école dE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

Rapport de laboratoire 1

PRÉSENTÉ À

SAbeur Lafi

Dans le cadre du cours :

ALGORITHMES

ELE440-01

par

Julien lemay (LEMJ16059303)

Alexandre Lessard (LESA30099400)

montréal, le 2 octobre 2015

cid:image002.jpg@01CCE021.010CA410 Julien Lemay et Alexandre Lessard, 2015

Sommaire

[Introduction 1](#_Toc431568571)

[Protocole de test 2](#_Toc431568572)

[1. Tri par insertion 3](#_Toc431568573)

[1.1. Algorithme 3](#_Toc431568574)

[1.2. Analyse théorique 3](#_Toc431568575)

[1.3. Analyse expérimentale 4](#_Toc431568576)

[2. Tri par fusion 7](#_Toc431568577)

[2.1. Algorithme 7](#_Toc431568578)

[2.2. Analyse théorique 8](#_Toc431568579)

[2.3. Analyse expérimentale 9](#_Toc431568580)

[3. Tri par pigeonnier 12](#_Toc431568581)

[3.1. Algorithme 12](#_Toc431568582)

[3.2. Analyse théorique 12](#_Toc431568583)

[3.3. Analyse expérimentale 13](#_Toc431568584)

[4. Tri par tas 15](#_Toc431568585)

[4.1. Algorithme 15](#_Toc431568586)

[4.2. Analyse théorique 16](#_Toc431568587)

[4.3. Analyse expérimentale 17](#_Toc431568588)

[5. Tri rapide 19](#_Toc431568589)

[5.1. Algorithme 19](#_Toc431568590)

[5.2. Analyse théorique 20](#_Toc431568591)

[5.3. Analyse expérimentale 21](#_Toc431568592)

[6. Tri par base 24](#_Toc431568593)

[6.1. Algorithme 24](#_Toc431568594)

[6.2. Analyse théorique 25](#_Toc431568595)

[6.3. Analyse expérimentale 25](#_Toc431568596)

[Conclusion 27](#_Toc431568597)

[Annexe 28](#_Toc431568598)

[Références 28](#_Toc431568599)

# Introduction

Le but de ce laboratoire est d’étudier le comportement et la performance de plusieurs algorithmes de tri selon différentes configurations.

Les objectifs de ce laboratoire sont l’implémentation des différents algorithmes en langage de programmation C, C++ ou Java ainsi que d’utiliser l’analyse asymptomatique afin d’analyser les performances des algorithmes de tri.

Ce rapport comporte trois grandes parties pour chacun des 6 algorithmes. La première partie contient les pseudo-codes qui forment le squelette de nos fonctions de tri. La seconde contient l’analyse théorique où l’on discute de l’efficacité des algorithmes se basant sur leurs formules asymptotiques et baromètres. La troisième section contient l’analyse expérimentale des 6 méthodes de tri c'est-à-dire une description de l’algorithme de test, les résultats obtenus par les 6 fonctions et les conclusions qu’on peut en tirer.

# Protocole de test

Le protocole de test est décrit dans l’énoncé de laboratoire à l’aide d’un pseudo-code. Ce dernier permet de tester les 6 méthodes de tri avec 3 paramètres d’entrée. N, R et D. Le paramètre N représente le nombre d’éléments  à trier. Le paramètre R représente la plus grosse valeur possible qu’on puisse trouver dans le tableau à trier et finalement, le paramètre D (en pourcentages) représente le degré de désordre dans le tableau à trier (la première valeur étant la plus petite).

L’algorithme de test utilise 10 valeurs de N, 4 valeurs de R et 5 valeurs de D;

N = 1000×K avec K = [10, 20, 30, ..., 100]

R =

D = [0, 25, 50, 75, 100] %

L’algorithme ne fait plus qu’itérer à travers toutes les combinaisons possibles de ces trois paramètres en plus de faire le test 10 fois pour chacune des combinaisons pour chacune des méthode de tri afin de s’assurer de la justesse des résultats pour un total de 10\*4\*5\*10\*6=12'000 tris de tableaux générés aléatoirement.

