Cours de programmation séquentielle

Tris divers et performances

1 Buts

— Utilisation de tableaux unidimensionnels.
— Utilisation de pointeurs de fonctions.
— Implémentations d'algorithmes de tris.
— Mesure de temps d'exécution d'une application.
— Utilisation de `git` et de `make`.

2 Énoncé

Implémenter les algorithmes du tri à bulles, du tri rapide et du tri à deux piles pour des tableaux de nombres entiers remplis de nombres aléatoires dont la taille sera passée en argument du programme et mesurer les performances de chaque algorithme.

Pour chaque algorithme, la signature de la fonction de tri doit être de la forme

```
void sort(int32_t *sorted, const int32_t *const orig,
          int32_t nitems, bool (*comp)(int32_t, int32_t));
```

afin de permettre de trier par ordre croissant ou décroissant sans changer l'implémentation de vote algorithme de tri.

Pour chacun de ces tris, écrire un programme permettant de vérifier automatiquement si le tableau final est bien trié. De plus, écrire un programme exécutant au moins 1000 tris successifs et mesurer la moyenne et l'écart-type du temps d'exécution de chaque tri pour différentes tailles de tableaux et comparer les temps moyens d'exécution entre les différents tris pour déterminer le plus efficace.

2.1 Remarques

— Chaque programme devra avoir sa propre cible dans le `Makefile`.
— Les mesures de temps peuvent être effectuées avec la fonction `clock_gettime()` de la librairie `time.h` comme montré ci-dessous:

```
struct timespec start, finish;
clock_gettime(CLOCK_MONOTONIC, &start);

// code à mesurer

clock_gettime(CLOCK_MONOTONIC, &finish);
```

```
double seconds_elapsed = finish.tv_sec-start.tv_sec;
seconds_elapsed += (finish.tv_nsec-start.tv_nsec)/1.0e9;
```

— N'oubliez pas de ne mesurer que les fonctions de tris.
— Pour le tri rapide, le pivot doit être choisi de façon aléatoire.