TD1

EXO 1 :

Instruction	Nbr Op	Nbr exé	Coût
Lire(n)	1	1	1
Lire(A[j])	1	n+1	n+1
Lire(x)	1	1	1
Poly←polynome(A,x,n)	2	1	2
Poly←T[0])	1	1	1
I←1	1	1	1
I<=n	1	n+1	n+1
Poly←poly T[i]*puissance(x,i)	4	n	4n
I←1	1	n	n
Puiss←1	1	n	n
i<=p	1	2+…+n+(n+1)	$n^2/2+3n/2$
Puis←puiss*x	2	1+…+n	$2(n^2/2+n/2)= n^2+n$
i←i+1	2	1+…+n	$2(n^2/2+n/2)= n^2+n$
Puissance->puiss	1	n	n
i←i+1	2	n	2n
Polynome←poly	1	1	1
Ecrire(p)	1	1	1

Le coût total $C(n)= 5n^2/2+29n/2+10 <= 5n^2$. On prend k= 27 et $g(n)= n^2$, alors la complexité de l'algorithme est en $O(n^2)$.

TD2

Exo 1 : Tri par fusion

1- Pour trier un tableau de taille n, on effectue log(n) divisions (jusqu'à avoir des tableaux de taille 1). A chaque étape i (1<= i <=log(n)), on fusionne 2^i tableaux dont le coût est O(n) (le coût de chaque étape est O(n)). Alors le coût total est O(nlog(n)).

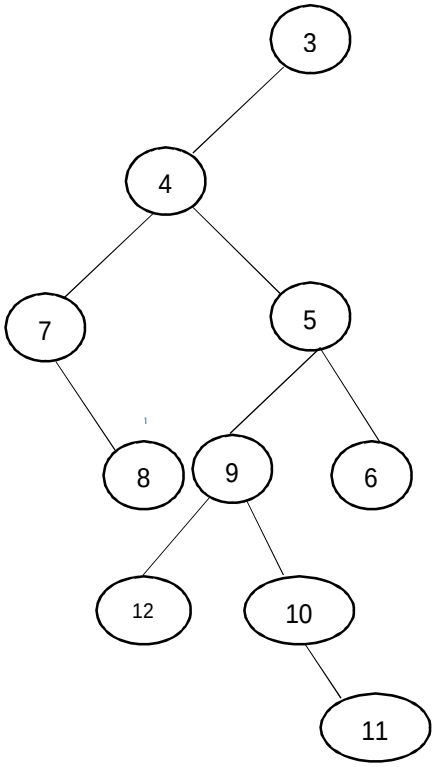
Exo 2 : Tri Rapide

1- De même, pour trier un tableau de taille n, on effectue log(n) divisions (jusqu'à avoir des tableaux de taille 1). A chaque étape i (1<= i <=log(n)), on partitionne un tableau de 2^{i-1} cases (n, n/2, …, 1) tableaux dont le coût est O(n) (le coût de chaque étape est O(n)). Alors le coût total est O(nlog(n)).

2- Le tri rapide est plus efficace que le tri par fusion. Car dans le tri par fusion, à chaque étape (log(n) étapes) on prend toutes les cases (n) ; par contre dans le tri rapide, à chaque étape i on réduit le nombre de case de 2^{i-1}.

3- Conclusion : Les algorithmes de même complexité n'ont pas toujours la même efficacité.

TD 3



Exo 1 :

1- H(a) =3
2- D(a)=3

3- Transformation de l'arbre en un arbre binaire.

4- a- Préfixe(RGD) = 3 4 7 8 5 9 12 10 11 6
b- Infixe(GRD) = 7 8 4 12 9 10 11 5 6 4 3
c- Postfixe(GDR) = 8 7 12 11 10 9 6 5 4 3

