1 Chapitre2

Exercice2.1

Exercice2.2

type nœud = **enregistrement** élément : **entier** ; fils-gauche, fils-droit : -1..n **fin-enregistre ;**
const max = 200 **;**
var B : **tableau** [1..max] de nœud ;
libre : 1 .. max ;

1) **Procédure d'insertion d'un entier dans B.**

```
procédure insérer (x : entier ; var racine : -1 .. max) ;
var     k, parent : -1..max ;
début
        k := racine ;
        tant que (k <> -1) faire
                début  parent := k ;
                       si x > B[k].élément alors k := B[k].fils-droit
                                           sinon k := B[k].fils-gauche ;
                fin ;
        B[libre].élément := x ;
        B[libre].fils-gauche := -1 ;
        B[libre].fils-droit := -1 ;
        si (racine <> -1) alors
                            si (x > B[parent].élément) alors B[parent].fils-droit := libre
                                           sinon B[parent].fils-gauche := libre
                       sinon racine := libre ;
        si (libre < max) alors libre := libre + 1
                            sinon écrire ('arbre plein')
fin ;

        programme principal
        début
                libre := 1 ;
                racine := 1 ;
                …..
                Insertion (47, racine) ;
        fin.
```

2) **Complexité de la procédure insertion**

Au pire des cas, l'élément sera inséré au niveau d'une feuille de l'arbre. Le nombre d'itérations de la boucle de l'algorithme est égal au nombre de nœud se trouvant sur la chaine allant de la racine vers la feuille. Ce nombre correspond à la profondeur de l'arbre. Il s'agit donc de mesurer la profondeur par rapport au nombre total de nœud. Si l'arbre est complet, le nombre de nœud dans un niveau de l'arbre est une puissance de 2. En effet,

Au niveau 0 (la racine), il y a 2^0 nœud
Au niveau 1, il y a 2^1 nœuds
Au niveau 2, il y a 2^2 nœuds
….
Au niveau p, il y a 2^p nœuds (p étant la profondeur de l'arbre)
Le nombre total de nœuds de l'arbre est donc égal à $\sum_{i=0}^{i=p} 2^i$. Il s'agit d'une somme d'une somme géométrique qui est égale à $\frac{1-2^{p+1}}{1-2} = 2^{p+1} - 1$.
$2^{p+1} - 1 = n \rightarrow \quad 2^{p+1} = n + 1 \quad \rightarrow \quad p + 1 = \log_2(n + 1) \rightarrow$
$p = \log_2(n + 1) - 1$
La complexité est donc en $O(\log_2 n)$.

Exercice2.3

1) Les différentes itérations sont les suivantes :

0	1	2	3	4	5	6	7	8	9	10
1	0	0	3	1	5	1	7	1	9	1
2	0	0	0	1	5	1	7	1	1	1
3	0	0	0	1	1	1	7	1	1	1
4	0	0	0	1	1	1	1	1	1	1

2) L'algorithme est le suivant :

```
var i, j, r: integer ;

écrire ('les nombres premiers sont);
Tab[1] :=0;

début
        pour i := 2 à n faire
        début
                si tab[i] <> 1 alors début
                                        tab[i] := 0;
                                        écrire (i);
                                     fin;
                pour j := i+1 à n faire
                        si (tab[j] mod i = 0) alors tab[j] :=1;
        fin;
fin;
```

3) Il existe deux boucles dans l'algorithme. La boucle externe s'exécute de 2 à n donc (n-1) fois. La boucle interne par contre s'exécute un nombre de fois qui dépend de i qui varie de n-2 à 1. La complexité est donc de : $\sum_1^{n-2} i = \frac{(n-2)(n-1)}{2}$. Elle est en $O(n^2)$.