# Tri par insertion

## Algorithme

Voici l’algorithme de tri par insertion tiré de la page Wikipedia de cet algorithme1:

**procédure** tri\_insertion(tableau T, entier n)

**pour** i de 1 à n-1

x ← T[i]

j ← i

**tant que** j > 0 et T[j - 1] > x

T[j] ← T[j - 1]

j ← j - 1

**fin tant que**

T[j] ← x

**fin pour**

**fin procédure**

Le tri par insertion fonctionne un peu comme on trie les cartes dans nos mains. On prend chaque carte (valeur). En allant de gauche à droite, une valeur à la fois, on vérifie si la valeur précédente est plus petite. Si oui, on la déplace devant cette valeur et on continue tant qu’il n’y a pas de valeurs plus petites (ou plus de valeurs) à gauche. Une fois la première valeur faite, on passe à la deuxième et ainsi de suite. Rendu à la fin, le « paquet » de valeurs est maintenant trié en ordre croissant.

Cet algorithme a été le plus simple à implémenter, mais est loin d’être efficace.

## Analyse théorique

Les baromètres sont :

Le baromètre le plus important à surveiller est .

Dans le meilleurs des cas, le tableau est en ordre et on ne rentre qu’une fois dans la boucle « tant que » de la ligne 5 à chaque « pour » à la ligne 2. Dans le pire des cas, le tableau est en désordre total et on rentre N fois dans la boucle « tant que » de la ligne 5 à chaque « pour » à la ligne 2.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Min/Max |  |  |

Donc, pour le tri par insertion :

## Analyse expérimentale

L’analyse expérimentale a été effectuée selon le protocole indiqué dans les instructions du laboratoire 1.

Lorsqu’on observe les performances selon N qui est le nombre de données, on remarque facilement que l’asymptote supérieure suit une courbe qui est de l’ordre de . L’asymptote inférieure, un peu plus dur à discerner, est quant à lui de l’ordre de . L’ordre moyen est aussi de l’ordre de .

Lorsqu’on observe les performances selon R qui est l’intervalle de données, on remarque que peut importe l’intervalle de donnée, le nombre d’instructions est indépendant de l’intervalle de données.

Lorsqu’on observe les performances selon D qui est le degré de désordre des données, on remarque que le nombre d’instruction est proportionnel au degré de désordre des données.

Il est aussi a noter que cet algorithme a été celui qui a été le plus long à exécuter parmi tous les algorithmes développés.

# Tri par fusion

## Algorithme

Voici l’algorithme de tri par fusion tiré du livre *Introduction à l’algorithmique*2:

