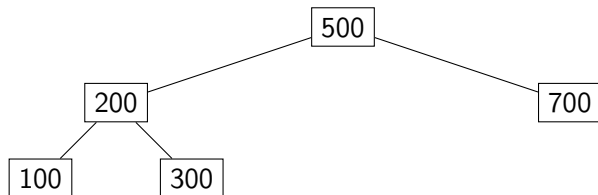


# Tas et tri par tas

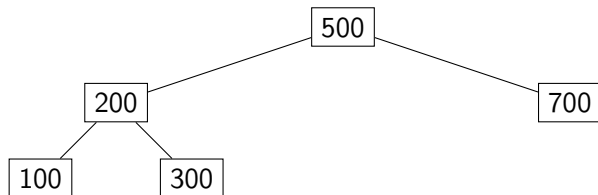
# comment un tableau représente-t-il un arbre parfait ?

L'arbre



# comment un tableau représente-t-il un arbre parfait ?

L'arbre

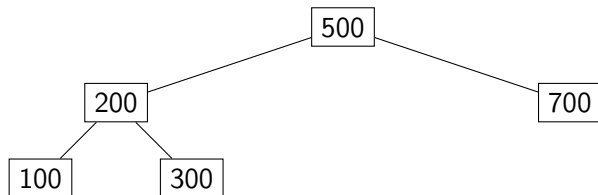


le tableau utile correspondant

500	200	700	100	300
1	2	3	4	5

# comment un tableau représente-t-il un arbre parfait ?

L'arbre




le tableau utile correspondant

500	200	700	100	300
1	2	3	4	5

un vrai tableau possible

IndicePremierSommetLibre=6

NombreMaximalDeSommets=10



500	200	700	100	300				
1	2	3	4	5	6			

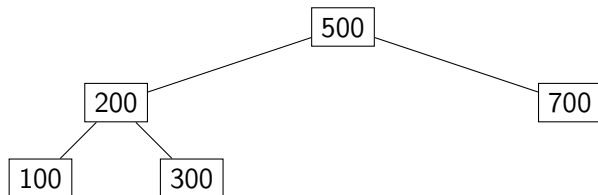
```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
  
};
```

```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
struct ArbreParfait{  
  
};
```

```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
struct ArbreParfait{  
    Sommet IndicePremierSommetLibre ;  
    int NombreMaximalDeSommets ;  
    Valeur contenu[] ;  
};
```

# comment un tableau représente-t-il un arbre ?

L'arbre




le tableau utile correspondant

500	200	700	100	300
1	2	3	4	5

un vrai tableau possible

IndicePremierSommetLibre=6

NombreMaximalDeSommets=10



500	200	700	100	300				
1	2	3	4	5	6			



```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
struct ArbreParfait{  
    Sommet IndicePremierSommetLibre ;  
    int NombreMaximalDeSommets ;  
    Valeur contenu[] ;  
    ArbreParfait(int) ;  
//on passe en parametre le nombre maximal de sommets de l'arbre  
  
};
```

```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
struct ArbreParfait{  
    Sommet IndicePremierSommetLibre ;  
    int NombreMaximalDeSommets ;  
    Valeur contenu[] ;  
    ArbreParfait(int) ;  
//on passe en parametre le nombre maximal de sommets de l'arbre  
    void AjouteSommetArbreParfait(Valeur) ;  
  
};
```

```
typedef int Sommet ;  
// un sommet sera un indice de tableau  
typedef int Valeur ;  
une Valeur est le contenu des noeuds (pour nous des entiers)  
struct ArbreParfait{  
    Sommet IndicePremierSommetLibre ;  
    int NombreMaximalDeSommets ;  
    Valeur contenu[] ;  
    ArbreParfait(int) ;  
//on passe en parametre le nombre maximal de sommets de l'arbre  
    void AjouteSommetArbreParfait(Valeur) ;  
    void SupprimerDansArbreParfait(Sommet) ;  
// on passe l'indice dans le tableau du sommet que l'on veut supprimer  
};
```



# arbre parfait

Les méthodes de base : constructeur et ajouter

```
ArbreParfait : :ArbreParfait(int n){  
    NombreMaximalDeSommets =n ;  
    IndicePremierSommetLibre=0 ;  
    contenu=new Valeur[n] ;  
}
```

arbre parfait

## Les méthodes de base : constructeur et ajouter

```
ArbreParfait : :ArbreParfait(int n){
    NombreMaximalDeSommets = n ;
    IndicePremierSommetLibre=0 ;
    contenu=new Valeur[n] ;
}
```

```
void ArbreParfait : :AjouteSommetArbreParfait(Valeur v){  
  
  
  
  
  
  
}
```

