Université de Limoges

Algorithmique et Complexité

TP n° 2 : Tri par insertion, Tri rapide

Vous écrirez vos programmes en langage C.

1 Les algorithmes

1.1 Tri par insertion

On rappelle l'algorithme du tri par insertion.

Algorithme 1 Tri par insertion(T)

Entrée: T un tableau de n entiers
Sortie: Le tableau T trié

1. **for** $j = 1$ **to** $n - 1$ **do**
2. $\quad v \leftarrow T[j]$
3. $\quad i \leftarrow j - 1$
4. $\quad$ **while** $i \geqslant 0$ **and** $T[i] > v$ **do**
5. $\quad\quad T[i + 1] \leftarrow T[i]$
6. $\quad\quad i \leftarrow i - 1$
7. $\quad T[i + 1] \leftarrow v$

1.2 Tri rapide

On rappelle l'algorithme récursif du tri rapide.

Algorithme 2 TriRapide(T, inf, sup)

Entrée: T un tableau de n entiers, inf et sup des indices
Sortie: Le tableau T avec le sous-tableau $T[inf \dots sup]$ étant trié

1. $p \leftarrow$ Pivoter(T, inf, sup)
2. **if** $inf < p$ **then**
3. $\quad$ TriRapide(T, inf, $p - 1$)
4. **if** $p < sup$ **then**
5. $\quad$ TriRapide(T, $p + 1$, sup)

La fonction *Pivoter* réorganise le sous-tableau $T[inf \dots sup]$ autour d'une valeur arbitraire appelée *pivot* (nous choisissons comme pivot le premier élément). Toutes les valeurs plus petites que le pivot sont placées avant lui, et toutes les valeurs plus grandes sont placées après.

Algorithme 3 Pivoter(T, inf, sup)

Entrée: T un tableau de n entiers, $inf \leqslant sup$ des indices
Sortie: Réorganise $T[inf \dots sup]$ autour du pivot et retourne son indice

```
1.  v ← T[inf]                                  [v est la valeur du pivot]
2.  i ← inf
3.  s ← sup
4.  while i ≠ s do
5.      if T[i + 1] ⩽ v then
6.          T[i] ← T[i + 1]
7.          i ← i + 1
8.      else
9.          permuter T[i + 1] et T[s]
10.         s ← s − 1
11. T[i] ← v                          [On place le pivot à sa position finale]
12. return i
```

2 Comparaison des deux méthodes de tri

1. Écrivez un programme qui compare les temps d'exécution des deux méthodes de tri.

   La taille du tableau n, et le nombre de répétitions r sont récupérées sur la ligne de commande (avec *argc* et *argv*).

   Le programme mesure le temps d'exécution pour effectuer r fois un tri par insertion, puis le temps pour effectuer r fois un tri rapide.

   Pour chaque répétition du tri le tableau est initialisé avec des entiers aléatoires compris entre 0 et $2n - 1$.

   Remarque : l'intérêt de répéter r fois le tri est d'obtenir un temps d'exécution non négligeable et dont la mesure est significative. On pourra prendre pour valeur $r = 1\,000\,000$ par exemple.

   La durée séparant deux instants d'exécution de votre programme peut être calculée à l'aide de la fonction suivante qui retourne le temps CPU (exprimé en milli-secondes) écoulé depuis le début de l'exécution du programme :

```
#include <sys/types.h>
#include <sys/resource.h>

unsigned long int cputime()
{
  struct rusage rus;

  getrusage (0, &rus);
  return rus.ru_utime.tv_sec * 1000 + rus.ru_utime.tv_usec / 1000;
}
```

2. Si votre programme affiche uniquement une ligne de la forme

   ```
   n t_insertion t_rapide
   ```

   vous pourrez facilement créer un fichier texte contenant ces données sur deux colonnes pour des valeurs croissantes de n.

   Remarque : l'avantage de lire la valeur de n sur la ligne de commande est de pouvoir enchaîner facilement diverses exécutions pour différentes valeurs de n et de rediriger le tout dans un fichier :

   ```
   $ for n in $(seq 1 200)
   > do
   > mon_programme $n
   > done > data.txt
   ```

3. Utilisez l'outil de visualisation *gnuplot* pour tracer la courbe de complexité de cette méthode classique :

   ```
   $ gnuplot
   gnuplot> plot "data.txt" with lines
   gnuplot> quit
   $
   ```