```
let decomposer entier n =
  let rec aux n n2 acc =
      match n, n2 with
           l, l2 when l2 = 0 \rightarrow acc \otimes [(l, l2)]
           l, l2 \rightarrow acc \otimes [(l, l2)] \otimes aux (l+1) (l2-1) acc
      in aux 0 n [];;
(* Ouestion 2. *)
let ajouter noeud unaire arbres =
  List.map (fun id \rightarrow Unaire(id)) arbres;
let ajouter_noeud_binaire arbres1 arbres2 =
  let rec aux a a2 acc =
      match a, a2 with
          [], \_ \rightarrow acc
          h::t, [] \rightarrow acc \otimes aux t arbres2 acc
          h::t, h'::t' \rightarrow acc \otimes [Binaire (h, h')] \otimes aux a (t') acc
  in aux arbres1 arbres2 [];;
let taille arb =
  let rec aux arb count =
      match arb with
           Binaire(a, b) \rightarrow count + aux a 1 + aux b 1
           Unaire(a) \rightarrow count + aux a 1
           Feuille(v) \rightarrow 0
  in aux arb 1;;
let rec generer_arbres v n =
  if n = 0 then [Feuille v]
  else
      let lst1 = ajouter_noeud_unaire (generer_arbres v (n - 1))
      and decomp = decomposer_entier (n-1) in
      let lst2 = List.fold_left (fun acc (a, b) \rightarrow acc @
      (ajouter_noeud_binaire (generer_arbres v a) (generer_arbres v b))) [] decomp
  in lst1 @ lst2;;
```