# arbre parfait

Les méthodes de base : constructeur et ajouter

```
ArbreParfait : :ArbreParfait(int n){  
    NombreMaximalDeSommets =n ;  
    IndicePremierSommetLibre=0 ;  
    contenu=new Valeur[n] ;  
}
```

```
void ArbreParfait : :AjouteSommetArbreParfait(Valeur v){  
    if (IndicePremierSommetLibre < NombreMaximalDeSommets){  
  
    }  
}
```

# arbre parfait

Les méthodes de base : constructeur et ajouter

```
ArbreParfait : :ArbreParfait(int n){
    NombreMaximalDeSommets =n ;
    IndicePremierSommetLibre=0 ;
    contenu=new Valeur[n] ;
}

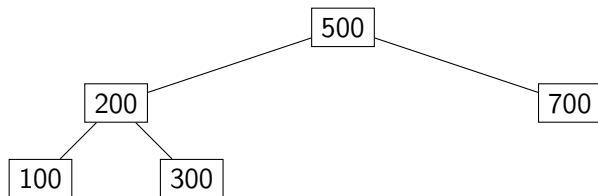
void ArbreParfait : :AjouteSommetArbreParfait(Valeur v){
    if (IndicePremierSommetLibre < NombreMaximalDeSommets){
        contenu[IndicePremierSommetLibre]=v ;
        IndicePremierSommetLibre++ ;}
    }
}
```



Le tableau

500	200	700	100	300	
-----	-----	-----	-----	-----	--

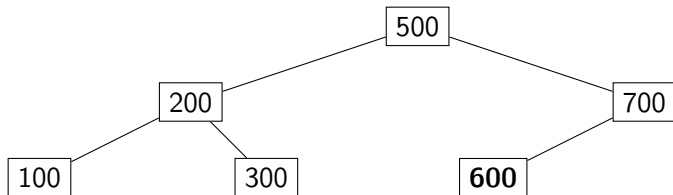
L'arbre correspondant



Le tableau

500	200	700	100	300	<b>600</b>
-----	-----	-----	-----	-----	------------

L'arbre correspondant



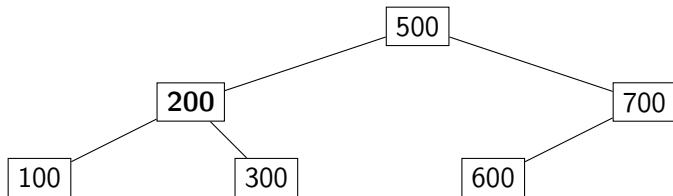
# arbre parfait

on veut maintenant retirer le sommet à la case d'indice 2

Le tableau

500	<b>200</b>	700	100	300	600
-----	------------	-----	-----	-----	-----

L'arbre correspondant



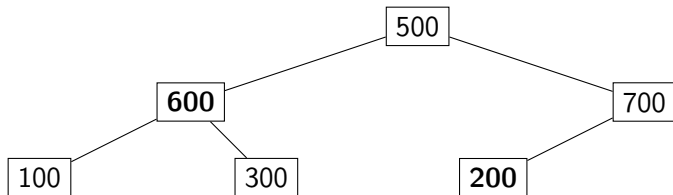
# arbre parfait

on échange les deux sommets

Le tableau

500	<b>600</b>	700	100	300	<b>200</b>
-----	------------	-----	-----	-----	------------

L'arbre correspondant



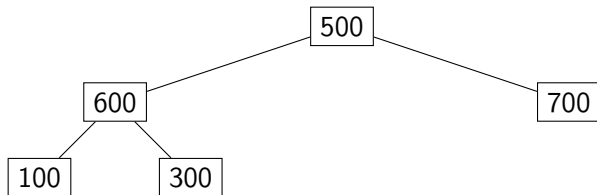
# arbre parfait

on a retiré le bon sommet

Le tableau

500	600	700	100	300	
-----	-----	-----	-----	-----	--

L'arbre correspondant



```
void ArbreParfait : :SupprimerDansArbreParfait(Sommet indice){  
    // 0 < indice < IndicePremierSommetLibre  
  
}
```

```
void ArbreParfait : :SupprimerDansArbreParfait(Sommet indice){  
    // 0 < indice < IndicePremierSommetLibre  
    EchangerSommets(indice, IndicePremierSommetLibre - 1);  
    IndicePremierSommetLibre-- ;  
}
```

# Tas

Ce qu'on veut



Trouver le plus petit sommet en

Trouver le plus petit sommet en  $\theta(1)$

Trouver le plus petit sommet en  $\theta(1)$   
ajout et retrait

Trouver le plus petit sommet en  $\theta(1)$   
ajout et retrait pas cher

Trouver le plus petit sommet en  $\theta(1)$   
ajout et retrait pas cher

La solution :

