Mesures de temps d’exécution d’algorithmes de tri et de recherche avec Python

BONNAIRE Lucas & RENAULT Rémi

16/12/2024

Toutes les mesures ont été prises sur la base de scripts Python exécutés sur un MacBook Pro M4 Pro avec 24Go de RAM.

Le code source utilisé pour ces mesures est disponible sur ce repository GitHub :

https://github.com/CORT1N/esgi-algorithmic-works

Toutes les mesures ont été moyennées sur la base de 100 exécutions.

Table des matières

1. Le tri à bulles

Le tri à bulles compare et échange les éléments adjacents d'une liste jusqu'à ce que celle-ci soit triée.

Pour 10 valeurs entre 0 et 2 :

```
The bubble sort function took on average 0.0019264233 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The bubble sort function took on average 0.12120723780000006 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The bubble sort function took on average 14.0910935406 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The bubble sort function took on average 0.0018215181000000003 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The bubble sort function took on average 0.1359605786 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The bubble sort function took on average 15.7772684098 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The bubble sort function took on average 0.0018095970000000002 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The bubble sort function took on average 0.1377201083 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The bubble sort function took on average 16.398272514100007 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The bubble sort function took on average 0.0020074855000000004 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The bubble sort function took on average 0.14153957350000007 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The bubble sort function took on average 16.915090084 ms
```

2. Le tri par sélection

Le tri par sélection consiste à trouver l'élément le plus petit (ou le plus grand) à chaque étape et à le placer à sa position correcte dans la liste.

Pour 10 valeurs entre 0 et 2 :

```
The selection sort function took on average 0.0014471995000000012 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The selection sort function took on average 0.07855653759999999 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The selection sort function took on average 7.8662967685 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The selection sort function took on average 0.0013899790000000013 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The selection sort function took on average 0.08055210150000003 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The selection sort function took on average 8.0833482747 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The selection sort function took on average 0.0013709054000000007 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The selection sort function took on average 0.07897853879999998 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The selection sort function took on average 8.1728148463 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The selection sort function took on average 0.001373289400000001 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The selection sort function took on average 0.08099794429999997 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The selection sort function took on average 8.140213489399999 ms
```

3. Le tri par insertion

Le tri par insertion consiste à insérer chaque élément dans sa position correcte par rapport aux éléments déjà triés, en les déplaçant progressivement.

Pour 10 valeurs entre 0 et 2 :

```
The insertion sort function took on average 0.0012373903000000013 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The insertion sort function took on average 0.06028413760000001 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The insertion sort function took on average 5.690827369699998 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The insertion sort function took on average 0.0012254694000000015 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The insertion sort function took on average 0.0868582728 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The insertion sort function took on average 7.8150272368000016 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The insertion sort function took on average 0.0013494477000000004 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The insertion sort function took on average 0.08661270140000003 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The insertion sort function took on average 8.638606071699995 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The insertion sort function took on average 0.001409052300000001 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The insertion sort function took on average 0.08766889569999999 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The insertion sort function took on average 8.897655010300001 ms
```

4. Le tri rapide

Le tri rapide consiste à choisir un pivot, partitionner la liste autour de ce pivot, puis trier récursivement les sous-listes à gauche et à droite du pivot.

Pour 10 valeurs entre 0 et 2 :

```
The quick sort function took on average 0.002567767900000001 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The quick sort function took on average 0.06512403430000005 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The quick sort function took on average 3.332204818699999 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The quick sort function took on average 0.002479553899999999 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The quick sort function took on average 0.04333496160000002 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The quick sort function took on average 1.3569736483000003 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The quick sort function took on average 0.0024747854000000003 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The quick sort function took on average 0.035216808500000016 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The quick sort function took on average 0.5498385434000002 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The quick sort function took on average 0.0029397007 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The quick sort function took on average 0.04017829900000001 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The quick sort function took on average 0.5056166649000001 ms
```

5. Le tri hollandais

Le tri hollandais (ou tri de Dijkstra) est un algorithme qui partitionne une liste en trois groupes distincts (par exemple, inférieur, égal et supérieur à un pivot) afin de trier efficacement des données avec des valeurs répétées ou limitées.

Par son fonctionnement, il est limité à 3 valeurs distinctes maximum, ce qui oblige les mesures à se restreindre à l'intervalle [0, 2].

Pour 10 valeurs entre 0 et 2 :

```
The dutch sort function took on average 0.0006818747999999997 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The dutch sort function took on average 0.003345012000000001 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The dutch sort function took on average 0.0356173513 ms
```

6. Le tri fusionné

Le tri fusionné divise récursivement la liste en sous-listes, puis fusionne les sous-listes triées pour obtenir une liste finale triée.

