U.S.T.H.B - 2025-2026
Faculté d'Informatique
N. Bensaou & M.N Djouada

Master Informatique Visuelle
Algorithmique et Complexité (M1 - MIV)

Série de TP

Exercice 1.- Test des fonctions de mesure de temps.

- Exécuter le programme montré dans l'exemple.
- Varier les valeurs de n et afficher les temps respectifs en secondes.
- Modifier le programe pour afficher le temps moyen d'execution pour chaque valeur de n.
- Modifier le programme pour générer un fichier `.csv` en sortie contenant pour chaque valeur de n le temps de moyen d'exécution.
- Exporter le fichier en sortie à excel et dresser les courbes à partir du tableau. Comprer le résultat avec la complexité théorique.

Exercice 2.- La primalité d'un nombre.

Soit $n \in \mathbb{N}$. Proposer trois algorithmes pour vérifier si n est premier ou non.

- Varier les valeurs de n et dresser le tableau de complexité pratique pour chacun des algorithmes.
- Expoter les résultats en Excel et comprer les courbes des trois algorithmes.

Exercice 3.- Calcul des sommes.

Soint les expressions des sommes suivantes :

$$\sum_{i=1}^{n} i \qquad (1) \qquad\qquad \sum_{i=1}^{n} i^2 \qquad (2) \qquad\qquad \sum_{i=1}^{n} i^3 \qquad (3)$$

- Proposer deux algorithmes (avec des complexités différentes) pour le calcul de chacune de ces sommes.
- Varier les valeurs de n et dresser le tableau de complexité pratique pour chacun des algorithmes.
- Expoter les résultats en Excel et comprer les courbes des différentes algorithmes.

Exercice 4.- La génération aléatoire des données.

- En utilisant la fonction `rand` , écrire une fonction `genererTab` qui permet de générer aléatoirement un tableau de n entiers compris entre 0 et k, $0 \leq T[i] \leq k$.
- Ecrire la fonction *rechercheSeq* qui effectue une recherche séquentielle de la valeur x dans le tableau T, elle retourne 1 si x se trouve dans T et 0 sinon.

— Ecrire la fonction $Trier$ qui permet de trier le tableau T.

— Soit T un tableau trié. Ecrire la fonction *rechercheDicho* qui effectue une recherche dichotomique de la valeur x dans le tableau T, elle retourne 1 si x se trouve dans T et 0 sinon.

— Evaluer et comprer les complexités des deux fonction de recherches en comparant leurs courbes pratiques et leurs complexités théoriques.

— Ecrire une fonction `genererMat` qui permet de générer aléatoirement une matrice de$(n \times m)$ entre 0 *et* $k, 0 \leq M[i][j] \leq k$.

— Ecrire la fonction de multiplication de matrices et étudier sa complexité.

— Etudier la complexité de la fonction `puissanceMat` qui calcul la $n^{ème}$ puissance d'une matrice.

Exercice 5.- Le calcul du pgcd.

Soient a et b deux entiers naturels.

1. On peut calculer $pgcd(a, b)$, en utilisant la formule suivante :

$$pgcd(a,b) = \begin{cases} b & si\ a = b \\ pgcd(a-b, b) & si\ a > b \\ pgcd(a, b-a) & si\ a < b \end{cases} \tag{4}$$

2. On peut aussi calculer $pgcd(a, b)$, en suivant la formule :

$$pgcd(a,b) = \begin{cases} a & si\ b = 0 \\ pgcd(b, a\%b) & si\ b \neq 0 \end{cases} \tag{5}$$

— programmer les deux méthodes et comparer leurs complexités pratiques.

Exercice 6.- La racine carrée d'un nombre.

1. Proposer un algorithme naïf (basé sur la dichotomie) pour calculer la racine carrée d'un nombre réel positif A (Avec une marge d'erreur).
2. Etant donnée la suite x, proposer un algorithme pour calculer la racine carrée d'un nombre réel donné A.

$$\begin{aligned} x_0 &= A/2 \\ x_{n+1} &= 1/2(x_n + A/x_n) \\ \lim_{n\to\infty} x_n &= \sqrt{A} \end{aligned}$$

3. Programmer les deux méthodes, étudier leurs complexités pratiques et comparer les avec la méthode `sqrt` dans la bibliothèque standard de votre langage de programmation.

