1]$ correspond à la racine du tas.
Tout noeud i a son père en $\frac{i}{2}$, et :

- son fils gauche en $2 \cdot i$ (si il existe, c.-à-d.$2i \leq n$), et
- son fils droit en $2 \cdot i + 1$ (si $2i + 1 \leq n$).

Algorithmes sur les tas

On dispose d'algorithmes efficaces pour :

- ajouter un élément au tas,
- extraire le minimum (et remettre l'arbre en tas), et

Efficace = linéaire dans la hauteur de l'arbre, donc en $O(\log(n))$.

Algorithmes sur les tas

On dispose d'algorithmes efficaces pour :

- ajouter un élément au tas,
- extraire le minimum (et remettre l'arbre en tas), et

Efficace = linéaire dans la hauteur de l'arbre, donc en $O(\log(n))$.

Le tri par tas consiste à :

- transformer T en un tas, $O(n)$
- puis extraire les éléments un à un... $O(n \log n)$

Opérations sur les tas : ajouter

Ajouter(T, n, x)

$n \leftarrow n + 1$

$i \leftarrow n$

tant que $i/2 > 0$ && $T[i/2] > x$ **faire**

  $T[i] \leftarrow T[i/2]$

  $i \leftarrow i/2$

$T[i] \leftarrow x$

Opérations sur les tas : extraire-min

1. retourner $T[1]$, $T[1] \leftarrow T[n]$, $n \leftarrow n-1$,
2. puis appeler Tasser$(T, n, 1)$

Tasser(T, n, i)

$g \leftarrow 2 \cdot i$,
$d \leftarrow 2 \cdot i + 1$
$min \leftarrow i$
si $g \leq n$ && $T[g] < T[i]$ **alors** $min \leftarrow g$
si $d \leq n$ && $T[d] < T[min]$ **alors** $min \leftarrow d$
si $min \neq i$ **alors**
  $T[i] \leftrightarrow T[min]$
  Tasser(T, n, min)

(NB : $T[2i]$ et $T[2i+1]$ sont des racines de tas)

ABR vs Tas

Les ABR et les tas (ou Files de priorité) servent à stocker des éléments en fonction d'une clé.

Pour les ABR, les opérations suivantes sont facilement implémentables :

- Ajouter, Supprimer et Rechercher un élément,
- Parcourir dans l'ordre.

Leur complexité sont en $O(h)$.

Pour les tas, on peut faire :

- Ajouter un élément,
- Extraire le plus petit élément.

Leur complexité sont en $O(\log(n))$.

Plan

Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Non si on n'utilise que des comparaisons 2 à 2 et sans hypothèse sur le contenu du tableau...

