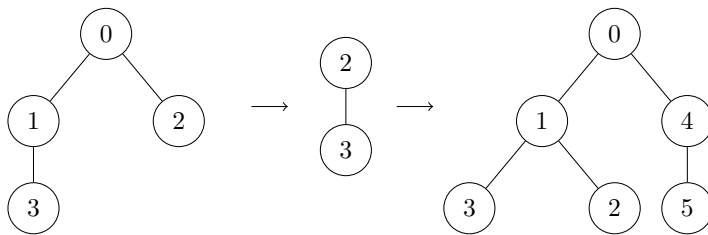


TD - Fil de priorité et Tas

I Un tas de question

1.



2.

3.

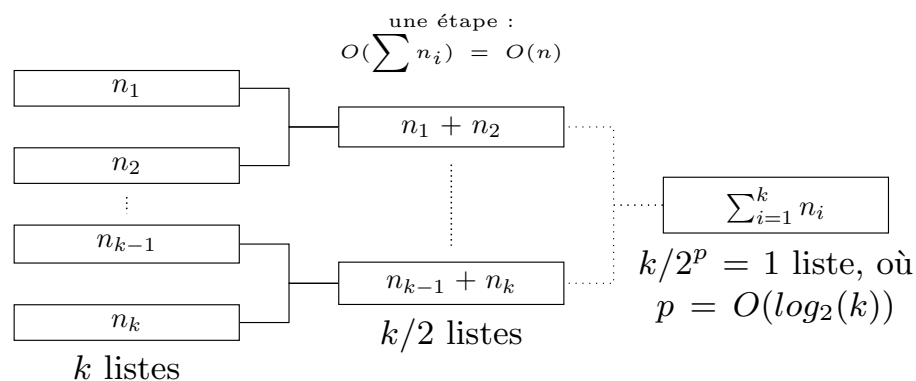
```
let rec is_tas_max heap =
  test = ref true in
  for i = 0 to heap.n - 1 do
    if a.(i) > a.((i-1)/2) then
      test := false
  done;
  !test ;;
```
4.

```
type 'a heap = {a : 'a array ; mutable n : int}
type 'a tree = E | N of 'a * 'a tree * 'a tree

let heap_to_tree heap =
  let rec aux i = (*renvoie le sous-arbre enraciné en heap.a *)
    if i >= heap.n
    then E
    else N(heap.a.(i), aux(2*i + 1), aux(2 * (i + 1))) in
  aux 0 ;;
```
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 $O(n \log(k))$

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

```

    | _ -> l1 ;;

let rec kfusion l1 = match l1 with
| [] -> []
| [1] -> 1
| _ -> kfusion (etape l1) ;;

```

II Compression de Huffman

1.

```

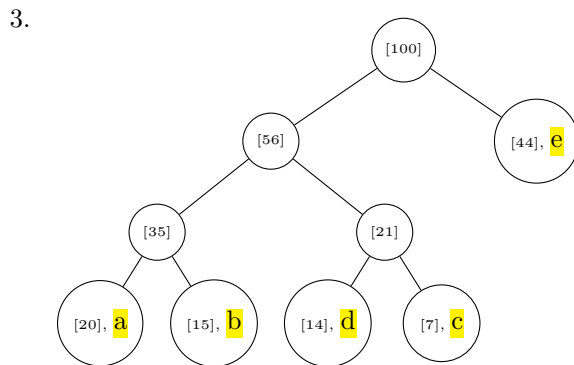
let rec read arb lst =
match arb with
| F(chr) -> chr, lst
| N(g, d) -> match lst with
                | 0::t -> read g t
                | 1::t -> read d t
                | _ -> failwith "liste non valide"

```
2.

```

let rec decode arb lst =
match (read arb lst) with
| chr, [] -> chr::[]
| chr, lst_end -> chr::(decode arb lst_end) ;;

```



4.

```

let to_huffman freq =
let fp = create () in
let rec fill freq=
match freq with
| [] -> ()
| (f, chr)::t -> add fp (f, F(chr)) ; fill t in
fill freq;
while len_over_2 fp do
let (f1, a1), (f2, a2)= extract_min fp, extract_min fp in
add fp (f1 + f2, N(a1, a2))
done;
snd (extract_min fp) ;;

```
5.

```

let to_dict arb =
let dict = Hashtbl.create 100 in
let rec aux arb dict code =
match arb with
| F(chr) -> Hashtbl.add dict chr (List.rev code)
| N(g, d) -> aux g dict (0::code) ; aux d dict (1::code)
in
aux arb dict [] ; dict ;;

```
6.

```

let rec code lst dict =
match lst with
| [] -> []
| h::t -> (Hashtbl.find dict h) @ code t dict

```

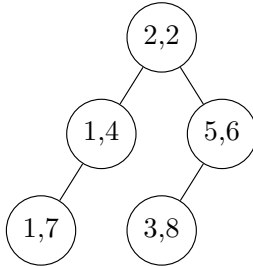
III Arbretas

1.

