ESIR 1 / Algorithmique et Complexité : TP 1
Analyse de programmes simples

Pierre Maurel, pierre.maurel@irisa.fr

1 Introduction

1.1 Fichiers fournis

Vous trouverez sur la page du cours (sur l'ENT) deux fichiers `Tri.java` et `Permutations.java` qu'il vous faudra compléter. Vous trouverez également un fichier MATLAB `etude_complexite.m` qui vous permettra d'afficher et analyser simplement les temps obtenus lors de l'exécution des différents algorithmes.

1.2 Travail demandé

Vous préparerez un compte-rendu (au format PDF) présentant les objectifs du TP, le travail effectué et les résultats obtenus. Il sera bien entendu illustré par des images qui vous sembleront significatives. La note sera essentiellement basée sur le compte-rendu (que vous rendrez sur la page du cours sur l'ENT avant la date indiquée).

2 Analyse de tris

2.1 Tri par insertion

1. Implémentez le tri par insertion dans le fichier `Tri.java`. Vous en trouverez le pseudo-code dans l'exercice 2 du TD 1 (attention : dans le pseudo-code l'indice du premier élément est 1, alors qu'en java c'est 0). Vous testerez rapidement votre algorithme sur un tableau aléatoire (voir `main`).
2. En vous aidant du squelette de `main` fourni, exécutez ce tri sur des tableaux de tailles croissantes et sauvez l'évolution du temps de calcul en fonction de la taille du tableau. Vous effectuerez cette analyse pour trois types d'entrée :
   – le meilleur des cas, un tableau déjà trié dans l'ordre croissant
   – le pire des cas, un tableau déjà trié mais dans l'ordre décroissant
   – des tableaux à valeurs aléatoires
3. Visualisation et analyse des résultats via MATLAB
   – La fonction `tab_to_matlab` (fournie) permet de sauvegarder la liste des tailles ainsi que la liste des temps d'exécution au format MATLAB.

- Ouvrez le fichier `etude_complexite.m` à l'aide du logiciel de calcul numérique MATLAB. Exécutez-le pour afficher la courbe $T(n)$ du temps d'exécution en fonction de la taille.
- En fonction des complexités correspondantes aux différents cas (et que nous avons calculées lors du TD 1), effectuez une régression manuelle : trouvez (approximativement) les coefficients du polynôme $T(n)$ qui expliquent le mieux vos mesures.
- Grâce au résultat précédent, estimez le temps que prendrait cet algorithme, dans les différents cas, pour trier un tableau contenant les numéros de sécurité sociale de la population française (66 millions d'habitants) dans les trois cas. Donner cette estimation en unités "compréhensibles" (années → jours → heures → minutes → secondes).

Remarques

- Si vous obtenez l'erreur suivante :

`Exception in thread "main" java.lang.OutOfMemoryError: Java heap space,`

  dans `Run->Run Configuration` trouvez le nom de votre classe, selectionnez la, cliquez sur l'onglet `Arguments` et ajouter `-Xms512M -Xmx1524M` aux arguments VM.
- Pour obtenir de l'aide dans MATLAB, tapez `doc` suivi du nom de la fonction au sujet de laquelle vous voulez des informations.
- La régression peut se faire de manière automatique à l'aide de `polyfit`.

2.2 Tri fusion

Complétez l'implémentation du tri fusion. Vous en trouverez le pseudo-code dans l'exercice 6 du TD 2. Effectuez la même analyse que pour le tri par insertion dans le cas de tableaux aléatoires. Modifiez `etude_complexite.m` pour prendre en compte la complexité calculée lors du TD 2 et estimer également le temps que prendrait cet algorithme pour trier un tableau contenant les numéros de sécurité sociale de la population française (66 millions d'habitants). Conclusions ?

3 Permutations

Vous trouverez dans le fichier `Permutations.java` une méthode `permutation`. Testez cette fonction à l'aide du `main` fourni.

- Que fait cette fonction ?
- Quelle est la complexité de cette fonction ? Son implémentation est récursive, ce qui ne facilite pas sa compréhension et son analyse. Cependant vous pouvez estimer la complexité de la fonction sans comprendre son fonctionnement.
- Désactivez l'affichage (à l'aide du paramètre `afficher`) et étudiez l'évolution du temps de calcul en fonction de n sous MATLAB comme précédemment.
- Estimez le temps nécessaire pour exécuter ce programme sur la totalité de l'alphabet ($n = 26$).