Partie I - Tri par insertion

1) Fonction classique

```
let rec insere x u =
   match u with
  |[] -> [x]
  |a::q -> if x<=a then x::u
           else a::(insere x q) ;;
```

2) Fonction classique

```
let rec tri_insertion l =
    match l with
   |[] -> []
   |x::u -> insere x (tri_insertion u);;
```

3) Dans le cas le meilleur, la fonction `insere` effectue une comparaison (quand la liste argument est non vide, 0 quand elle est vide) et dans le cas le pire elle en effectue $|u|$ (notation pour le nombre des éléments de u). On a donc

$$\forall n \geq 2,\ M_I(n) = 1 + M_I(n-1) \ \text{ et } \ \forall n \geq 1 P_I(n) = (n-1) + P_I(n-1)$$

Comme $P_I(0) = 0$ et $M_I(1) = 0$, une récurrence immédiate donne

$$\forall n \in \mathbb{N}^*,\ M_I(n) = n-1 \ \text{ et } \ \forall n \in \mathbb{N},\ P_I(n) = \sum_{k=0}^{n-1} k = \frac{n(n-1)}{2}$$

Ces deux cas existent bien. Le premier est le cas où la liste est déjà croissante, le deuxième où la liste était décroissante.

Partie II - Tas binaires

4) On a $m_0 = 0$ (l'arbre vide est de taille nulle) et

$$\forall k \in \mathbb{N},\ m_{k+1} = 2m_k + 1$$

On en déduit, par exemple par récurrence, que

$$\forall k \in \mathbb{N},\ m_k = 2^k - 1$$

5) Le minimum est à la racine

```
let min_tas (Noeud(x,a1,a2)) = x;;
```

6) Les deux fils d'un quasi-tas sont simultanément vides ou non vides. Dans le second cas, le minimum est égal au plus petit de la racine et des racines des fils.

```
let min_quasi = function
   |Noeud(x,Vide,Vide) ->  x
   |Noeud(x,a1,a2) -> min x (min (min_tas a1) (min_tas a2));;
```

7) L'arbre est un tas si la racine x est plus petite que le minimum des racines des fils (quand elles existent). Dans ce cas, on renvoie l'arbre. Dans l'autre, on échange x avec la racine la plus petite et on percole dans l'arbre associé.

```
let rec percole a = match a with
  | Vide -> Vide
  | Noeud (x, Vide, Vide) -> a
  | Noeud (x, Noeud (x1, g1, d1), Noeud (x2, g2, d2)) ->
     if x <= (min x1 x2) then a
     else if x1 < x2 then
       Noeud (x1, percole (Noeud (x, g1, d1)), Noeud (x2, g2, d2))
     else Noeud (x2, Noeud (x1, g1, d1), percole (Noeud (x, g2, d2)));;
```

L'appel récurrent se faisant sur un unique fils, on ne fait le parcours que d'une seule branche. À chaque étape, on effectue un nombre constant d'opérations. La complexité est donc linéaire en fonction de la hauteur de l'arbre argument.

Partie III - Décomposition parfaite d'un entier

8) On a

$$6 = m_2 + m_2,\ 7 = m_3,\ 8 = m_1 + m_3,\ 9 = m_1 + m_1 + m_3,\ 10 = m_2 + m_3$$

$$27 = m_1+m_1+m_2+m_3+m_4,\ 28 = m_2+m_2+m_3+m_4,\ 29 = m_3+m_3+m_4,\ 30 = m_4+m_4,\ 31 = m_5$$

9) On suppose $n = m_{k_1} + \cdots + m_{k_r}$ avec $(k_1, \ldots, k_r)$ qui vérifie la propriété QSC. On distingue des cas.
- Si $r = 0$ alors $n = 0$ et $n + 1 = m_1$ ce qui correspond bien au second cas de l'énoncé.
- Si $r = 1$ alors $n = m_{k_1}$ et $n + 1 = m_1 + m_{k_1}$ ce qui correspond bien au second cas de l'énoncé.
- Si $r \geq 2$ et $k_1 \neq k_2$ alors $n + 1 = m_1 + m_{k_1} + \cdots + m_{k_r}$ ce qui correspond bien au second cas de l'énoncé.
- Sinon $r \geq 2$ et $k_1 = k_2$ et $n + 1 = m_1 + 2m_{k_1} + m_{k_3} + \cdots + m_{k_r} = m_{k_1+1} + (m_{k_3} + \cdots + m_{k_r})$.

Les formules de l'énoncé sont donc correctes. Il reste à remarquer que dans les deux cas de l'énoncé, la décomposition donnée correspond bien à un uplet qui vérifie la propriété QSC, ce qui est immédiat.