4) Le programme assembleur :

```
MOV        #10, R4
```

```
        MOV     #2, R0
L5:     SUB     R4, R0
        JZ      R0, Fin
        ADD     R4, R0
        SUB     #1, T[R0]
        JZ      T[R0], L0
        MOV     #0, T[R0]
        OUTPUT  R0
L0:     MOV     R0, R1
L3:     ADD     #1, R1
        MOV     T[R1], R2
        MOV     T[R1], R3
        DIV     R0, R2
        IDIV    R0, R3
        SUB     R2, R3
        JZ      R3, L1
        JMP     L2
L1:     MOV     #0, T[R1]
L2:     SUB     R4, R1
        JZ      R1, L4
        ADD     R4, R1
        JMP     L3
L4:     ADD     #1, R0
        JMP     L5
Fin :   STOP
T :     0
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
```

5) La complexité du programme est identique à celle de l'algorithme de la question 2, car le programme est constitué de deux boucles imbriquées fonctionnant de la même manière que les boucles de l'algorithme. Elle est par conséquent en $O(n^2)$

Exercice 2.7

1) L'algorithme de fusion de deux listes triées

a. En utilisant la structure de tableau

Chacune des deux listes est représentée par deux tableaux : élément et suivant. Soient tête1 et tête2 les indices respectifs des premiers éléments des listes. L'algorithme de fusion est comme suit :

```
algorithme fusion ;
entrée : tête1, tête2 :entier
sortie : tête3 : entier

const max = 200 ;
var p1, p2, p3, libre : entier ;
        T1, T2, T3 : tableau [1..max] de
                enregistrement élément : entier ; suivant : 1..max fin;
début
        p1 := tête1 ;
        p2 := tête2 ;
        libre := 1
        tant que (p1 ≠ -1) et (p2≠-1) faire
        début
                si (T1[p1].élément < T2[p2].élément) alors
                        début T3[libre].élément := T1[p1].élément ;
                                p1 := T1[p1].suivant ;
                        fin
                sinon début T3[libre].élément := T2[p2].élément ;
                                p2 := T2[p2].suivant ;
                        fin ;
                T3[libre].suivant := libre+1 ;
                Libre := libre+1 ;
        fin ;
        si (p1 = -1) alors
                tant que (p2 ≠ -1) faire
                début T3[libre].élément := T2[p2].élément ;
                        T3[libre].suivant := libre+1 ;
                        Libre := libre+1 ;
                fin  sinon
                tant que (p1 ≠ -1) faire
                début T3[libre].élément := T1[p1].élément ;
                        T3[libre].suivant := libre +1;
                        Libre := libre+1 ;
                fin ;
        si (libre = 1) alors tête3 := -1 sinon début
                                                T3[libre-1].suivant := -1 ;
                                                 tête3 := 1 ;
                                               fin ;
fin ;
```

b. En utilisant la structure dynamique de listes

```
algorithme fusion ;
entrée : tête1, tête2 : ↑p
sortie : tête3 : ↑p

var
        p : enregistrement élément : entier ; suivant : ↑p fin;
```

```
        p1, p2, p3, libre, prec : ↑p
début
        p1 := tête1 ;
        p2 := tête2 ;
        si  (p1 ≠ nil) et (p2≠ nil) alors
                début alloc (libre)
                        tête3 := libre
                fin sinon tête3 := nil ;
        tant que (p1 ≠ nil) et (p2≠ nil) faire
        début
                si (p1↑.élément < p2↑.élément) alors
                        début libre↑.élément := p1↑.élément ;
                                p1 := p1↑.suivant ;
                        fin
                sinon début libre↑.élément := p2↑.élément ;
                                p2 := p2↑.suivant ;
                        fin ;
                prec := libre ;
                alloc(libre) ;
                prec↑.suivant := libre ;
        fin ;
        si (p1 = nil) alors
                tant que (p2 ≠ nil) faire
                début libre↑.élément := p2↑.élément ;
                        prec := libre ;
                        alloc(libre) ;
                        prec↑.suivant := libre ;
                fin  sinon
                tant que (p1 ≠ nil) faire
                début libre↑.élément := p1↑.élément ;
                        prec := libre ;
                        alloc(libre) ;
                        prec↑.suivant := libre ;
                fin ;
        libérer (libre) ;
        prec↑.suivant := nil ;
fin ;
```

2) L'algorithme parcourt les deux listes fournies en entrée linéairement pour construire une troisième liste qui contient les éléments des deux listes. Si l1 et l2 sont les longueurs respectives des deux listes, la complexité de l'algorithme serait en O(l1+l2)

3) Pour représenter une liste chainée en assembleur, il faut réserver un espace mémoire contigu pour contenir les éléments de la liste en sachant qu'un élément est un enregistrement d'un entier et d'une adresse relative. Plus précisément, deux mots contigus seront nécessaires pour mettre l'entier et l'adresse de l'élément qui suit.

