Examen du 1er Semestre (Année Universitaire 2025-2026)

Niveau : 2ème année Licence en Informatique Date : 18/01/2026

Module : Algorithmique et Structures de Données III (ASDIII) Durée: 02 Heures

Corrigé type

Note : ..... / 20

Exercice N°1 : (8 pts) La complexité algorithmique

Par le calcul de la complexité algorithmique nous voulons principalement :

- ✓ Comparer l'efficacité d'algorithmes résolvant le même problème.
- ✓ Vérifier si un algorithme résolvant un problème peut donner une réponse dans un temps raisonnable.

Soit les deux programmes suivants calculant la somme des n premiers entiers naturels :

Le programme P1

```
#include <stdio.h>
int main ()
{
    int i, n;
    long S=0, Nbriteration=0;….. 2
    printf("Entre un entier n :");..... 1
    scanf("%d", &n);……1
    for ( i=1; i<=n; i=i+1) ……1+3n
    {
            S = S+ i;…..2n
            Nbriteration = Nbriteration +1;……2n
    }
     printf("\n la somme est : %ld ", S) ;....1
     printf("\n le nombre d'iterations est : %ld ", Nbriteration);…..1
}
```

Réponse 1/

Nombre opérations = 7n+7 .....(*1 point*)

Complexité O(n) = n (linéaire) .....(*0.5 point*)

Le programme P2

```
#include <stdio.h>
int main ()
{
        int i, j, n;
        long S=0, Nbriteration=0;…..2
        printf("Entre un entier n :");……1
        scanf("%d", &n);……1
          for ( i=1; i<=n; i=i+1)……1+3n
          {
           for ( j=1; j<=i; j++)…. n+3n(n+1)/2
            {    S = S+ 1;…. 2n(n+1)/2
                 Nbriteration = Nbriteration +1; …. 2n(n+1)/2
            }
          }
           printf("\n la somme est : %ld ", S) ;…..1
           printf("\n le nombre d'iterations est : %ld ",
        Nbriteration);……1
}
```

Réponse 2/

Nombre opérations = $7+6n+(3n/2)+(7n^2/2)$ ....(*2 points*)

Complexité O(n) = n^2 (quadratique) ……(*0,5 point*)

Réponse 3/

```
#include <stdio.h>
int main ()
{
  int n;
  long S ;
  printf("Entre un entier n :");..... 1
  scanf("%d", &n);……1
  S=n*(n+1)/2 ; …… 4 .……(2 points)
  printf("\n la somme est : %ld ", S) ;....1
}
```

Nombre opérations = 7 .....(*1point*)

Complexité O(n) = 7 (constante) …….(*1point*)

Exercice N°2 : les arbres (6 pts)

Soit le programme suivant :

```
struct Noeud{
        int valeur;
        struct Noeud *filsg;
        struct Noeud *filsd;
        } ;
struct Arbre{
        struct Noeud *racine;
        };
struct Noeud *CreerNoeud(int val, struct Noeud
*filsg, struct Noeud *filsd){
        struct Noeud *No;
        No =malloc(sizeof(struct Noeud));
        No->valeur=val;
        No->filsg=filsg;
        No->filsd=filsd;
        return(No);
}
```

```
struct Noeud *CreerFeuille(int val){
        struct Noeud *Fe;
        Fe = malloc(sizeof(struct Noeud));
        Fe->valeur=val;
        Fe ->filsg=NULL;
        Fe ->filsd=NULL;
        return(Fe);
}
struct Arbre *CreerAbArbre(){
        struct Arbre *Ar;
        Ar=malloc(sizeof(struct Arbre));
        Ar -> racine=NULL;
        return(Ar);
}
```

Questions :

1/ Quelle est la différence entre la fonction CreerFeuille et la fonction CreerNoeud ?

Réponse 1/ ***(1 point)***

La fonction CreerFeuille( )	La fonction CreerNoeud( )
1- Reçoit en entrée un seul paramètre (val)	**1-** Reçoit en entrée trois paramètres (val, *filsg,*filsd)
2- Réserve en MC un emplacement pour une feuille	**2-** Réserve en MC un emplacement pour un nœud
3- Le fils gauche = NULL, le fils droit = NULL	**3-** Le fils gauche = filsg, le fils droit = filsd
4- Renvoi en sortie la feuille qui a été créée.	**4-** Renvoi en sortie le noeud qui a été créée.

