```
// 1er appel: deb = 0, fin = taille-1
// algo récursif qui sit le paradigme diviser pour règner
Tri_rapide(tab: tableau, deb: entier, fin: entier)
donnée ref : tab : tableau à trier
données : deb, fin : indice de debut et fin  du tab à trier
Debut
        SI (deb<FIN) alors
                pivot <- partition (tab, deb, fin)
                Tri_rapide(tab, deb, pivot-1)
                Tri_rapide(tab, pivot+1, fin)
        FINSI
FIN
// 1er appel deb = 0, fin = taille-1
// algorithme itératif
partition (tab: tableau, deb: entier, fin: entier): entier
Donnée ref :  tab : tableau à partitionner de telle sorte à placer tos les elms
                        inf au pivot à gche et tous elmts sup au pivot à dte
Données :   deb, fin : indice debut et fin du tab a partitionner
Var loc :       pivot :indice du pivot
                        indexG, indexD : les deux index qui vont faire le balayage

Résultat : lindice pivot une fois il est bien placé
Debut
        pivot <- fin
        indexG <- deb
        indexD <- fin - 1
        TANTQUE (indexG<indexD) FAIRE
                TANTQUE (indexG<indexD) et (tab[indexG]<=tab[pivot]) faire
                        indexG++
                FINTANQUE
                TANTQE (indexG<indexD) et (tab[indexD])>=tab[pivot]) FAIRE
                        indexD--
                FINTANTQUE
                SI (indexG<indexD) ALORS
                        permuter(tab, indexG, indexD)
                FINSI
        FINTANTQUE
        // cas particulier si l'elemt pivot est le plus grand elmt du tab
        SI(indexG=pivot-1)et (tab[indexG]<=tab[pivot]) ALORS
                retourner pivot
        FINSI
        permuter(tab, indexG, pivot)
        retourner indexG
FIN
```

```
// algorithme récursif
triFusion(tab : tableau, taille : entier)
Donnee ref : tab : tableau a trier
Donnee : taille : taille du tableau ( en java on peut virer ce parametre et utiliser tab.e)
Var loc : tab_g : partie gauche de tab
                  tab_d : partie droite de tab
DEBUT
        SI (taille = 1) ALORS retourner
        FINSI
        //  etape 1 : création de deux sous-tableaux
        tab_g <- copie (tab, 0, taille/2)
        tab_d <- cope (tab, taille/2, taille)
        // Etape2 : appel récursif sur les sous-tab
        triFusion(tab_g, taille/2)
        triFusion(tab_d, taille -taille/2)
        fusion (tab_g, tab_d, tab)
FIN
// algorithme itératif qui va fusionner les deux sous-tab triés tabG et tabD dans le tableau tab
fusion ( tabG, tabD, tab : tableaux)
Donnée ref : tab
Données : tabG, tabD
VAr loc : indexG, indexD :les index pour manipuler les deux sous-tab
DEBUT
        indexG <- 0
        indexD <- 0
        Pour i allant de 0 à tab.length-1 faire
        //cas extreme, quand un des deux sous-tab est déjà completement copié
                SI (indexD=tabD.length) OU ((indexG< tabG.length) ET (tabG[indexG]<
                tabD[indexD])) ALORS
                        tab[i]<- tabG[indexG]
                        indexG++
                SINON
                        tab[i]<- tabD[indexD]
                        indexD++
                FINSI
        FINPOUR
FIN

Copie(tab : tableau, deb, fin : entiers) : tab
Données : tab : tableau origine
            Deb, fin : indice deb et fin de la copie
Var loc :    tab_copie : le nouveau tableau qui représente la copie
Resultat : tab_copie
DEBUT
        Tab_copie <- allouer(f-d)
        Pour i allant de deb à fin-1 faire
                Tab_copie[i-d]<- tab[i]
        FinPour
        Retourner tab_copie
FIN
```