10) Une première fonction `transforme` modélise en termes de listes les transformations évoquées dans la question précédente. La fonction principale fait un appel à l'entier inférieur et transforme le résultat.

```
let transforme l = match l with
    | [] -> [1]
    | [x] -> [1; x]
    | x :: y :: q -> if x = y then (2 * x + 1) :: q
        else 1 :: l;;

let rec decomp_parf n =
  if n = 0 then []
  else transforme (decomp_parf (n - 1));;
```

Partie IV - Création d'une liste de tas

11) a) Soit $h = ((a_1, t_1), \ldots, (a_r, t_r))$ une liste de tas. La hauteur de ce tas est égale à la hauteur de l'un des a_i. Sa taille est plus grande que la taille de $|a_i|$ qui vaut $2^{\text{haut}(a_i)} + 1$. On a donc $2^{\text{haut}(h)} \leq |h| + 1$ et donc $\text{haut}(h) = O(\log_2(|h|))$.
Considérons la liste de tas comportant r arbres réduits à leurs racines. La longueur de ce tas est r et sa taille est aussi égale à r. La longueur n'est donc en général pas dominée par le logarithme de la taille.

b) Montrons que le second des résultats précédents est valable quand la condition TC est vérifiée. Soit donc $h = ((a_1, t_1), \ldots, (a_r, t_r))$ une liste de tas croissants. La taille de a_i est égale à la somme des t_i. Comme $(t_1, \ldots, t_r)$ vérifie QSC, on a $t_1 \geq m_1$, $t_2 \geq m_1$, $t_3 \geq m_2$, …, $t_r \geq m_{r-1}$ et donc

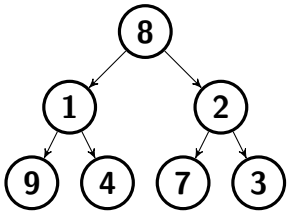
$$|h| \geq 1 + \sum_{k=1}^{r-1} m_k \geq 1 + \sum_{k=1}^{r-1} 2^k = 2^{r-1}$$

On en déduit que $r = \text{long}(h) \leq \log_2(|h|) + 1 = O(\log_2(|h|))$.

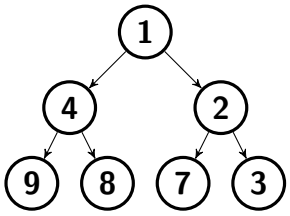
12) a) Le tas h_1 est de taille $11 = m_1 + m_2 + m_3$. On en déduit que h'_1 sera de taille $12 = m_1 + m_1 + m_2 + m_3$ et donc composée de 4 tas de tailles $1, 1, 3, 7$. Il suffit ici de poser

$$h'_1 = (a, 1) :: h_1 \quad \text{avec} \quad a = \texttt{Noeud}(8, \texttt{Vide}, \texttt{Vide})$$

Le tas h_2 est de taille $13 = m_2 + m_2 + m_3$ et h'_2 sera de taille $14 = m_3 + m_3$. Les tas a_2^1 et a_2^2 fusionnent avec l'arbre réduit à la racine 8. Dans un premier temps, cette fusion n'est qu'un quasi-tas représenté ci-dessous :



Il suffit alors d'utiliser l'opération de percolation pour obtenir



h'_2 est constituée de l'arbre ci-dessus et de a_2^3.

b) Le processus général est le même et reprend l'idée de l'obtention d'une décomposition parfaite de $n + 1$ à partir de celle de n.
- Si h est vide ou de longueur 1, on pose $h' = (a, 1) :: h$.
- Sinon, notons (a_1, t_1) et (a_2, t_2) les deux premiers éléments de t. Si $t_1 < t_2$, on pose $h' = (a, 1) :: h$. Sinon $t_1 = t_2$ et on fusionne a, a_1, a_2 pour obtenir un quasi-tas (puisque a_1 et a_2 sont des tas de même taille) et on utilise la percolation. On obtient un tas a' de taille $t' = 1 + t_1 + t_2$ et on pose $h' = [(a', t'), (a_3, t_3), \ldots]$.

Il est clair que la liste renvoyée a les mêmes éléments que $(a, 1) :: h$ et que c'est une liste de tas. Elle vérifie la condition TC car la suite des tailles vérifie QSC (c'est l'algorithme de décomposition parfaite qui le montre et cet algorithme a été prouvé).

Dans le meilleur cas, un consage suffit. Dans le pire, on utilise la percolation dont la complexité est linéaire en la hauteur du quasi-tas, c'est à dire aussi en la hauteur de a_1 (plus 1, ce qui ne change rien).

c) Il suffit de suivre fidèlement le programme précédent.

```
let ajoute x h =
     match h with
    |[] -> [Noeud(x,Vide,Vide),1]
    |[(a1,t1)] -> (Noeud(x,Vide,Vide),1)::h
    |(a1,t1)::(a2,t2)::q ->
                if t1<t2 then (Noeud(x,Vide,Vide),1)::h
                else (percole (Noeud(x,a1,a2)),1+t1+t2)::q ;;
```

13) a) Soit C_n le nombre d'opération dans le cas le pire dans l'appel `construit_tas l` pour `l` liste triée d'entiers. On a $C_{n+1} = C_n = \alpha$ où α est le coût de l'ajout de l'arbre de racine l_1 à une liste de tas d'étiquettes toutes $\geq l_1$. Cet ajout a un coût $O(1)$ sauf s'il induit une percolation. Mais si cette dernière a lieu, elle se fait en temps constant car, avec l'hypothèse sur les étiquettes, elle est appelée avec un tas (et non un quasi-tas). Ainsi $C_{n+1} = C_n + O(1)$. Comme $C_0 = O(1)$, on a bien $C_n = O(n)$.

b) Dans l'appel `construit_tas l`, on effectue n ajouts avec des tas qui ont moins de n éléments et donc qui sont de hauteur au plus $\log_2(n)$. Un ajout se fait donc en temps inférieur à $O(\log_2(n))$ et le coût total est $O(n \log_2(n))$.