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

2. La fonction `shuffle t` permute 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 subi une permutation aléatoires.

3.



4. Soit $n \in \mathbb{N}$, le nombre d'éléments de l'arbretas, on note $e_1, \dots, e_n \in \mathbb{N}^2$ ces éléments. Montrons par récurrence forte sur n l'unicité de l'arbretas.

On note $P(n)$ l'assertion "Un arbretas à n éléments distincts est unique"

Initialisation : Si $n = 1$, alors l'arbretas est évidemment unique.

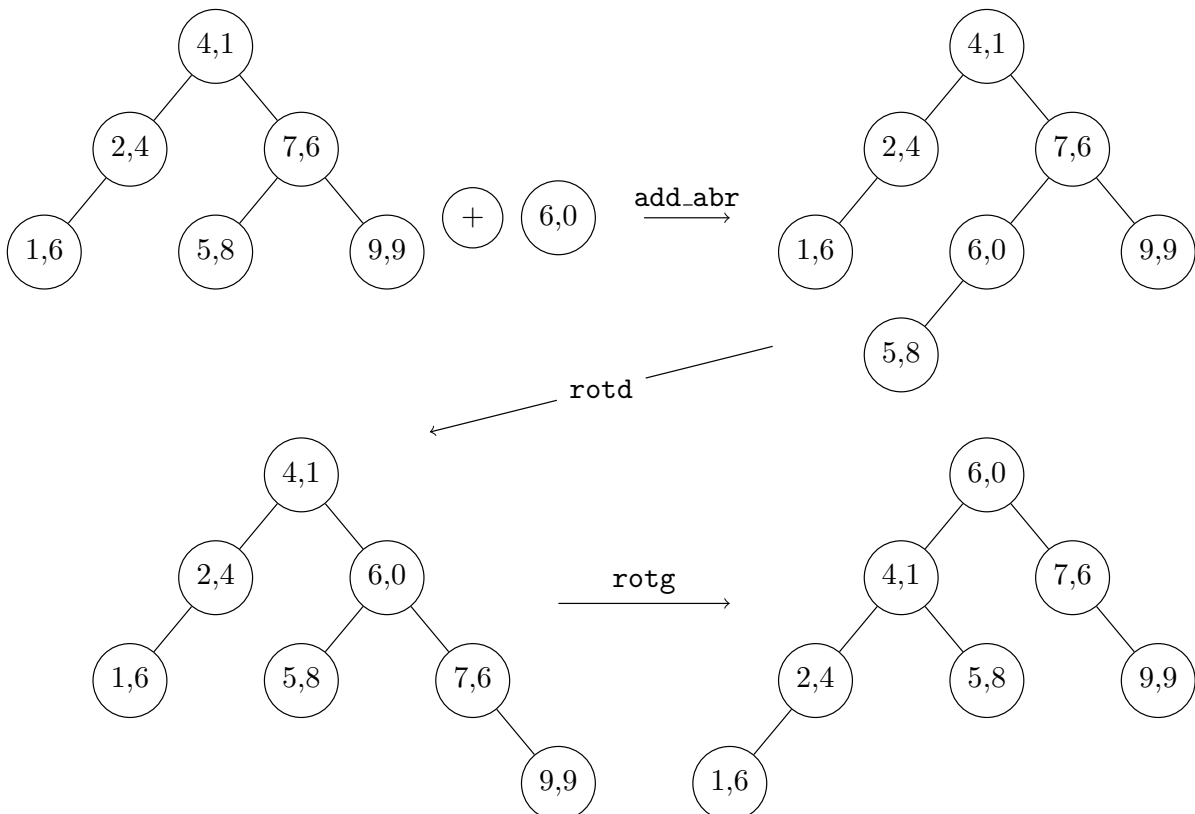
Hérédité : Soit $n \in \mathbb{N}$, supposons $P(n), P(n-1), \dots, P(1)$, montrons $P(n+1)$: On choisi i tel que $se_i = (x, p)$, alors p est la priorité minimum, alors par propriété de l'arbretas, e_i est la racine, on peut alors creer les sous-arbres gauche et droite, de taille au plus n qui sont donc uniques par *hypothèse de récurrence*.

D'où l'unicité.

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
    ;;

```