Correction TD4 Atelier de Programmation : Récursivité

A.

```
void AjoutElemP (Liste** TL, Liste *ne) {
   Liste * T;
        if ( *TL == NULL)
              *TL = ne;
            else {
                    if ( ne->val <= (*TL)->val ) {
                       ne->suivant = *TL;
                       *TL=ne;
                      }
                      else {
                       AjoutElemP(&(*TL)->suivant, ne);
                    }
            }
    }
```

B.

```
void ProceduracreeRecursif( int n, Liste **t) {
        *t=NULL;
        if (n>0)  {
           *t=malloc(sizeof(Liste));
           (*t)->val=rand()%100;
           ProceduracreeRecursif(n-1, &(*t)->suivant);
        }
```

C.

```
void viderliste ( Liste ** tete) {
   Liste *p;
   if (*tete !=NULL) {
      if( (*tete)->suivant ==NULL) {
         free(*tete);*tete=NULL;
      }
      else {
         p=*tete;
         *tete=(*tete)->suivant;
         free(p);
         viderliste(tete);
      }
   }
}
```

D.

```
void AfficherListeRecursif(Liste *t) {
   if ( t) {
       printf("%d | ", t->val);
       AfficherListeRecursif(t->suivant);
   }
}
```

E.

```
int RechercheLtriee( Liste *TL, int X) {
   int trouve = 0;
         if (TL !=NULL ) {
                  if ( TL->val == X)
                           trouve =1;
                       else {
              if (TL->val < X)
                  trouve = RechercheLtriee (TL->suivant, X);
                       }
         }
         return (trouve);
}
```

F.

```
Liste * Supprimelem(  Liste *tete, int X) {
         Liste *P;
         if (tete !=NULL ) {
                  if ( tete->val == X) {
                       P = tete;
                       tete = tete->suivant;
                       free(P);
                  }
                  else {
           if ( tete->val < X)
           tete->suivant= Supprimelem ( tete->suivant, X);
       }
         }
         return tete;
}
```

G.

```
Liste * Copier (Liste *TL ) {
   Liste * ne;
   ne = NULL;
        if (TL != NULL) {
                ne = malloc(sizeof(Liste));
                ne->val= TL->val;
                TL=TL->suivant;
                ne->suivant = Copier( TL);
                //ne->suivant = Copier( TL->suivant);
        }
        return ne;
}
```

H.

```
int egalListe ( Liste * t1, Liste * t2) {
   int eg=1;
   if ((t1!=NULL) && (t2!=NULL)) {
         if (t1->val != t2->val) {
            eg = 0;  printf (" \nt1 %d t2 %d", t1->val, t2->val);
            }
            else
               eg = egalListe(t1->suivant,t2->suivant);
   }
     else {
       if (( t1 && !t2) || (t2 && !t1))
            eg=0;
   }
   return eg;
}
```