Exercice 1 :

$f(n) \in O(g(n))$: $f(n)$ est majoré par $g(n)$ à partir d'un certain n_0 $(f(n) \leq c \cdot g(n), n \geq n_0)$
$\exists\, c$ et n_0

$f(n) \in \Omega(g(n))$: $f(n)$ est minoré par $g(n)$ à partir d'un certain n_0 $(f(n) \geq c \cdot g(n), n \geq n_0)$
$\exists\, c$ et n_0

$f(n) \in \Theta(g(n))$: ssi $f(n) \in O(g(n))$ et $f(n) \in \Omega(g(n))$

↳ ex : $\Theta(n) = \{n, 2n, n+2, \underbrace{n + \log_2(n)}_{\leq 2n}\}$ ($\log_2(n) \leq n$)

3) $\log_2(\log_2(n)) \ll \log_{10}(n) \ll \log_2(n) \ll \sqrt{n} \ll n \ll n\log(n) \ll n\sqrt{n} \ll n^2 \ll n^3 \ll 2^n \ll 3^n \ll n! \ll n^n$

4)
- ✗ n et n^2
- ✗ n^2 et n
- ✓✓✓
- ✓✗✗
- ✗✗✓
- ✓✓✓
- ✓
- ✗
- $f(n) \leq c_1 h(n)$, $g(n) \leq c_2 h(n)$ $\Rightarrow f(n) + g(n) \leq (c_1 + c_2) h(n)$ ✓
- $f(n) + g(n) \in O(h(n))$ ssi $f(n), g(n)$ et $h(n)$ sont positive ✓
  $f(n) + g(n) \leq c \cdot h(n)$
- ✗
- n et $\log_2(n)$ n ✗
  $f(n)$ $g(n)$ $h(n)$

Exercice 2 :

1) Peut importe comment on coupe ça mène au même.