Pour 10 valeurs entre 0 et 2 :

```
The merge sort function took on average 0.004315377800000004 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The merge sort function took on average 0.059151650899999999 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The merge sort function took on average 0.8315277093999996 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The merge sort function took on average 0.004000665500000004 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The merge sort function took on average 0.0609326352 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The merge sort function took on average 0.8681321143999999 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The merge sort function took on average 0.004215242400000005 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The merge sort function took on average 0.061023233800000035 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The merge sort function took on average 0.8735227581000005 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The merge sort function took on average 0.004391672200000002 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The merge sort function took on average 0.0648355488 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The merge sort function took on average 0.90245962139999995 ms
```

Interprétation des résultats pour les algorithmes de tri

Pour le tri à bulles, on remarque surtout qu'avec 1000 valeurs, quelle que soit l'intervalle de valeurs utilisée, il devient beaucoup plus lent.

Même affirmation pour le tri par sélection et le tri par insertion qui divisent cependant pour la même quantité et la même intervalle le temps d'exécution par deux.

Concernant le tri rapide, il porte bien son nom, mais surtout devient de plus en plus efficace à mesure que l'intervalle de valeurs utilisée s'agrandit.

Enfin, pour le tri fusionné, on remarque qu'il est nettement meilleur et plus stable que tous les autres, sans dépasser une seule fois 1ms de temps d'exécution.

Nous n'avons pas abordé le tri hollandais qui n'est pas une mesure cohérente comparée aux autres, ne s'appliquant qu'aux intervalles très petites d'un maximum de trois.

Nous pouvons conclure de nos observations que le tri fusionné est le plus efficace en toutes circonstances selon ces mesures.

Le tri rapide le concurrence très facilement à mesure que l'intervalle de valeurs utilisées s'agrandit.

Les autres types de tri sont plus efficaces pour de petites quantités de données.

7. La recherche dichotomique

La recherche dichotomique (ou binaire) consiste à diviser récursivement une liste triée en deux moitiés pour localiser un élément en réduisant à chaque fois l'espace de recherche.

Pour 10 valeurs entre 0 et 2 :

```
The dichotomic search function took on average 0.00037431599999999997 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The dichotomic search function took on average 0.0006651856999999995 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The dichotomic search function took on average 0.0008797619000000004 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The dichotomic search function took on average 0.00042676769999999974 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The dichotomic search function took on average 0.00047445129999999953 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The dichotomic search function took on average 0.0008726094000000003 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The dichotomic search function took on average 0.000305175 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The dichotomic search function took on average 0.00047445129999999953 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The dichotomic search function took on average 0.0010275811000000008 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The dichotomic search function took on average 0.0003647792 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The dichotomic search function took on average 0.0005054458999999997 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The dichotomic search function took on average 0.0007963152 ms
```

8. La recherche linéaire

La recherche linéaire consiste à parcourir séquentiellement chaque élément d'une liste jusqu'à trouver l'élément recherché.

Pour 10 valeurs entre 0 et 2 :

```
The linear search function took on average 0.00025749130000000004 ms
```

Pour 100 valeurs entre 0 et 2 :

```
The linear search function took on average 0.0012898423000000012 ms
```

Pour 1000 valeurs entre 0 et 2 :

```
The linear search function took on average 0.0115823731999999996 ms
```

Pour 10 valeurs entre 0 et 9 :

```
The linear search function took on average 0.00036477959999999985 ms
```

Pour 100 valeurs entre 0 et 9 :

```
The linear search function took on average 0.0012469273000000015 ms
```

Pour 1000 valeurs entre 0 et 9 :

```
The linear search function took on average 0.011417864199999996 ms
```

Pour 10 valeurs entre 0 et 99 :

```
The linear search function took on average 0.00017166070000000001 ms
```

Pour 100 valeurs entre 0 et 99 :

```
The linear search function took on average 0.0008964518000000006 ms
```

Pour 1000 valeurs entre 0 et 99 :

```
The linear search function took on average 0.001251696400000001 ms
```

Pour 10 valeurs entre 0 et 999 :

```
The linear search function took on average 0.0002717963 ms
```

Pour 100 valeurs entre 0 et 999 :

```
The linear search function took on average 0.0014972679000000008 ms
```

Pour 1000 valeurs entre 0 et 999 :

```
The linear search function took on average 0.008139610099999998 ms
```

Interprétation des résultats pour les algorithmes de recherches

Contrairement aux algorithmes de tri où les écarts étaient flagrants avec les mêmes intervalles et quantités de valeurs, ici les algorithmes de recherche s'avèrent efficaces en toutes circonstances.

Un point intéressant cependant à noter est que l'algorithme de recherche linéaire s'avère légèrement moins puissant lorsqu'on atteint 1000 valeurs.

Même après avoir testé avec beaucoup plus de valeurs, où une intervalle plus grande, les deux algorithmes ne perdaient pas en efficacité.