Exemple :
Soit l= {11, 45, 76} une liste, sa représentation en assembleur est la suivante :

10	11
11	12
12	45
13	14
14	76
15	-1

4) Bien sûr, seule la représentation de tableau convient pour l'assembleur. L'algorithme de fusion en assembleur à l'aide du jeu d'instructions donné en cours est comme suit:

	MOV	tête1 , R1	
	MOV	tête2 , R2	
	MOV	tête3, R3	
	MOV	R3, R0	/ libre := 1
	ADD	#1, R1	**tant que** (p1 ≠ -1) et (p2≠-1) **faire**
	JZ	R1, L0	**début**
	ADD	#1, R2	**si** (T1[p1].élément < T2[p2].élément) **alors**
	JZ	R2, L0	**début** T3[libre].élément := T1[p1].élément ;
	SUB	#1, R1	p1 := T1[p1].suivant ;
	SUB	#1, R2	**fin**
	SUB	(R2), (R1)	
	JGT	(R1), L1	
	ADD	(R2), (R1)	
	MOV	(R1), (R3)	
	ADD	#1, R1	
	MOV	(R1), R1	
	JMP	L2	
L1:	ADD	(R2), (R1)	
	MOV	(R2), (R3)	**sinon début** T3[libre].élément := T2[p2].élément ;
	ADD	#1, R2	p2 := T2[p2].suivant ;
	MOV	(R2), R2	**fin** ;
L2 :	ADD	#2, R0	T3[libre].suivant := libre+1 ;
	ADD	#1, R3	Libre := libre+1 ;
	MOV	R0, (R3)	**fin** ;
	ADD	#1, R3	
L0 :	JZ	(R1), L4	
L6 :	JZ	(R2), L3	
	JMP	L5	**si** (p1 = -1) **alors**
L3 :	SUB	#1, (R2)	**tant que** (p2 ≠ -1) **faire**
	MOV	(R2), (R3)	**début** T3[libre].élément := T2[p2].élément ;
	ADD	#1, R2	T3[libre].suivant := libre+1 ;
	MOV	(R2), R2	Libre := libre+1 ;
	ADD	#1, R3	**fin sinon**
	ADD	#2, R0	
	MOV	R0, (R3)	
	ADD	#1, R3	
	ADD	#1, (R2)	
	JMP	L6	

L4 :	SUB	#1, (R1)	
	MOV	(R1), (R3)	**tant que** (p1 $\neq$ -1) **faire**
	ADD	#1, R1	**début** T3[libre].élément := T1[p1].élément ;
	MOV	(R1), R1	T3[libre].suivant := libre +1;
	ADD	#1, R3	Libre := libre+1 ;
	ADD	#2, R0	**fin** ;
	MOV	R0, (R3)	
	ADD	#1, R3	
	ADD	#1, (R1)	
	JMP	L0	
L5 :	SUB	#1, (R1)	
	SUB	#1, (R2)	
	SUB	tête3, R0	**si** (libre = 1) **alors** tête3 := -1 **sinon début**
	JZ	R0, L7	
	SUB	#1, R3	
	MOV	#-1, (R3)	T3[libre-1].suivant := -1 ;
	MOV	tête3, R3	tête3 := 1 ;
	JMP	Fin	**fin** ;
L7 :	MOV	#-1, R3	
Fin :	STOP		**fin** ;
tête1 :	31		
	tête1+2		
	65		
	tête1+4		
	121		
	-1		
tête2 :	7		
	tête2+2		
	17		
	tête2+ 4		
	53		
	tête2+6		
	329		
	-1		
tête3 :			

On suppose que les listes fournies en entrée apparaissent à la fin du programme respectivement aux adresses tête1 et tête2. A la fin de l'exécution de ce programme, tête3 contient le début de la liste résultat de la fusion des deux listes données en entrée.

5) La complexité de ce programme est identique à celle de l'algorithme de 1) à une constante multiplicative près. Elle est donc en O(l1+l2).