2) entrée : une tablette T

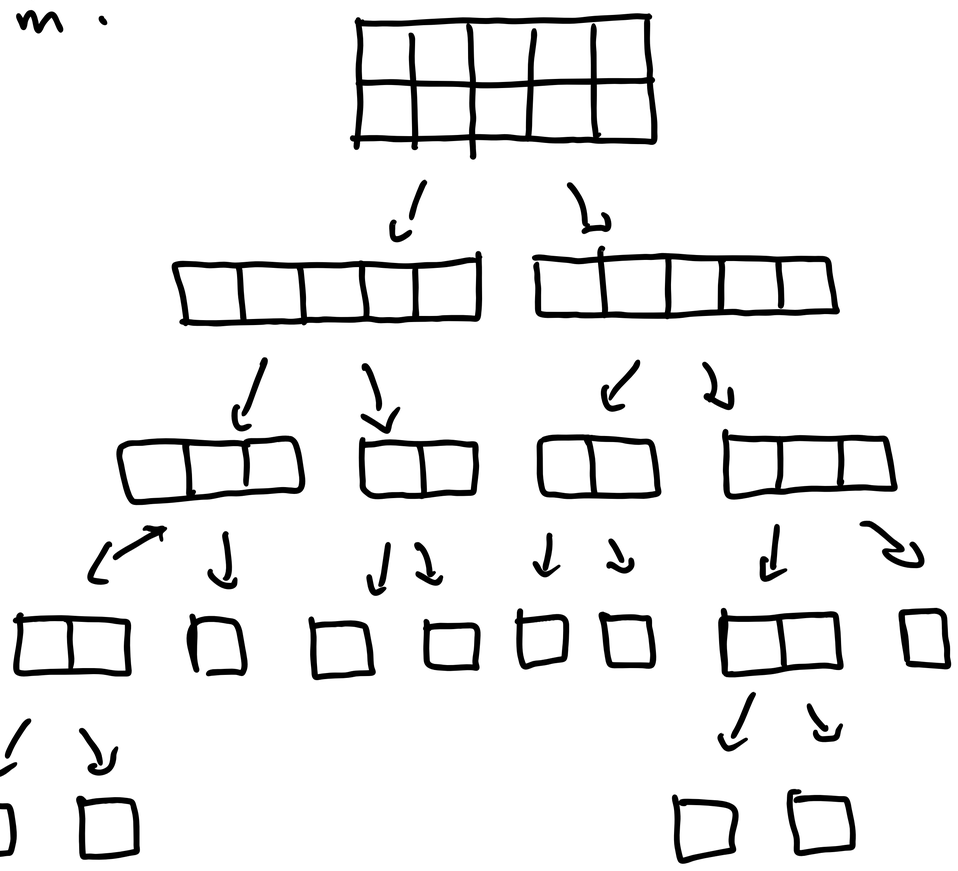
morceaux ← {T}

Tant que morceaux contient un morceau non carré on fait

prendre un client E non carré de morceaux

et de couper E en T_1 et T_2

morceaux ← (morceaux \ {E}) ∪ {T_1, T_2}



3) Donc optimal

Exercice 3 :

[10, 3, 5, 25, 50, 1]

[3,5,10] // (return)

[10,3,5]

[10] // (return) \ [3,5]

[10] [3,5]

[3] // (return) \ [5] (return)

[3] [5]

\ [1,25,50] (return)

[25,50,1]

.

.

.

.

```
def trifusion(t):
    if len(t)<=1:
        return t
    else:
        moitie = len(t)//2
        t1 = trifusion(t[0:moitie])
        t2 = trifusion(t[moitie:])
        return(fusionner(t1, t2))
```

```
def fusionner(t1, t2):
    t = []
    while t1 and t2:
        if (t1[0] < t2[0]):
            t.append(t1.pop(0))
        else:
            t.append(t2.pop(0))
    return t+t1+t2
```

Exercice 2 :

1) Peut importe comment on coupe ça mène au même.

2) entrée : une tablette T

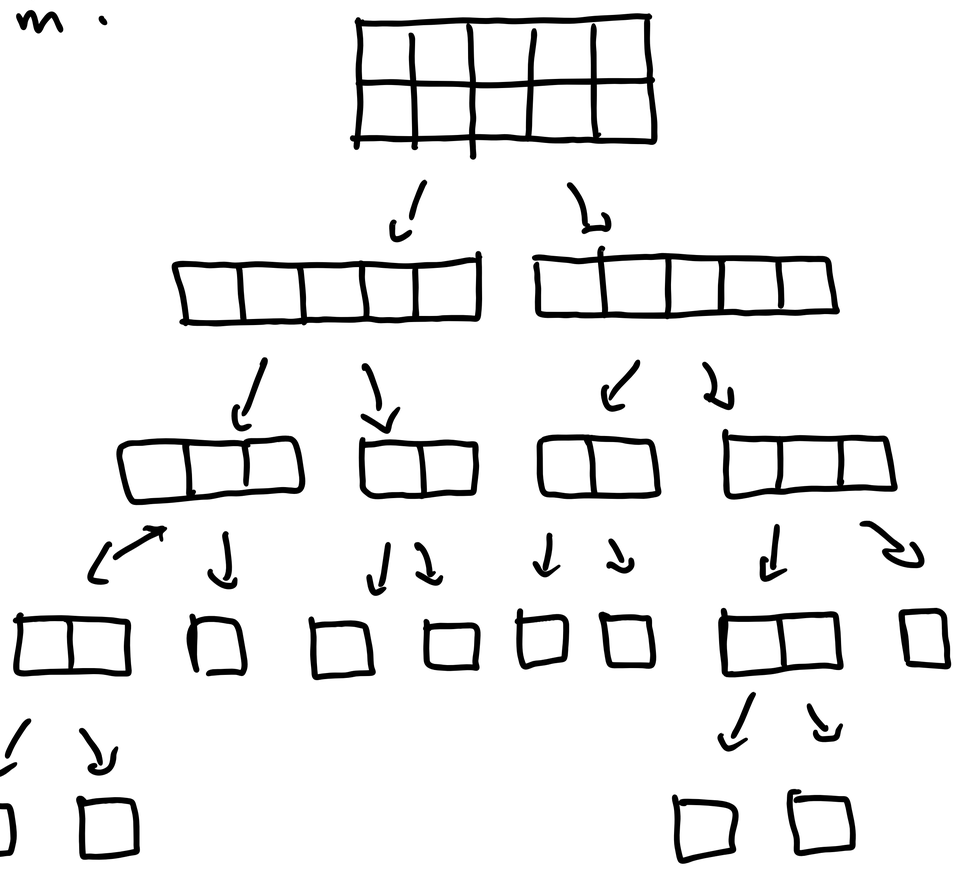
morceaux ← {T}

Tant que morceaux contient un morceau non carré on fait

prendre un client E non carré de morceaux

et de couper E en T_1 et T_2

morceaux ← (morceaux \ {E}) ∪ {T_1, T_2}



3) Donc optimal

Exercice 3 :