Trouver le plus petit sommet en  $\theta(1)$   
ajout et retrait pas cher

La solution :

Chaque sommet est à ses descendants

Trouver le plus petit sommet en  $\theta(1)$   
ajout et retrait pas cher

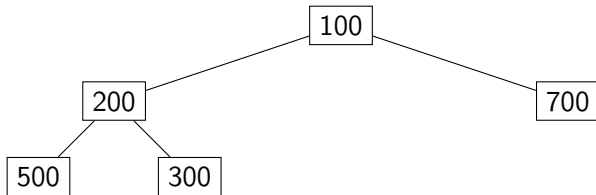
La solution :

Chaque sommet est inférieur à ses descendants

Le tableau

100	200	700	500	300
-----	-----	-----	-----	-----

L'arbre correspondant

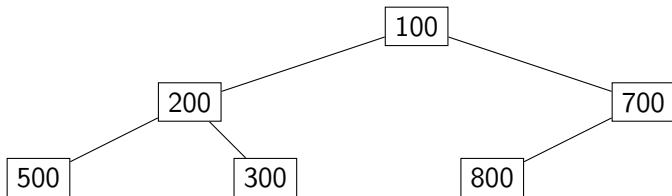




Le tableau

100	200	700	500	300	800
-----	-----	-----	-----	-----	-----

L'arbre correspondant



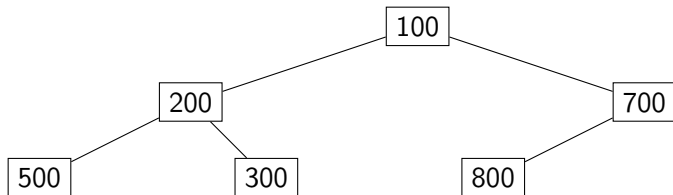
# Tas

on a bien ajouté 800 dans le tas

Le tableau

100	200	700	500	300	800
-----	-----	-----	-----	-----	-----

L'arbre correspondant



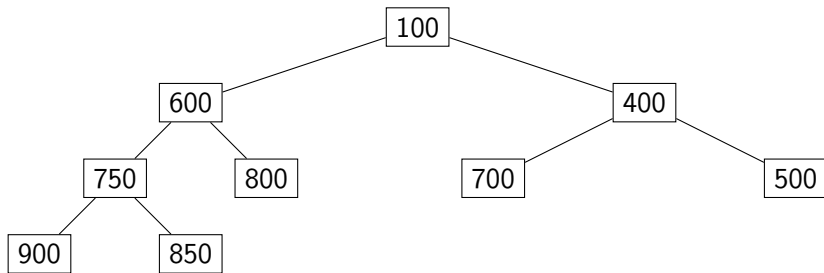
# Tas

un exemple plus compliqué

Le tableau

100	600	400	750	800	700	500	900	850
-----	-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



# Tas

ajoutons 350 dans l'arbre parfait

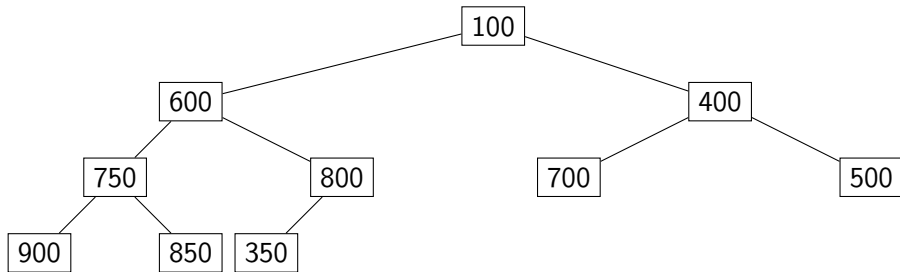
Le tableau

100	600	400	750	800	700	500	900	850	350
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

5

10

L'arbre correspondant



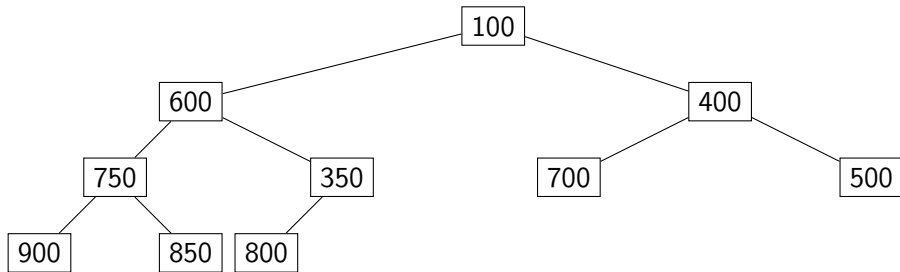
# Tas

remontons d'un niveau

Le tableau

100	600	400	750	350	700	500	900	850	800
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