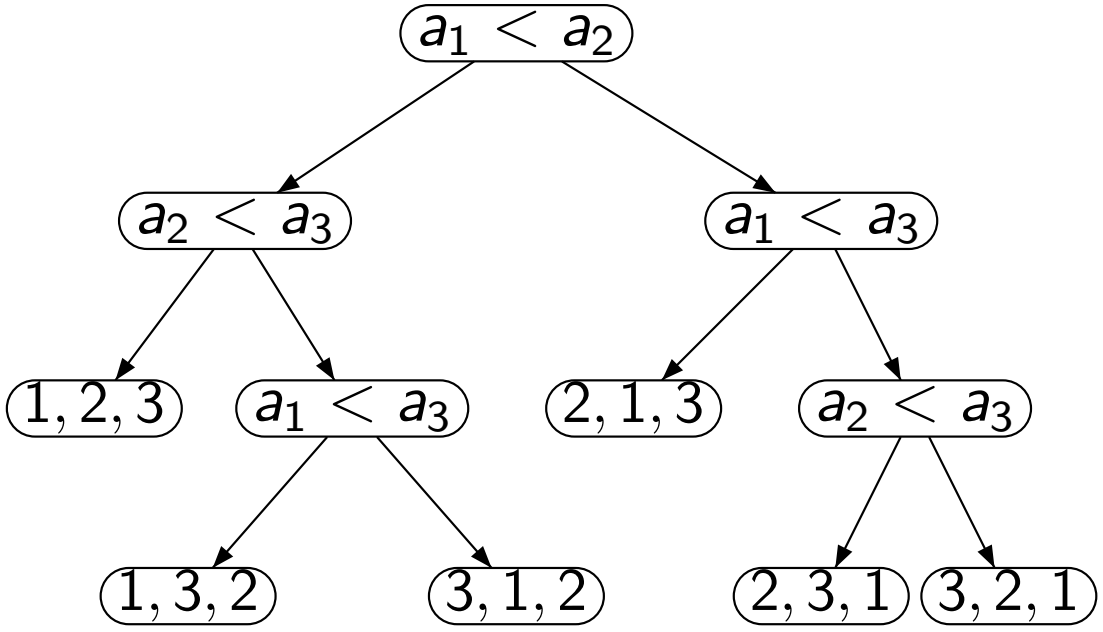
Optimalité des algorithmes de tri

Question : Est-il possible de trier un tableau de n éléments en moins de $n \cdot \log(n)$ opérations dans le pire cas ?

Non si on n'utilise que des comparaisons 2 à 2 et sans hypothèse sur le contenu du tableau...

Dans ce cadre, tout algorithme de tri peut être représenté par un arbre de décision où chaque noeud correspond à un test de deux éléments (le fils gauche correspond à la réponse négative et le droit la réponse positive).

Optimalité des algorithmes de tri



Chaque feuille correspond à la permutation à effectuer par l'algorithme. Il y a donc au moins $n!$ feuilles dans tout arbre de décision pour trier n éléments.
La complexité dans le pire correspond à la hauteur de l'arbre.
Tout arbre binaire équilibré a une hauteur log(nb feuilles).

Le tri fusion (et le tri par tas) sont optimaux (asymptotiquement).

Attention aux hypothèses

Si on trie des pailles...

Attention aux hypothèses

Si on trie des pailles...

Le tri est linéaire !

Attention aux hypothèses

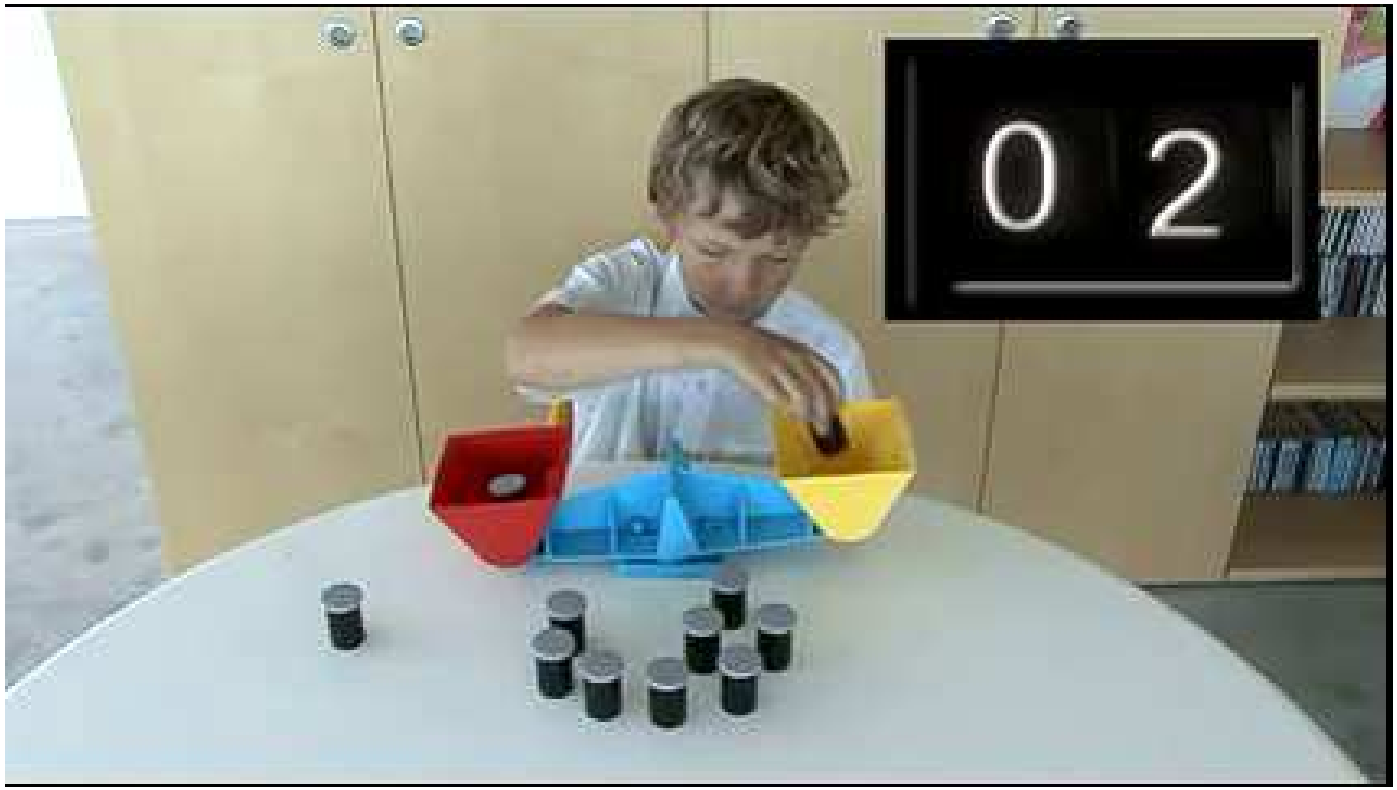
Si on trie des pailles...

