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

Pierre Maurel, pierre.maurel@irisa.fr

1 Introduction

1.1 Fichiers fournis

Vous trouverez sur la page du cours (sur l'ENT/moodle) plusieurs fichiers `.java` qu'il vous faudra compléter.

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).

1.3 Remarques

— Vous allez utilisez la fonction Java `System.currentTimeMillis` pour mesurer des temps d'exécution. Il est important de comprendre que cette mesure sera approximative (précision de la mesure + variabilité dûe à votre machine qui fait d'autres choses en même temps). Des temps de quelques millisecondes ne seront donc pas significatifs. À vous d'adapter la taille des tableaux dans chaque situation pour avoir des temps de calculs d'au moins plusieurs dizaines de millisecondes.

— L'objectif de ce TP réside plus dans la compréhension des observations que dans la programmation à proprement dit : n'hésitez donc pas à poser des questions à vos encadrants (via discord) !

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` de `Mesure_tri`).
2. Vous allez maintenant analyser expérimentalement le temps d'exécution du tri par insertion pour trois types d'entrées
   — le meilleur des cas : tableaux déjà triés dans l'ordre croissant
   — le pire des cas : tableaux déjà triés mais dans l'ordre décroissant
   — cas aléatoire : tableaux à valeurs aléatoires

   Successivement, pour chaque type d'entrée, il faudra donc :

(a) En vous aidant du squelette de `main` fourni (`Mesure_tri.java`), exécutez ce tri sur des tableaux de tailles croissantes et sauvez les temps de calcul en fonction de la taille du tableau.

(b) Toujours dans le `main`, vous trouverez des instructions permettant l'affichage des mesures effectuées (temps de calcul en fonction de la taille)

(c) En fonction des complexités correspondantes aux différents cas (et que nous avons calculées lors du TD 1), effectuez une régression manuelle, c'est à dire : cherchez (approximativement) les coefficients du polynôme $T(n)$ qui expliquent le mieux vos mesures.

(d) Grâce au résultat précédent, estimez le temps que prendrait le tri par insertion, sur le type d'entrées considérées, pour trier un tableau contenant, par exemple, les numéros de sécurité sociale de la population française (67 millions d'habitants). Donner cette estimation en unités "compréhensibles" (années → jours → heures → minutes → secondes).

2.2 Tri fusion

Complétez l'implémentation du tri fusion. Vous en trouverez le pseudo-code dans l'exercice 5 du TD 2. Effectuez la même analyse que pour le tri par insertion, **uniquement** dans le cas de tableaux aléatoires [1]. En particulier, estimez également le temps que prendrait cet algorithme pour trier un tableau contenant les numéros de sécurité sociale de la population française (67 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. On peut, par exemple, supposer que la complexité va être du même ordre de grandeur que le nombre d'affichages réalisés.

— Désactivez l'affichage (à l'aide du paramètre `afficher`) et étudiez l'évolution du temps de calcul en fonction de n, comme précédemment.

— Estimez le temps nécessaire pour exécuter ce programme sur la totalité de l'alphabet ($n = 26$).

4 Bonus, s'il vous reste du temps

Le fichier `Ackermann.java` contient une implémentation de la fonction d'Ackermann. Vous pouvez la regarder, tester et éventuellement mesurer des temps d'exécution. Si besoin vous pouvez augmenter la taille de la pile, en ajoutant, par exemple, `-Xss1m` dans les `VM argument` (Run> Run configuration> Ackerman).

1. Contrairement au tri par insertion, le tri fusion présente la même complexité, en ordre de grandeur, quelque soit le type d'entrée