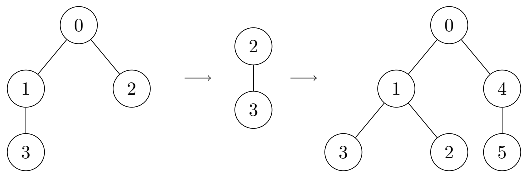
TD - Fil de priorité et Tas

I Un tas de question

1.



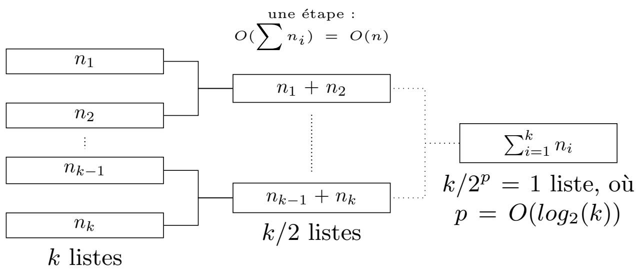
2.
3.

```
let rec is_tas_max heap =
    test = ref true in
    for i = 0 to heap.n - 1 do
        if a.(i) > a.((i-1)/2) then
        test := false
    done;
    !test ;;
```

4.
5.

```
let rec fusion l1, l2 = match l1, l2 with
    |[], _ -> l2
    |_, [] -> l1
    |e1::q1, e2::q2 -> if e1 < e2 then
                            e1::fusion q1 l2
                        else
                            e2::fusion l1 q2
;;
```

La complexité est $O(n+m)$



d'où une complexité en $O(nlog(k))$

```
let rec etape ll = match ll with
    |l1::l2::q -> (fusion l1 l2)::etape q
    |_ -> ll
;;

let rec kfusion ll = match ll with
```

```
        |[] -> []
        |[l] -> l
        |_ -> kfusion (etape ll)
;;
```

II Compression de Huffman

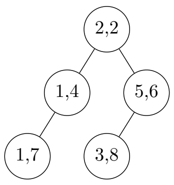
III Arbretas

1.

```
let swap t i j =
    let tmp = t.(j) in
    t.(j) <- t.(i); t.(i) <- tmp
;;
```

2. La fonction `shuflle t` permutte tous les éléments du tableau `t` avec un autre éléments du tableau choisi au hasard dans les indices inférieurs. Après avoir terminé la boucle `for`, le tableau a donc subit une permutation aléatoires.

3.

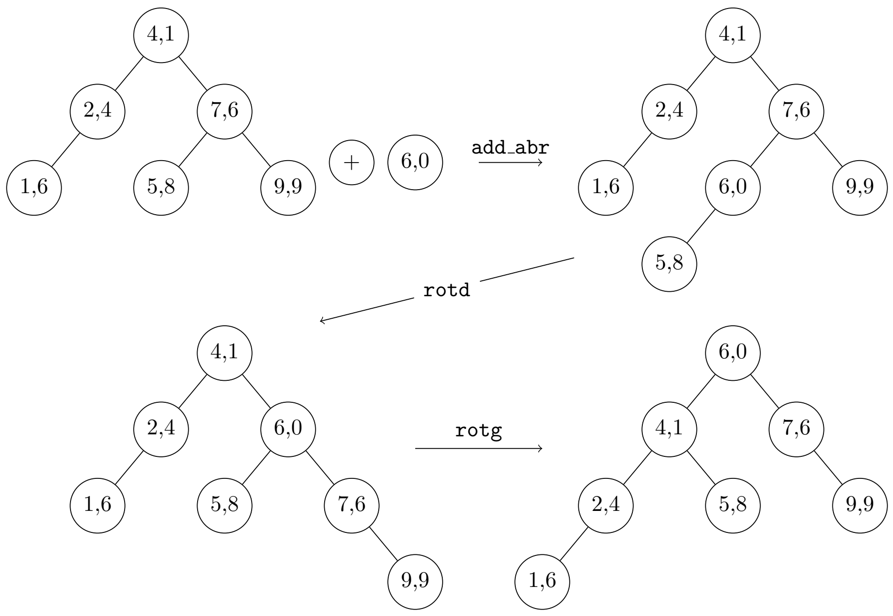


4.

5.

```
let rotd treap = match treap with
|N(r, N(gr, gg, gd), d) -> N(gr, N(r, gd, d), gg)
|_ -> treap
```

6.



7.

```
let prio tree = match tree with
|V -> max_int
|N((_, p), _, _) -> p
;;
```

8.

```
let rec add treap e =
  let elem, _ = e in
  match treap with
  |V -> N(e, V, V)
  |N((x, p), g, d) -> if elem >= x then
                          let d_upt = add d e in
                              if (prio d_upt) < p then
                                rotg (N((x,p), g, d_upt))
                              else N((x,p), g, d_upt)
                      else
                          let g_upt = add g e in
                          if (prio g_upt) < p then
                              rotd (N((x,p), g_upt, d))
                            else N((x,p), g_upt, d)
                            ;;
```

9.

```
let rec del treap e = match treap with
|V -> V
|N((x,p), g, d) -> if e > x then
                     N((x,p), g, (del d e))
                   else if e < x then
                     N((x,p), (del g e), d)
                   else match g,d with
                     |V, V -> V
                     |V, f |f, V -> f
                     |_ -> if prio g < prio d then
                          let treap_rot = rotd(treap) in
                            del treap_rot e
                          else let treap_rot = rotg(treap) in
                            del treap_rot e
                        ;;
```