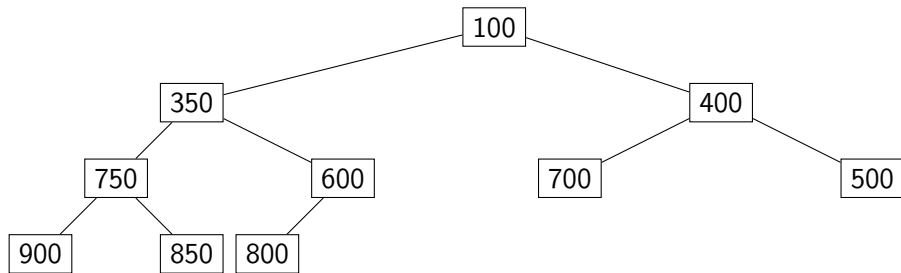
# Tas

remontons encore d'un niveau pour obtenir un tas

Le tableau

100	350	400	750	600	700	500	900	850	800
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



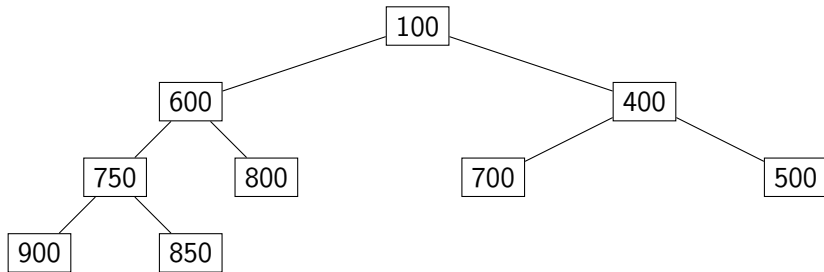
# Tas : Suppression

supprimons la racine de l'arbre parfait

Le tableau

100	600	400	750	800	700	500	900	850
-----	-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



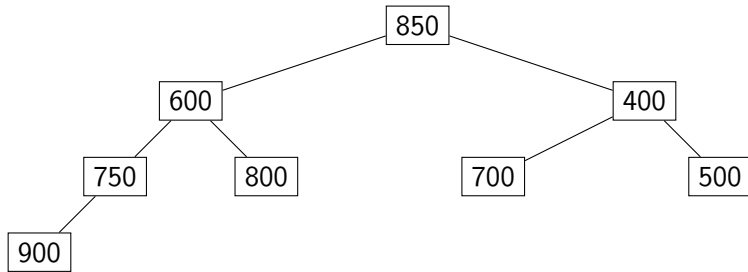
# Tas : Suppression

transformons le nouvel arbre parfait en tas

Le tableau

850	600	400	750	800	700	500	900
-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant





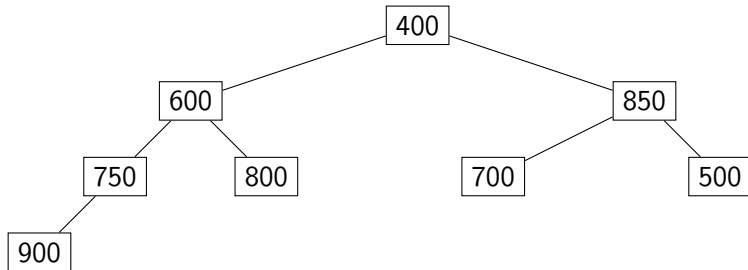
# Tas : Suppression

grace à ce passage intermédiaire

Le tableau

400	600	850	750	800	700	500	900
-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



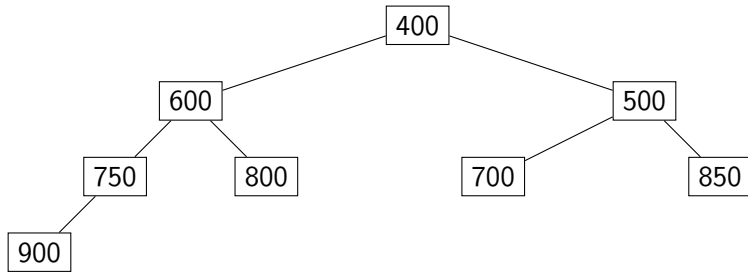
# Tas : Suppression

voilà le résultat que l'on veut obtenir

Le tableau

400	600	500	750	800	700	850	900
-----	-----	-----	-----	-----	-----	-----	-----

L'arbre correspondant



```
struct Tas : public ArbreParfait {  
  
    Tas(int);  
    le nombre maximal de sommets de l'arbre  
  
    void Remonter(Sommet);  
    void Descendre(Sommet);  
  
    void SupprimerTas(Sommet);  
  
    void AjouterTas(Valeur);  
  
    int Supmin();  
};
```