**FUSION**(A, p, q, r)

n1 ← q − p + 1

n2 ← r − q

**créer tableaux** L[1 . . n1 + 1] et R[1 . . n2 + 1]

**pour** i ← 1 à n1

**faire** L[i] ← A[p + i − 1]

**pour** j ← 1 à n2

**faire** R[j] ← A[q + j]

L[n1 + 1]←

R[n2 + 1]←

i ← 1

j ← 1

**pour** k ← p à r

**faire** si L[i] <= R[j]

**alors** A[k] ← L[i]

i ← i + 1

**sinon** A[k] ← R[j]

j ← j + 1

**TRI-FUSION**(A, p, r)

**si** p < r alors

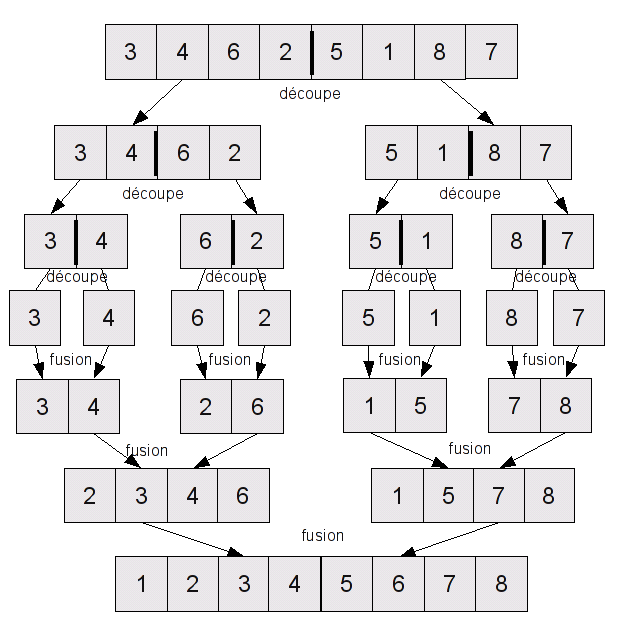
q ← (p + r)/2

**TRI-FUSION**(A, p, q)

**TRI-FUSION**(A, q + 1, r)

**FUSION**(A, p, q, r)

Cet algorithme fonction par la stratégie « *Divide and conquer »*. Le tri par fusion fonctionne en coupant les données en deux jusqu’à ce qu’il ne reste plus que des nombre isolés. Par la suite, il compare deux par deux les blocs de nombres puis les met en ordre pour former à la fin un seul bloc trié.



Sur la figure ci-dessus3, on voit une représentation du fonctionnement de l’algorithme de tri par fusion

## Analyse théorique

Les baromètres sont :

Si on analyse *Tri-Fusion(A,p,r)*, on trouve que c’est une fonction récursive. Si le nombre de données est en bas de 2, le nombre d’opérations effectuées est proche de 1. Si le nombre de données est plus grand ou égal à deux, les données sont séparés en deux moitié et renvoyer récursivement. Une fois, que la récursivité est terminée, elle passe dans une fonction de fusion.

Cet algorithme suit donc le modèle suivant :

Trouvons l’ordre de :

Les baromètres de la fonction sont :

\*\*Attention, N est toujours au minimum à 2.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Avec a=2 et b=2, si

## Analyse expérimentale

L’analyse expérimentale a été effectuée selon le protocole indiqué dans les instructions du laboratoire 1.

Lorsqu’on observe les performances selon N qui est le nombre de données, on remarque que l’asymptote supérieure et inférieure suit une courbe qui semble être de l’ordre de N. Si on observe davantage le graphique, on peut voir un peu que l’ordre est plutôt de N\*log(N), mais les données sont tellement grandes que la différence n’est pas beaucoup perceptible.

Lorsqu’on observe les performances selon R et D qui est l’intervalle de données et le degré de désordre, on remarque que peut importe l’intervalle de donnée, le nombre d’instructions est indépendant de R et D.

Il est aussi a noter que cet algorithme a été celui qui a été le plus rapide à exécuter parmi tous les algorithmes développés.

# Tri par pigeonnier

## Algorithme

Le pseudo-code de cet algorithme à été créé à la main depuis les informations données au cours d’ELE440. Le principe étant de créer une case mémoire pour chacune des valeurs possibles dans le tableau reçu puis compter combien de fois on peut retrouver chaque valeur dans le tableau.

**procédure** tri\_pigeonnier(Tableau de données, Nombre de données, Valeur maximum)

**Pour** la valeur maximale d’un nombre dans le tableau

Initialiser cette valeur à zéro.

**Pour** le nombre de données dans le tableau

Placer la valeur dans son pigeonnier respectif

**Pour** tous les chiffres du pigeonnier

**Tant** qu’il y a des chiffres dans le pigeonnier

Placer la valeur directement après la dernière valeur placée dans le tableau trié.

Incrémenter de 1 le chiffre à remplacer dans le tableau à retourner.

**fin procédure**

## Analyse théorique

Les baromètres sont :

On peut remarquer rapidement que les formules asymptotiques minimales et maximales pour le tri par pigeonnier sont pareilles. Ceci signifie que peu importe le degré de désordre, il y aura toujours le même nombre d’instructions utilisées. Aucune condition n’altère le cours de cet algorithme. C’est un algorithme non-efficace puisqu’il ne s’adapte pas aux données qui lui sont imposés et donc prend toujours le temps maximal pour trier un tableau. Cependant, la formule asymptotique est R2+NR (R étant le rang des données du tableau), donc plus petit le rang, plus rapide l’algorithme. Bref, le tri par pigeonnier est un mauvais algorithme de tri mais peut être plus que d’autres lorsque le rang est petit.

|  |  |  |
| --- | --- | --- |
| Ligne | Min | Max |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Analyse expérimentale

Pour les tests du tri par pigeonnier, nous avons diminuer l’intervalle des données, car lorsque l’intervalle était trop grand, l’application « plantait ».

