TD1

Prof. Habiba Drias

Exercices

Exercice 1.1

Ecrire six algorithmes différents pour déterminer si un nombre entier est premier ou composé. Evaluer la complexité pour chacun des algorithmes proposés.

Exercice 1.2

Considérer 7 algorithmes A0, A1, …, A6 conçus pour résoudre un même problème de taille n. La complexité en temps de chacun de ces algorithmes est exprimée dans la table ci-dessous.

Algorithme	Complexité temporelle	Temps		
		n= 10	n= 100	n= 1000
A0	$O(1)$			
A1	$O(\log_2 n)$			
A2	$O(\sqrt{n})$			
A3	$O(n)$			
A4	$O(n^2)$			
A5	$O(n^3)$			
A6	$O(2^n)$			

1) Calculer les ordres de grandeurs suivantes en secondes :
    a. 1 heure
    b. 1 jour
    c. 1 semaine
    d. 1 mois
    e. 1 année
    f. 1 siècle
    g. 1 millénaire
2) En supposant qu'une unité de taille s'exécute en une milliseconde, calculer le temps nécessaire pour traiter des tailles respectivement égales à 10, 100 et 1000.
3) Répéter la question 2 avec une unité de temps égale à une microseconde
4) Que peut-on en conclure ?

Exercice 1.3

Considérer le théorème suivant sur le pgcd ou le plus grand commun diviseur :

pgcd(a,b) = pgcd(b, a mod b)

1) Utiliser le théorème pour écrire un algorithme de calcul itératif du pgcd de deux entiers a et b
2) Calculer la complexité en temps de l'algorithme
3) Ecrire la version récursive de l'algorithme trouvé en 1) et calculer sa complexité
4) Ecrire un algorithme pour calculer le ppcm de deux nombres a et b et calculer sa complexité

Exercice 1.4

Considérer l'alphabet {0,1}, un palindrome est un mot qui se présente sous le format ww^{-1}
Où w est un mot quelconque de l'alphabet et w^{-1} est l'image miroir de w.

1) Ecrire un algorithme qui reconnaît des palindromes et calculer sa complexité
2) Ecrire un programme assembleur pour reconnaitre des palindromes et calculer sa complexité
3) Concevoir une machine de Turing pour reconnaître les palindromes. Le résultat doit être égal à 1 si le mot en entrée est un palindrome, 0 sinon. Calculer sa complexité

Exercice 1.5

Considérer l'alphabet {0,1}, et le langage L = {(0/1)*, nb de 0 = nb de 1}.

1) Ecrire un algorithme qui reconnaît L, calculer sa complexité
2) Ecrire un programme assembleur, calculer sa complexité
3) Concevoir une machine de Turing pour reconnaître les mots de L. Le résultat doit être égal à 1 si le mot en entrée est correct, 0 sinon. Calculer sa complexité

Exercice 1.6

1) Concevoir une machine de Turing pour additionner 2 nombres binaires. Calculer sa complexité
2) Ecrire un algorithme correspondant à la machine de Turing de la première question. Calculer sa complexité
3) Ecrire un programme assembleur pour additionner 2 nombres binaires et calculer sa complexité

Exercice 1.7

Considérer un tableau constitué des n premiers nombres entiers (1, 2, …, n). Une méthode de détermination des nombres premiers inférieurs au sens large à n, consiste à considérer le nombre 2 qui est premier puis à éliminer tous les nombres multiples de 2 car ils ne sont pas premiers, ensuite itérer ce processus en continuant avec le nombre suivant non éliminé c'est-à-dire 3 jusqu'à traiter tous les entiers du tableau.

1) Illustrer chaque itération du procédé de calcul des nombres premiers décrit ci-dessus sur les 10 premiers nombres entiers

2) Ecrire un algorithme de calcul et d'impression des nombres premiers inférieurs au sens large à n. Il est recommandé par souci de simplification de l'algorithme de mettre le nombre à 0 s'il est premier et à 1 sinon, une fois qu'il est traité.

3) Calculer la complexité de l'algorithme.

4) Ecrire l'algorithme en assembleur à l'aide du jeu d'instructions donné en cours.

5) Calculer la complexité du programme assembleur