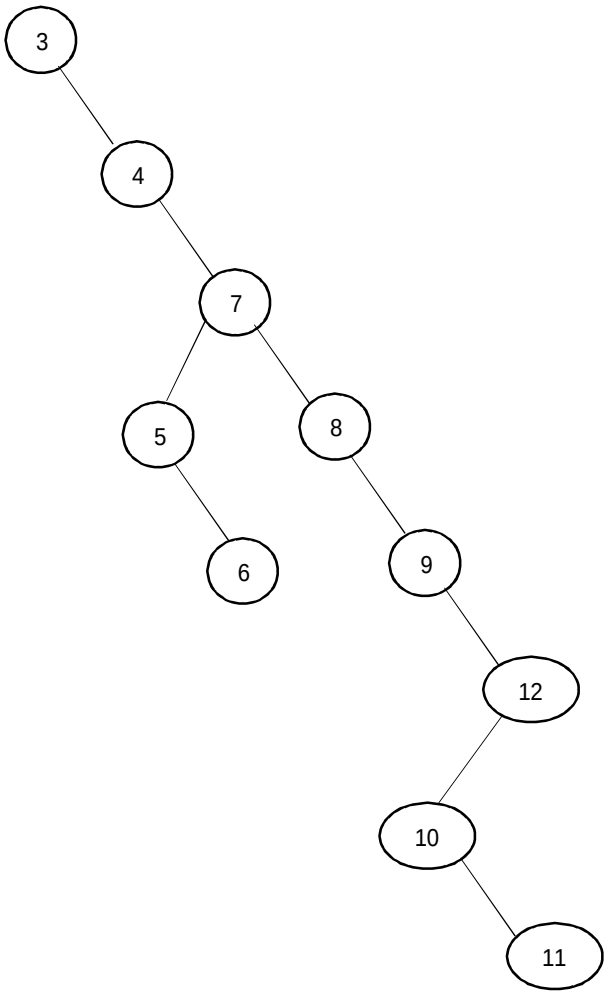
5-

```
procédure listefeuilles (a:arbre; var l: arbre);
var q:arbre;
begin
        if a‡nil then
        begin
                if  ((a^.fg=nil) and (a^.fd=nil)) then
                begin
                        new(q); q^.val:=a^.val;q^.suiv:=nil;
                        q^.suiv:=l;
                        l:=q;
                end
                else
                begin
                        listefeuilles(a^.fg,l);
                        listefeuilles(a^.fd,l);
                end;
        end;
end;
```

6-

```
fonction doubleFils(a:arbre):boolean;
début
        si  ((a^.fg=nil) and (a^.fd=nil)) then
        doubleFils ←true
        sinon si ((a^.fg‡nil) and (a^.fd‡nil)) then
                si (a^.val) > (2*(a^.fg^.val+a^.fd^.val)
                        doubleFils ←( doubleFils (a^.fg)) and (doubleFils (a^.fd)))
                sinon
                        doubleFils ←faux ;
                finsi
        sinon si (a^.fg‡nil) then
                si (a^.val) > (2*(a^.fg^.val)
                        doubleFils ←( doubleFils (a^.fg))
                sinon
                        doubleFils ←faux ;
                finsi
        sinon si (a^.fd‡nil) then
                si (a^.val) > (2*(a^.fd^.val)
                        doubleFils ←( doubleFils (a^.fd))
                sinon
                        doubleFils ←faux ;
                finsi
        finsi
fin;
```

7- L'ABR obtenu du parcours Préfixe :



8- Créer un ABR à partir d'une liste l :

```
procédure créerABR(var a : arbre ; l : arbre)
var q : arbre ;
début
        Tq l ‡ nil faire
                q←l ;
                ajouter(l,q) ;
        finTq
fin ;

fonction  ajouter(var a:  arbre;  q  : arbre) ;
début
   si a=nil alors
        a←q
   sinon
        si val> a^.info alors
                si a^.FD=nil alors
                        a^.FD←q ;
                sinon
                        inserrer(a^.FD ; q) ;
                finSi
        sinon
                si a^.Fg=nil alors
                        a^.FG←q ;
                sinon
                        inserrer(a^.FG ; q) ;
                finSi ;
        finSi ;
   finSi
fin ;
```

Exo 2 :

1 -Définition des types :

```
Type   arbre = ^personne ;
        personne=enregistrement
                num : entier ;
                nom, prénom  : chaîne [30] ;
                sexe : caractère ;
                age :entier ;
                fg : arbre;
                fd : arbre;
        fin ;
```

2- liste des descendants d'une personne donnée.

```
procedure listeDesc(a:arbre; var l:arbre; num:integer; trv:boolean);
var q:arbre;
début
        si a‡nil alors
        début
                si  trv alors
                début
                        new(q); q^.val←a^.num; //q^.suiv←nil;
                        q^.suiv←l;
                        l←q;
                        listeDesc(a^.fg, l,num, trv);
                        listeDesc(a^.fd, l, num, trv);
                fin
                sinon
                si a^.num=num alors
                début
                        trv←vrai;
                        listeDesc(a^.fg, l,num, trv);
                        listeDesc(a^.fd, l, num, trv);
                fin
                sinon
                début
                        listeDesc(a^.fg, l,num, trv);
                        listeDesc(a^.fd, l, num, trv);
                fin;
        fin;
end;
```

3- Les personnes ayant gardé le même nom que la racine.

```
procedure affichePers(a:arbre; num:entier ; nom : chaîne);
début
        si a‡nil alors
        début
                si (a^.num‡num) et a^.nom = nom alors
                début
                        écrire(a^.num);
                        affichePers(a^.fg, num, nom);
                        affichePers(a^.fd, num, nom);
                fin
                sinon
                début
                        affichePers(a^.fg, num, nom);
                        affichePers(a^.fd, num, nom);
                fin;
        fin;
fin;
```