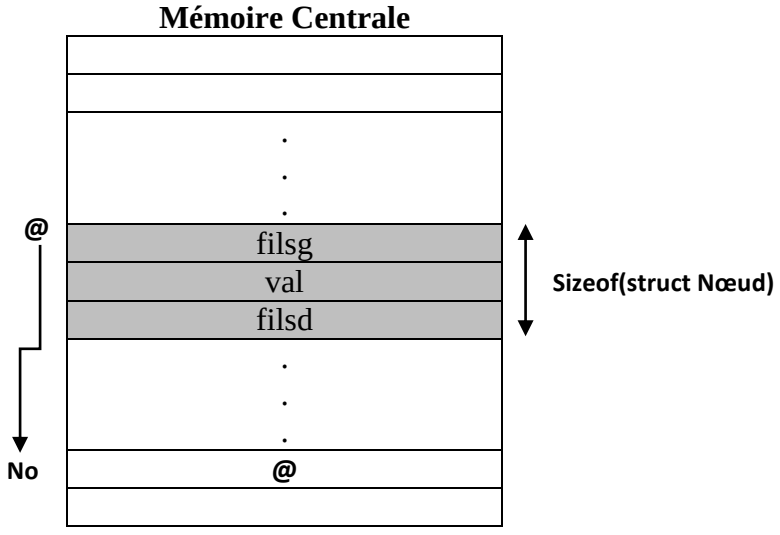
2/ Expliquer le principe de fonctionnement de l'instruction : No =malloc(sizeof(struct Noeud));

Réponse 2/ ***…. (1 point)***

La fonction malloc va réserver en mémoire centrale un emplacement pour un nouveau nœud dont sa taille est calculée par sizeof(struct Nœud). L'adresse en MC de cet emplacement sera stockée dans le pointeur No.

3/ Établir le schéma (dessin) correspondant en mémoire centrale de la fonction CreerNoeud.

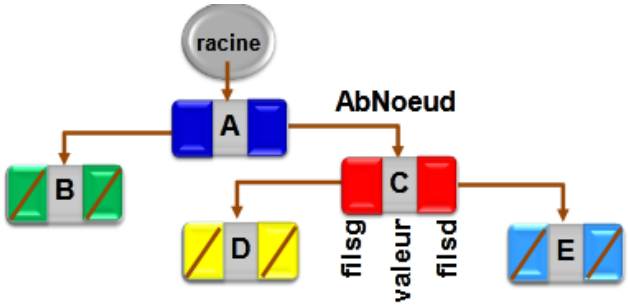
Réponse 3/ ***…. (1 point)***



4/ soit l'arbre représenté par la structure suivante :

Réponse 4/ … *(1.5 point)*

```
int main ( )
{
    struct Nœud *A, *B, *C, *D, *E ;
    struct Arbre ARB;
      D=CreerFeuille('D',NULL,NULL) ;
      E=CreerFeuille('E',NULL,NULL) ;
      C=CreerNoeud('C',D,E) ;
      B=CreerFeuille('B',NULL,NULL) ;
      A=CreerNoeud('C',B,C) ;
      ARB.racine=A;
}
```



Écrire la fonction main() qui permet d'appeler les fonctions précédentes afin de créer cette structure.

5/ affichez les résultats des parcours de cet arbre : infixe, prefixe, postfixe

Réponse 5/

infixe : B, A, D, C, E … ***(0.5 point)***

prefixe : A, B, C, D, E … ***(0.5 point)***

postfixe : B, D, E, C, A … ***(0.5 point)***

Exercice N°3 : **(6 pts)**

Veuillez répondre par vrai ou faux

Remarque : réponse correcte = +1/ réponse erronée = - 1 /pas de réponse = 0

1. La complexité temporelle permet de mesurer l'efficacité des algorithmes. (***VRAI***)
2. Un graphe est une structure de données non linéaire statique. (***FAUX***)
3. Un arbre est une structure de données hiérarchique dynamique. (***VRAI***)
4. La complexité temporelle de la recherche dichotomique est meilleure que celle de la recherche séquentielle. (***VRAI***)
5. La représentation par matrice d'adjacence d'un graphe permet de garantir une meilleure économie de l'utilisation de la mémoire centrale. (***FAUX***)
6. La complexité logarithmique est meilleure que la complexité racine carrée. (***VRAI***)

Bon courage