Corrigé type

Exo 1

1. $T^{add}_{fonc}(n) = n+1$

n	T
1	$1 = 1$
1	$1+1 = 2$
2	$1+1+1 = 3$
3	$1+1+1+1 = 4$
4	$1+1+1+1+1 = 5$
⋮	
1	$1+1+\dots+1 = n$
0	$1+1+\dots+1 = n+1$

(2)

2) $T_{proc}(n) = ?$

avec $T^{add}_{fonc}(n) = n$.

(2)

i	
n	$1 + n \qquad n = T^{add}_{fonc}(n)$
n/4	$1 + n + 1 + n/4 = 1+1+n+n/4$
$n/4^2$	$1+1+1+n+n/4+n/4^2$
$n/4^3$	$1+1+1+1+n+n/4+n/4^2+n/4^3$
⋮	
0	$\underbrace{1+1+\dots+1}_{\log_4(n)\ \text{fois}} + n\left(1+\frac{1}{4}+\frac{1}{4^2}+\dots+\frac{1}{4^{\log_4(n)}}\right)$
	$= \sum_{j=1}^{\log_4(n)} 1 + n\sum_{j=0}^{\log_4(n)}\left(\frac{1}{4}\right)^j$

si $a = \log_4(n) \Rightarrow T_{proc}(n) = \frac{4}{3}\left(1-\left(\frac{1}{4}\right)^a\right)n + a.$

Exo 3 (05/05)

1) Les structures qui ne sont pas des arbres de recherche sont :

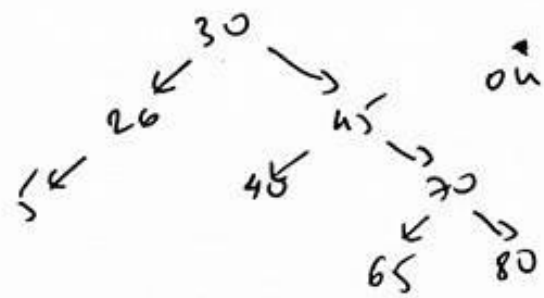
A1 car 45 est sup à son fils D : (1)

A2 car A2 est un arbre 3-aire. (1)

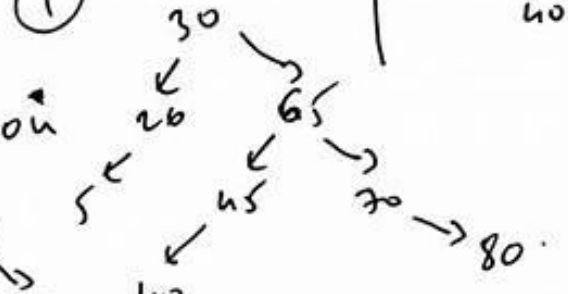
A3 car A3 n'est pas un arbre. (1)
"un noeud a deux pères !"

2).

A5 (1)

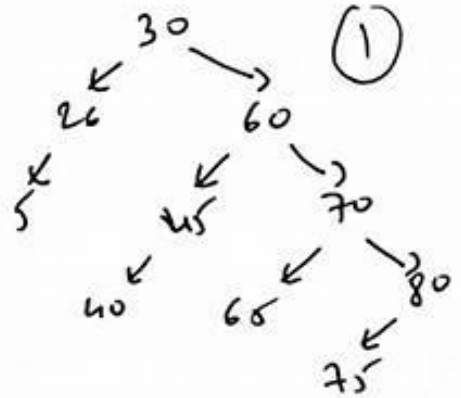


ou



3)

A6.



(1)

0/6

```
…java.util.Scanner
…ss Programme {

  public static void main (String[] args)
  {
    Scanner sc = new Scanner(System.in);
    CEnsemble e = new CEnsemble();

    introduire(e);
    afficher (e);

  }

static void afficher (CEnsemble e){
    int x ;
    boolean nonVide=true;

    while (nonVide){
        try{
            x= e.supp();

        }catch ( Evide a) {
            System.out.println(" fin affichage")
            nonVide = false;
        }
    }
}
```

(2)

```
static void introduire ( CEnsemble e) {
    boolean nonplein=true;
    boolean unEntiers;

    while (nonplein) {.
        try {
            boolean unEntiers = true;
            do
                try {
                    n= sc.nextInt () ;
                }catch (NumberFormatException a)
                { System.out.println("il faut un
                    entiers ");
                    enters = false;
                }
            while ( enters == false);
            e.ajout (n);
        }catch ( Eplein a) {.
            System.out.println(" l'ensemble
                est plein ");
            nonplein = false;
        }
    }
}

}
```

(2)

(2)

```
boolean existe (int n) {
boolean e1, e2, e3;
e1 = false; e2 = false; e3 = false;
if (n = this.cle) {return true; }
else {
  if (this.fils1 != null)
    { e1 = this.fils1.existe(n); }
  if (this.fils2 != null)
    { e2 = this.fils2.existe(n); }
  if (this.fils3 != null)
    { e3 = this.fils3.existe(n); }
  return e1||e2||e3;
  }
}
```

(2)

et (1) pts pour le constructeur.

```
int feuilles ( ) {
  int f1, f2, f3;
  f1 = 0; f2 = 0; f3 = 0;
  if (fils1 == null && fils2 == null && fils3 == null)
    {return 1; }
  else {
    if (fils1 != null)
      { f1 = fils1.feuille(); }
    if ( fils2 != null )
      { f2 = fils2.feuille(); }
    if ( fils3 != null )
      { f3 = fils3.feuille(); }
    return f1 + f2 + f3;
    }
}
```

(2)