On peut remarquer ici que le nombre de données ne fait qu’augmenter linéairement le nombre d’instructions et donc de temps.

On remarque avec ce graphique que quand le rang augmente, le nombre d’instructions augmente rapidement à chaque incrémentation.

On remarque que peu importe le degré de désordre, l’algorithme garde le même nombre d’instruction dépendamment du nombre de données à traiter.

# Tri par tas

## Algorithme

Le pseudo-code de cet algorithme à été créé à la main depuis les informations données dans le cours d’ELE440. Malheusement, dû à un algorithme qui n’est pas optimal, ce tri est plutôt lent.

POUR le nombre de données que possède le tableau,

Pour le nombre de nœuds restant à trier,

Si le dernier nœud Possède deux feuilles,

S'il faut échanger la feuille de droite

Échanger la valeur de la feuille de droite avec la valeur du nœud.

Sinon s'il faut échanger avec la feuille de gauche

Échanger la valeur de la feuille de gauche avec la valeur du nœud

FIN DU SI

Sinon si le nœud ne possède qu'une feuille,

S'il faut échanger avec la feuille de gauche

Échanger la valeur de la feuille de gauche avec la valeur du nœud

FIN DU SI

FIN DU SI

Tant que je ne suis pas remonté au premier nœud,

Si la valeur du nœud permuté est plus grande que celle du nœud supérieur,

Permuter avec la valeur supérieure

FIN DU SI

Remonter d'un nœud.

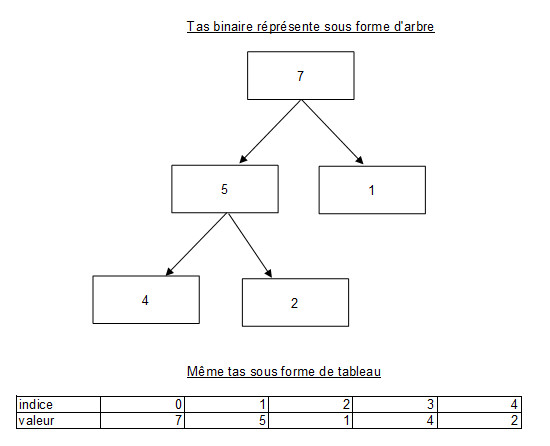
FIN DU TANT QUE

Remplacer la première valeur avec la dernière valeur.

Bloquer la dernière valeur permutée à la fin du tableau.

FIN DU POUR

Le principe de cet algorithme est de créer un arbre en divisant toujours les valeurs en 2 sous-valeurs créant des nœuds et des feuilles. La première donnée du tableau est à la pointe de l’arbre. Tous les nœuds doivent avoir une valeur plus grande que leurs sous-nœuds ou feuilles. En appliquant cette loi à tous les nœuds en partant de la pointe, échanger les valeurs des nœuds avec ceux d’en dessous puis à chaque fois qu’un nombre est échangé, vérifier que le chiffre échangé est toujours plus bas que le nœud du haut. Une fois terminé, la valeur de la pointe devrait être la valeur la plus grande du tableau. Il faut maintenant permuter la première valeur du tableau et la dernière pour envoyer un très petit chiffre à la pointe. Le but étant de sortir un tableau en ordre croissant, la plus grande valeur est maintenant à la fin du tableau. Il est important de faire abstraction des valeurs placées à la fin dans le reste de l’algorithme pour ne pas faire remonter des valeurs déjà triés. Puis on répète le processus d’échange du dessus de la pyramide autant de fois qu’il y a de données.



Sur la figure ci-dessus4, on voit une représentation du fonctionnement de l’algorithme de tri par tas.