Exercice 7.- La recherche du motif dans une chaine.

Soit T un texte de taille , et P un motif de taille m, on veut chercher P dans T.

— Proposer un algorithme naïf pour résoudre ce problème. Etudier la complexité pratique de cette solution.
— Etudier la complexité des algorithmes proposés par la bibliothèque standard de votre langage. Comparer la complexité pratique avec la solution précédente.

Exercice 8.- Les reines dans un échiquier.

Soit un échiquier de taille $(n \times n)$. nous nous intéressons au problème de placer n reines sur l'échiquer de manière à ce qu'aucune reine ne menace une autre, c'est-à-dire qu'elles ne partagent ni la même ligne, colonne, ni diagonale.
— Etudier le complexité pratique de ce problème.
— Quel est le nombre minimum de reines à placer afin que l'échiquer soit totalement couvert. Etudier la complexité pratique de ce problème.

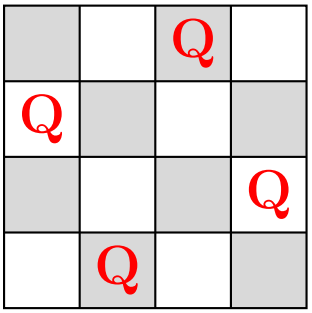

FIGURE 1 – Exemple de solution du problème des n-reines pour n=4.

Exercice 9.- Les tableaux de suffixes et les algorithmes de tri

Une des solutions intéressantes du problème de pattern matching (exercice 7), est de représenter la chaine de texe en utilisant le tableau de suffixes qui représente tous les suffixes du texte. Le temps de la construction de tableau de suffixes est essentiellement basé sur l'algorithme utilisé pour trier les suffixes.
— Etudier la complexité pratique de l'algorithme de création d'un tableau de suffixes en utisant chacun des algorithmes de Tri suivants :
  — Le tri rapide
  — Le tri par tas
  — Le tri fusion
  — Le tri par paquets
— Etudier la complexité pratique de l'algorithme de recherche dans le tableau de suffixes.

Exercice 10.- Les tours de Hanoi

Le problème des Tours de Hanoï consiste à déplacer n disques de tailles différentes d'une tige de départ (A) vers une tige de destination (C), en utilisant une tige intermédiaire (B), tout en respectant les règles suivantes : un seul disque peut être déplacé à la fois, et un disque ne peut jamais être placé sur un disque plus petit.
— Programmer la solution et étudier la complexité pratique de ce problème. Votre solution doit montrer tous les mouvements à effectuer.

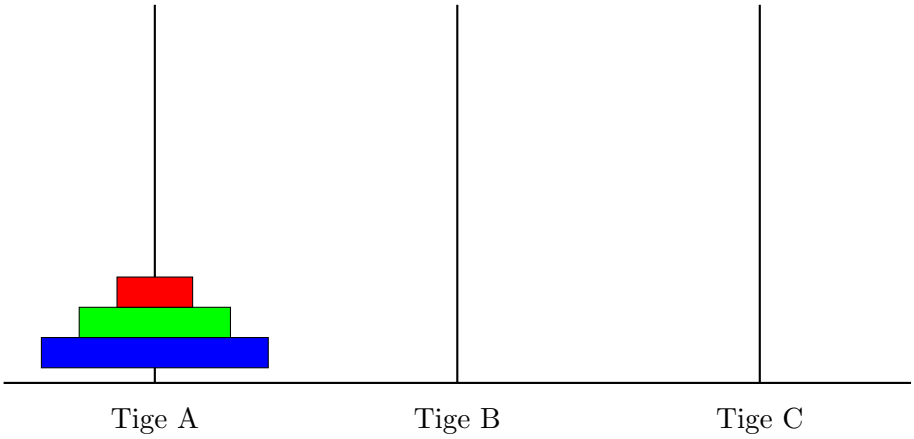

FIGURE 2 – Illustration de la configuration initiale du problème des tours de Hanoi.