[10, 3, 5, 25, 50, 1]

[3,5,10] // (return)

[10,3,5]

[10] // (return) \ [3,5]

[10] [3,5]

[3] // (return) \ [5] (return)

[3] [5]

\ [1,25,50] (return)

[25,50,1]

.

.

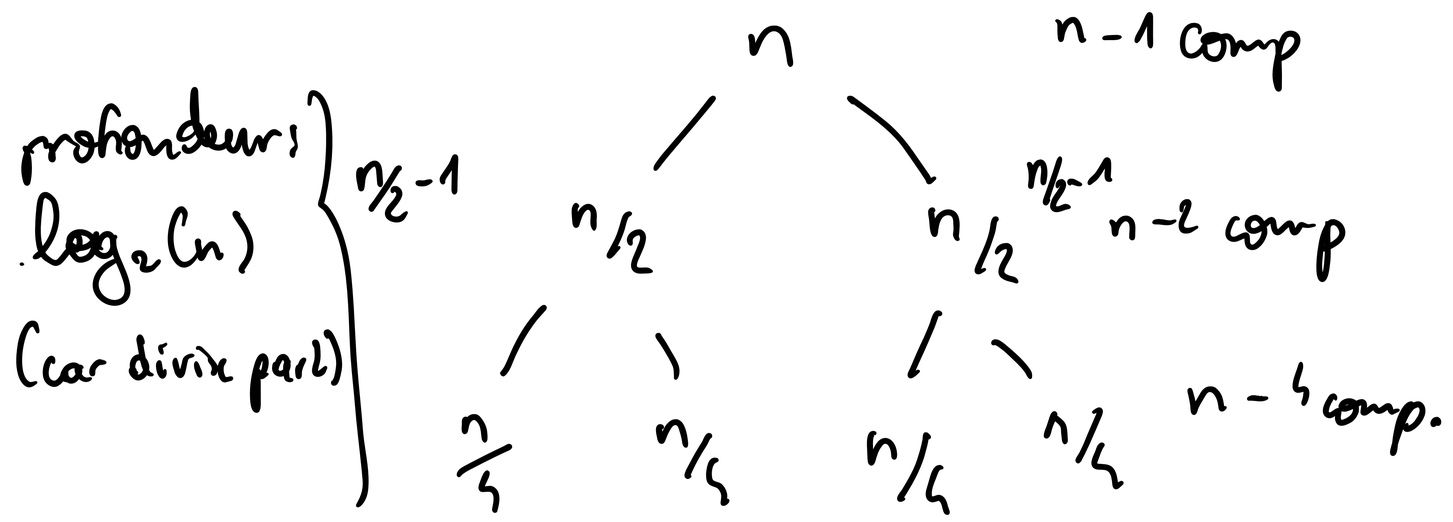
.

.

```
def trifusion(t):
    if len(t)<=1:
        return t
    else:
        moitie = len(t)//2
        t1 = trifusion(t[0:moitie])
        t2 = trifusion(t[moitie:])
        return(fusionner(t1, t2))
```

```
def fusionner(t1, t2):
    t = []
    while t1 and t2:
        if (t1[0] < t2[0]):
            t.append(t1.pop(0))
        else:
            t.append(t2.pop(0))
    return t+t1+t2
```

// complexité algo tri ⇝ comparaison (tri fusion :



2) pour se passer de mémoire auxiliaire utiliser des indices, mais prob fusion ?

Exercice 4 :

- prendre les chemins 1 par 1 les plus court -> algo glouton.
  ↳ pas efficace
- essayer tt tester.
  ↳ très long