Le tri est linéaire !

Si on connaît le nombre de valeurs possibles pouvant figurer dans le tableau... c'est aussi linéaire !

Plan

Une idée d'activité en classe



- Activité de découverte des algorithmes de tri (conçue pour le primaire)
- Description en Français téléchargeable gratuitement sur http://www.csunplugged.org

Plan

Travaux pratiques

Programmes en Python...

(Pourquoi Python ?)

Tri par sélection

```
def IndiceMin(T,i) :
    res = i
    for k in range(len(T))[i+1:] :
        if T[k] < T[res] : res = k
    return res

def TriSelection(T) :
    for i in range(len(T)-1) :
        ind = IndiceMin(T,i)
        T[i], T[ind] = T[ind], T[i]
    return T
```

Tri par insertion

```
def Inserer(T,i) :
    if i>0 :
        if T[i-1] > T[i] :
            T[i-1], T[i] = T[i], T[i-1]
            Inserer(T,i-1)

def TriInsertion(T) :
    for i in range(len(T))[1:] :
        v = T[i]
        Inserer(T,i)
    return T
```

Tri par fusion

```
def TriFusion(T) :
  if len(T) > 1 :
      if len(T) == 2 :
          if T[0]>T[1] : T[0],T[1] = T[1],T[0]
      else :
          m = len(T)/2
          T = Fusion(TriFusion(T[:m]),TriFusion(T[m:]))
  return T
```

La fusion de deux tableaux triés

```
def Fusion (L1,L2) :
    res = [ ]
    i1 = i2 = 0
    while i1<len(L1) and i2 < len(L2) :
        if L1[i1] < L2[i2] :
            res.append(L1[i1])
            i1 = i1+1
        else :
            res.append(L2[i2])
            i2 = i2+1
    if i1==len(L1) :
        res += L2[i2:]
    else :
        res += L1[i1:]
    return res
```

Tri rapide

```
def quicksort(T,bg =0,bd = None) :
    if bd== None : bd=len(T)-1
    if bg<bd :
        indp = indicepivot(T,bg,bd)
        quicksort(T,bg,indp-1)
        quicksort(T,indp+1,bd)
```

Pivotage

```
def indicepivot(T,bg,bd) :
    p = T[bg]
    l=bg+1
    r=bd
    while (l<=r) :
        while (l<=bd) and (T[l]<=p) :
            l += 1
        while (T[r] > p) :
            r  -= 1
        if (l < r) :
            T[r], T[l] = T[l], T[r]
            l , r = l+1, r-1
    T[r], T[bg]  = p, T[r]
    return r
```