## Analyse théorique

L’analyse asymptotique du tri par tas donne une formule plutôt grande ce qui sous-entend une efficacité moindre. N2\*(int(racine(N))) décrit le plus grand nombre de fois que l’algorithme passera à un seul endroit. On peut remarquer avec la table des instructions minimum et maximum qu’il est possible de sauter des instructions grâce aux conditions. Cependant, aucune boucle se trouve à l’intérieur d’une condition donc il doit quand même passer à travers les trois boucles même si les données sont déjà tous triés. En ne regardant que les boucles, on remarque que les minimums et maximums sont pareil ce qui n’est pas efficace. Pour que ce soit rapide, l’algorithme doit être capable de détecter rapidement si le tableau est déjà trié.

|  |  |  |
| --- | --- | --- |
| ligne | Min | Max |
| 1 | N | N |
| 2 | N\*N | N\*N |
| 3 | N\*N | N\*N |
| 4 | 0 | N\*N |
| 5 | 0 | N\*N |
| 6 | 0 | N\*N |
| 7 | 0 | N\*N |
| 9 | N\*N | N\*N |
| 10 | 0 | N\*N |
| 11 | 0 | N\*N |
| 14 | N\*(N+1) | N\*N\*(int(racine(N)+1)) |
| 15 | 0 | N\*N\*(int(racine(N))) |
| 16 | 0 | N\*N\*(int(racine(N))) |

## Analyse expérimentale

Pour les tests du tri par tas, pour éviter que le test ne dure trop longtemps, nous avons diminué le nombre de données pour accélérer les tests.

On peut constater que la performance de cet algorithme détériore rapidement avec des grand nombre de données à trier.

On voit sur le graphique ci-haut que le rang n’affecte pas la performance de cet algorithme.

Le graphique ci-haut illustre bien que le degré de désordre n’a pas d’effet sur l’algorithme.

# Tri rapide

## Algorithme

Voici l’algorithme de tri par rapide tiré de la page Wikipedia de cet algorithme5:

**partitionner**(tableau T, entier premier, entier dernier, pivot)

{

**échanger** T[pivot] et T[dernier]

j :=premier

**pour** i de premier à dernier - 1

{

si T[i] <= T[dernier] alors

échanger T[i] et T[j]

j := j + 1

}

échanger T[dernier] et T[j]

**renvoyer** j

}

**tri\_rapide**(tableau T, entier premier, entier dernier)

**début**

**si** premier < dernier **alors**

pivot := choix\_pivot(T, premier, dernier)

pivot := partitionner(T, premier, dernier, pivot)

tri\_rapide(T, premier, pivot-1)

tri\_rapide(T, pivot+1, dernier)

**fin si**

**fin**

Tout d’abord, cet algorithme sépare les données en plus petit et plus grand qu’un index qui servira de pivot. Une fois les données séparées, on tombe en récursivité en refaisant cette étape mais pour chacune des deux moitiés. L’algorithme continue jusqu’à ce que les données soient triées.

## Analyse théorique

Cet algorithme est un algorithme récursif qui ressemble à :

Trouvons l’ordre de :

Les baromètres de la fonction sont :

\*\*Attention, N est toujours au minimum à 2.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Avec a=2 et b=2, si

## Analyse expérimentale

Le tri rapide semble être plus efficace avec de plus petit nombres de données.

Ce graphique démontre bien comment cet algorithme préfère avoir un rang plus grand pour trier ces données.

Pour ce qui est du degré de désordre, le tri rapide ne semble pas avoir de corrélation pertinente.

# Tri par base

## Algorithme

Cet algorithme est basé sur des observations d’une visualisation de son fonctionnement sur Internet.6 Par contre, on y a ajouté la possibilité de gérer les nombres négatifs. Un deuxième tableau a été créé pour gérer ces nombres négatifs.

Pour toutes les puissances de 10

Initialiser le tableau de comptage à zéro.

Pour tout les nombres à trier

Trouver le chiffre de la puissance de 10 correspondantes

Incrémenter de 1 la case correspondant du tableau de compteurs

Pour toutes les cases du tableau de compteurs

Additionner la valeur de la case (i) avec celle de la case adjacente (i+1))

Pour toutes les données à trier,

Trouver le chiffre de la puissance de 10 correspondantes

Diminuer la valeur de la case du tableau de compteurs de 1

Placer la donnée complète dans la nouvelle matrice à la case précisé par le nombre de la case du tableau de compteur.

