Université USTHB – Bab-Ezzouar Bab-Ezzouar, 11 Octobre 2017

Faculté de l’Electronique et de l’Informatique, Département de l’Informatique Année universitaire 2017/2018

1ère année Master Informatique, Semestre 1 Semestre 1

Module : Conception et Complexité des Algorithmes

-------------------------------------------------------------------------------------------------------------------------------------------------------

**Série de Travaux Pratiques n° 2 (TP n°2)**

**Algorithmes de Complexités temporelles**

**linéaire et racine carrée**

L’objet de ce TP est une étude expérimentale de 3 algorithmes du problème du test de la primalité d’un nombre entier naturel. Les 2 premiers algorithmes ont une complexité linéaire en n et le 3ème algorithme a une complexité en racine carrée . On utilise le langage de programmation C.

**Rappel : Un nombre entier naturel n est premier s’il n’a que 2 diviseurs : le nombre 1 et le nombre n lui-même**.

**Partie I (Algorithme 1 du test de la primalité)**

1- Développer un algorithme qui permet de déterminer si un nombre entier naturel n est premier (n>=2).

**Ind :** Utiliser la fonction modulo qui donne le reste de la division de n par i, i variant de 1 jusqu’à n.

2.1- Calculer les complexités temporelles en notation exacte et/ou en notation asymptotique de Landau O (Grand O) de cet algorithme au meilleur cas, notée f1(n), et au pire cas, notée f2(n).

2.2- Calculer la complexité spatiale en notation exacte et/ou en notation asymptotique de Landau O (Grand O) de cet algorithme notée s(n).

3- Développer le programme correspondant avec le langage C.

4- Vérifier par programme si les nombres n donnés dans le tableau ci-dessous (1.000.003, 2.000.003, …) sont premiers. On doit remarquer que comme les nombres n varient dans l’intervalle 1 million à environ 2 milliards, le nombre de tests (le reste de la division de n par i) varie aussi dans le même intervalle.

5- Mesurer les temps d’exécution T pour ces nombres n et compléter le tableau ci-dessous.

**Ind :** Pour mesurer le temps d’exécution d’un programme avec le langage C, on utilise les fonctions de gestion du temps qui sont fournies dans la bibliothèque time.h (inclure l’instruction : #include <time.h>).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | 1.000.003 | 2.000.003 | 4.000.037 | 8.000.009 | 16 .000.057 | 32.000.011 | 64.000.031 |
| T |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 128.000.003 | 256.000.001 | 512.000.009 | 1024.000.009 | 2048.000.011 |
| T |  |  |  |  |  |

6- Développer un programme de mesure du temps d’exécution du programme qui a en entrée les données de l’échantillon ci-dessus et en sortie les temps d’exécution. Les données et les mesures du temps sont à enregistrer dans des tableaux notés respectivement Tab1 et Tab2.

7- Représenter par un graphe, noté au choix soit Gf1(n) soit Gf2(n), les variations de la fonction de la complexité temporelle correspondant soit au meilleur cas f1(n) soit au pire cas f2(n) en fonction de n ; et par un autre graphe, noté GT(n), les variations du temps d'exécution T(n) en fonction de n. Utiliser pour cela un logiciel graphique tel que excel.

8- Interprétation des résultats :

8.a- Les mesures du temps obtenues correspondent-elles au meilleur cas ou au pire cas ?

//comparer le graphe des valeurs de l’échantillon avec les 2 autres graphes meilleur et pire des cas.

8.b- Que remarque-t-on sur les données de l'échantillon et sur les mesures obtenues ? Peut-on déduire, même de façon approximative, une fonction T(n) reliant T et n ; c'est-à-dire une fonction T(n) permettant de déterminer directement la valeur de T à partir de n.

//comme pour the last question de tp1.

**Ind:** comparer chaque nombre n avec le suivant ; et chaque mesure du temps avec la suivante.

8.c- Comparer entre la complexités théorique et la complexité expérimentale (çàd., les mesures expérimentales). Les prédictions théoriques sont-elles compatibles avec les mesures expérimentales ? //remplacer les valeurs de n dans la formule obtenu en 8.b et comparer avec celles obtenu avec l’algorithme.

**Partie II (Algorithme 2 du test de la primalité)**

**On cherche à améliorer l’algorithme précédent.**

**On sait que tout diviseur i du nombre n vérifie la relation : i ≤ n/2, avec i≠n.**

1- Développer un 2ème algorithme en tenant compte de cette propriété et refaire les questions 2 à 8 de la partie 1.

2- Comparer les 2 algorithmes (représenter pour cela dans une même figure les graphes des 2 algorithmes). Lequel des 2 algorithmes est meilleur (ou plus performant) ?

**Partie III (Algorithme 3 du test de la primalité)**

**On cherche à améliorer encore l’algorithme du test de la primalité.**

**Il existe une propriété mathématique sur les nombres entiers :**

**Propriété : les diviseurs d’un nombre entier n sont pour la moitié ≤ et pour l’autre moitié compris entre .**

1- Développer un 3ème algorithme en tenant compte de cette propriété et refaire toutes les questions 2 à 8 de la partie 1.

2- Comparer les 3 algorithmes (représenter pour cela dans une même figure les graphes des 3 algorithmes). Lequel des 3 algorithmes est meilleur (ou plus performant) ?

**Partie IV (Rapport de travail)**

1- Rédiger un rapport décrivant le travail réalisé.