Algorithmes et structures de données

Laboratoire 3

Nathalie Mégevand, Lara Chauffoureaux, Marie Lemdjo

10 mai 2016

Estimations théoriques

Tri par sélection

Pour le tri par sélection, nous avons une première affectation de iMin dans la boucle extérieur (qui va donc s'exécuter N fois). Puis, probablement une permutation des valeurs de i et iMin. La boucle interne parcours le tableau de N à i. Cela revient à faire $\sum_{k=0}^{N}(N-k) = \frac{N(N+1)}{2}$, donc, les instructions à l'intérieur (une comparaison et éventuellement une affectation) s'exécutent $\frac{N^2+N}{2}$ fois. Nous avons donc au pire : $N + N + 2 * \frac{N^2+N}{2} = N^2 + 3N$ opérations.

Tri rapide

Le tri rapide est un algorithme particulier. Son temps d'exécution en moyenne est de $\mathcal{O}(Nlog(N))$ et de $\mathcal{O}(N^2)$ dans le pire des cas(situation d'un tableau trié de façon inverse). Il n'utilise pas de tableau annexe pour trier, donc il nécessite très peu de place mémoire pour s'exécuter. Néanmoins, c'est un tri instable. De manière plus détaillée, on a Nlog(N) comparaisons, Nlog(N) permutations, 2N appels de la fonction elle-même et N partitionements. Ce qui nous donne un total de $Nlog(N) + Nlog(N) + 2N + N = 2Nlog(N) + 3N$ opérations.

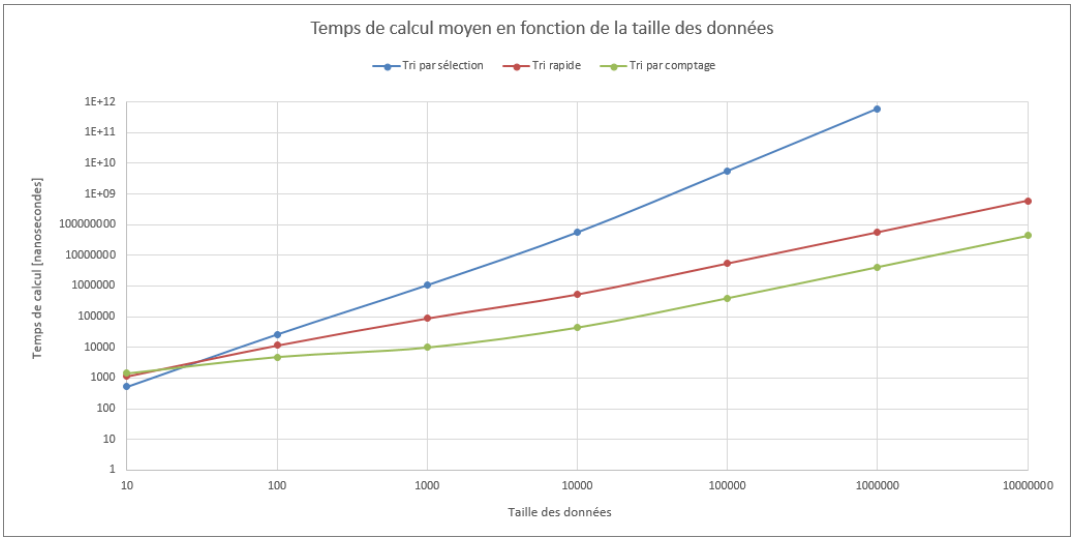
Tri par comptage

Pour le tri par comptage, nous exécutons une première boucle for parcourant tout le tableau à trier afin de trouver le min et le max, dans laquelle nous executons au plus une seule instruction d'affectation et une instruction de comparaison par itération. Puis, nous exécutons une deuxième boucle for parcourant tout le tableau lors de notre phase de comptage. À chaque itération, nous exécutons une incrémentation. Enfin, nous allons replacer chaque élément dans le tableau final, cela revient à parcourir une troisième fois le tableau dans son ensemble, tout en exécutant une instruction par itération. Nous avons donc $2N + N + N = 4N$ instructions pour l'ensemble du tri comptage.

Résultats empiriques

Estimations théoriques[1]			
Taille	**Tri par sélection**	**Tri rapide**	**Tri par comptage**
10^1	1300	500	400
10^2	103000	7000	4000
10^3	1.003e+07	30000	40000
10^4	1.0003e+09	1.1e+06	40000
10^5	1.00003e+11	1.3e+07	400000
10^6	1.000003e+13	1.5e+07	4.00e+06
10^7	1.0000003e+15	1.7e+08	4.00e+e07

Résultats empiriques			
Taille	**Tri par sélection**	**Tri rapide**	**Tri par comptage**
10^1	515.2	1151.87	1424.7
10^2	26211.9	11587.5	4874.43
10^3	1.05628e+06	87814.7	9936.43
10^4	5.538e+07	530346	44343.7
10^5	5.54646e+09	5.36855e+06	402290
10^6	5.9372e+11	5.53667e+07	4.15027e+06
10^7	NA	5.8798e+08	4.35604e+07



1. Ces temps théoriques on été calculés en prenant arbitrairement 10 nanosecondes de temps d'exécution pour une instruction.

Discussion des résultats

Après avoir lancé les simulations jusqu'à 10^6, nous avons extrapolé nos résultats et obtenu un résultat théorique de 15h de simulation pour le tri par sélection sur un tableau de 10^7 données. Lancer les 30 simulations nous aurait pris $\sim$ 3 semaines. Nous avons donc exécuté les dernières simulations pour les tris rapide et par comptage seulement.

Le graphique ci-dessus (échelle logarithmique sur les deux axes), rend bien visible la différence de performance entre les algorithmes de tris. Le tri par sélection, bien que plus rapide sur des petites tailles de données, devient vite inutilisable pour de plus grands N.

Le tri par comptage quant à lui est très efficace du moment que la taille des tableaux à trier n'est pas trop grande. En effet, ce tri est très rapide, mais demande une grande quantité de mémoire. De plus, ce tri n'est possible que sur des valeurs discrètes.

Le tri rapide est un bon compromis temps-mémoire et peut être utilisé sur toutes les données pouvant être comparées. Il est normal que cet algorithme soit le premier choix pour l'implémentation d'un tri générique.