Analyse théorique - Tri par base

Le tri par base utilise des puissances de 10 pour itérer à travers chaque chiffre des nombres à trier. Il commence avec les unités, puis les dizaines, centaines, milliers, etc. En vérifiant chacune des valeurs, il incrémente un compteur correspondant au chiffre trouvé. Par la suite, on additionne la valeur du premier compteur avec la valeur de la case adjacente jusqu’à la case 9. (EX : case[0] + case[1]; case[1]+case[2]; ….)

La prochaine étape est de retrouver le chiffre utiliser précédemment mais cette fois-ci, on décrémente le conteur de 1. Le chiffre qui est maintenant la valeur du compteur représente la case d’un nouveau tableau de même taille que le tableau désordonné, dans laquelle il faut placer le nombre. On répète le tout avec les autres chiffres (puissances de 10) de chaque nombre pour trouver un tableau trié à la fin.

## Analyse théorique

D’après l’analyse théorique, le tri par base utilise le même nombre d’instructions peu importe le désordre et le rang des données qui lui sont envoyé. Il est cependant important de mentionner que sa formule asymptotique est R\*N ce qui est un excellent résultat pour de grand nombre de données et le rang n’est pas nécessairement ce qui ralentira l’algorithme

|  |  |  |
| --- | --- | --- |
| ligne | Min | Max |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Analyse expérimentale

Nous pouvons constater que le temps de l’algorithme augmente linéairement dépendamment du nombre d’éléments à trier.

Le tri par base itère à travers chaque chiffre d’un nombre. On peut voir grâce au graphique ci-haut que le temps d’exécution est un peu plus grand a chaque fois que le rang du test augmente.

La figure ci-haut nous permet de constater la constance dans l’algorithme totalement indépendant du degré de désordre.

# Conclusion

En bref, ce rapport compare l’efficacité de 6 algorithmes de tri dépendamment de la taille des données, du nombre de données et du degré de désordre des données. Il contient aussi les pseudo-codes écrits pour construire les algorithmes qui nous permettent de calculer la fonction asymptotique de chaque méthode. 3 graphiques pour chacune des méthodes illustrent la relation entre le nombre d’instructions et le nombre de données à traiter, la grosseur des données à traiter ainsi que le degré de désordre des donnés. Pour conclure, le tri insertion est le pire; le tri pigeonnier est efficace uniquement avec un petit rang; le tri rapide est efficace lorsque la taille des données et que le degré de désordre est grand; Le tri fusion semble être le plus efficace de tous en tout et pour tout; le tri par tas n’a pas été conçu de la bonne manière or les données affichés ne sont pas les bonnes; Le tri par base est efficace mais le temps de processus grandit linéairement avec la taille des données à trier. Certainement, il manque encore plusieurs algorithmes de tri comme, par exemple le tri à bulle qui n’était pas demandé dans ce laboratoire.

# Annexe

## Références

1 :Wikipedia. Tri par insertion,[En ligne], https://fr.wikipedia.org/wiki/Tri\_par\_insertion (Consulté le 1er octobre 2015).

2 : Cormen,T., Leiserson,C., Rivest, R., Stein, C.(2001). Introduction à l’algorithmique (2e ed.). MIT : DUNOD.

3 : OpenClassrooms. Introduction au problème de tri,[En ligne]. https://openclassrooms.com/courses/algorithmique-pour-l-apprenti-programmeur/introduction-au-probleme-du-tri (Consulté le 1er octobre 2015).

4 : OpenClassrooms. Tri par tas, [En ligne]. https://openclassrooms.com/courses/le-tri-par-tas (Consulté le 1er octobre 2015).

5 :Wikipedia. Tri rapide,[En ligne], https://fr.wikipedia.org/wiki/Tri\_rapide   
(Consulté le 1er octobre 2015).

6 : University of San Francisco : Department of computer science. Radix Sort,[En ligne], https://www.cs.usfca.edu/~galles/visualization/RadixSort.html (Consulté le1er octobre 2015).