
Projet 1 : 2017 - 2018

# Structures des données et algorithmes

Prof. P. Geurts

Perin Alexis
Muramutsa Jean

23 mars 2018

# 1) Analyse expérimentale

| $n$ | InsertionSort | QuickSort | MergeSort | HeapSort |
|---|---|---|---|---|
| 10 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 0 |
| 1.000 | 0.002 | 0 | 0 | 0 |
| 10.000 | 0.061 | 0.004 | 0.004 | 0.004 |
| 100.000 | 5.869 | 0.035 | 0.015 | 0.025 |
| 1.000.000 | 783.77 | 2.494 | 0.166 | 0.291 |

Nous remarquons, en premier lieu, que les quatre algorithmes testés ont des temps de calcul quasiment nuls pour des valeurs de $n$ inférieures à 10000. $InsertionSort$ est celui qui se démarque le plus au delà de ce seuil. En effet sa complexité en temps en moyenne et dans son pire cas $\mathcal{O}(n^2)$ fait de lui l'algorithme le moins rapide des quatre. Les trois autres algorithmes ont une complexité moyenne dans le temps $\mathcal{O}(nlog(n))$. Tandis que $QuickSort$ a une complexité $\mathcal{O}(n^2)$ dans son pire cas, faisant de lui l'algorithme qui triera le moins rapidement des tableaux de tailles plus élevées parmi les trois restant, les deux autres ont une complexité dans le temps identique, $\mathcal{O}(nlog(n))$ dans leur pire cas. La différence de temps de calcul entre ces derniers s'explique par le système d'arbre binaire utilisé par $HeapSort$ qui rend cet algorithme moins rapide que $MergeSort$.

# 2) OtherSort

## 1) Pseudo-Code

```
OtherSort(A, p, r)
1  n = r − p + 1
2  if n > 1
3        if A[p] > A[r]
4              swap(A[p], A[r])
5        if n ≥ 3
6              x = n/3
7              OtherSort(A, p, r − x)
8              OtherSort(A, p + x, r)
9              OtherSort(A, p, r − x)
```

## 2) Correction de l'algorithme

Nous démontrons que l'algorithme est correct par induction. Sachant qu'un tableau de taille $n = 2$ sera trié en une seule opération ($swap$) et que pour un tableau de taille $n = 3$, des sous-tableaux de taille 2 seront triés récursivement de la même manière, nous concluons que l'algorithme parviendra également à trier des tableaux de taille supérieure.

## 3) Stabilité.

L'algorithme est stable car, l'opération d'échange entre deux valeurs du tableau n'ayant lieu que lorsque l'une est strictement supérieure à l'autre, l'ordre relatif des éléments égaux est conservé. L'algorithme $Othersort$ est également en place car il opère directement dans la structure sans créer de nouveau tableau.

## 4) Étude expérimentale de la complexité en temps

| $n$ | InsertionSort | QuickSort | MergeSort | HeapSort | OtherSort |
|---|---|---|---|---|---|
| 10 | 0 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 0 | 0.004 |
| 1.000 | 0.002 | 0 | 0 | 0 | 0.272 |
| 10.000 | 0.061 | 0.004 | 0.004 | 0.004 | 194.663 |
| 100.000 | 5.869 | 0.035 | 0.015 | 0.025 | $\infty$ |
| 1.000.000 | 783.77 | 2.494 | 0.166 | 0.291 | $\infty$ |

*OtherSort* est l'algorithme de tri ayant les temps de calcul les plus élevés parmi les cinq algorithmes testés dans le cadre de ce projet. Ces temps de calcul augmentant exponentiellement, des valeurs infinies sont observées pour les tris de tableaux de grandes tailles.

## 5) Complexité en temps et en espace

L'algorithme effectuant trois appels récursifs en travaillant dans des sous-tableaux dont la taille vaut deux tiers de la taille initiale du tableau et y effectuant au besoin des opérations élémentaires, le temps d'exécution est, pour un tableau de taille n, donné par $T(n) = 3T(2n/3) + \mathcal{O}(1)$. Ainsi, la complexité moyenne dans le temps est $\mathcal{O}(n^{ln(3)/ln(1.5)}) = \mathcal{O}(n^{2.7095\cdots})$. *OtherSort* est donc moins rapide que *InsertionSort*, qui avec sa complexité moyenne en temps $\mathcal{O}(n^2)$ était le plus lent des quatre autres algorithmes étudiés. Notons qu'il s'agit également de la complexité en temps dans le meilleur et le pire cas, étant donné que l'algorithme effectue des divisions de la taille du tableau peu importe l'ordre initial des éléments qu'il contient. La complexité spatiale est $\mathcal{O}(n)$.

## 6) Intérêt pratique

*OtherSort* est un algorithme relativement facile à implémenter, son code ne tenant que sur quelques lignes, et qui permet de trier rapidement des tableaux de petite taille. Cependant, pour des tableaux de tailles plus élevées, ses temps de calcul vont rapidement et fortement augmenter, ce qui rendra impossible son utilisation en pratique. Nous lui préférerons donc les autres algorithmes de tri cités plus tôt.