**INF4705 – Analyse et conception d’algorithmes**

TP1 – Hiver 2018

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Tremblay, David, 1748125  Souli, Taleb, 1770491 |
| **Note finale / 13** | 0 |

**Informations techniques**

* Répondez directement dans ce document ODT avec LibreOffice. Veuillez ne pas inclure le texte en italique servant de directive.
* La correction se fait à même le rapport.
* Avant votre troisième séance de laboratoire, vous devez faire une remise électronique en suivant les instructions suivantes:
  + Le dossier remis doit se nommer matricule1\_matricule2\_tp1 et doit être compressé sous format zip.
  + À la racine de ce dernier, on doit retrouver :
    - Ce rapport sous format ODT.
    - Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
    - L’exemplaire *ex.txt*. Voir énoncé plus bas.
    - Un fichier texte nommé *emails.txt* contenant le courriel de chaque membre de l’équipe
    - Le code source et les exécutables
* Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. Le code et les exécutables soumis devront être compatible avec les ordinateurs de la salle L-4714.
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.
* Votre chargé de laboratoire vous encourage très fortement à lire le guide intitulé “guide bash” sur moodle pour faire vos graphiques.

**Mise en situation**

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l’analyse empirique et hybride des algorithmes. À la section 3.2 des notes de cours, trois approches d’analyse de l’implantation d’un algorithme sont décrites. Vous les mettrez en pratique pour des algorithmes de tri.

Vous implanterez 3 algorithmes de tri:

1. Insertion sort
2. Mergesort
3. Mergesort + seuil de récursivité

De plus, vous devrez comparer ces algorithmes avec *std::sort* de la librairie standard en C++.

Pour l’algorithme 3, vous devez déterminer un seuil de récursivité expérimentalement. Les exemplaires dont la taille est en deça de ce seuil ne sont plus résolus récursivement mais plutôt directement avec le insertion sort.

**Jeu de données**

Pour tester les algorithmes, vous devez générer un jeu de données avec 10 exemplaires

pour les tailles: 1000, 5000, 10000, 50000, 100000 et 500000. Vous pouvez utiliser le script suivant:

for n in {"1000","5000","10000","50000","100000","500000"}; do

for i in {1..10}; do

shuf -i 1-$n > testset\_${n}\_${i}.txt

done

done

*Mettez ce script dans un fichier (disons* gen.sh*), donnez lui les permissions avec* chmod +x gen.sh*, puis exécuter ce programme dans un nouveau dossier pour générer votre dataset.*

**Présentation des résultats**

|  |  |
| --- | --- |
| 0 | / 4 pt |

**Tableaux des résultats**

**Tableau des résultats - InsertionSort**

|  |  |
| --- | --- |
| ***Taille*** | ***Temps (s)*** |
| *1000* | *0,0002212* |
| *5000* | *0,0039746* |
| *10000* | *0,0129139* |
| *50000* | *0,2851531* |
| *100000* | *1,1489517* |
| *500000* | *29,1804184* |

**Tableau des résultats - MergeSort**

|  |  |
| --- | --- |
| ***Taille*** | ***Temps (s)*** |
| *1000* | *0,0001162* |
| *5000* | *0,0005657* |
| *10000* | *0,0011424* |
| *50000* | *0,0052115* |
| *100000* | *0,0092589* |
| *500000* | *0,0465252* |

**Tableau des résultats - MergeSort avec seuil**

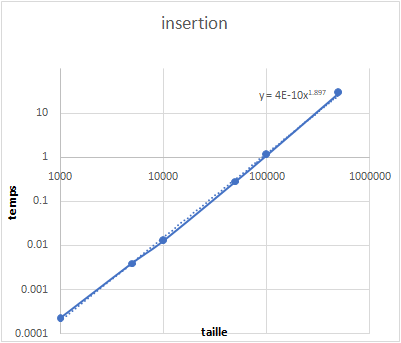
|  |  |
| --- | --- |
| ***Taille*** | ***Temps (s)*** |
| *1000* | *0,0001178* |
| *5000* | *0,0006286* |
| *10000* | *0,0010989* |
| *50000* | *0,0049619* |
| *100000* | *0,0091543* |
| *500000* | *0,0468879* |

**Tableau des résultats - stdSort**

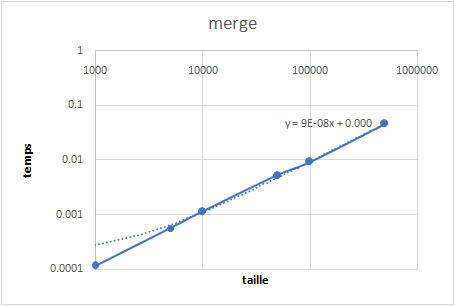
|  |  |
| --- | --- |
| ***Taille*** | ***Temps (s)*** |
| 1000 | 0,0000688 |
| 5000 | 0,0003831 |
| 10000 | 0,0007742 |
| 50000 | 0,0031555 |
| 100000 | 0,0060048 |
| 500000 | 0,0300084 |

**Tests de puissance**

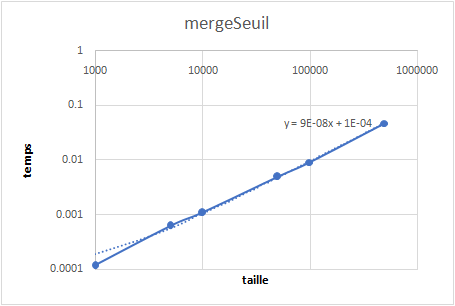
**Test de puissance - InsertationSort**

**

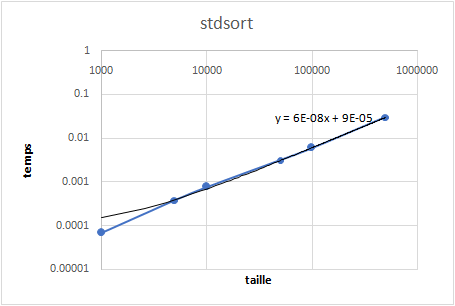
**Test de puissance - MergeSort**

**

**Test de puissance - MergeSort avec seuil**

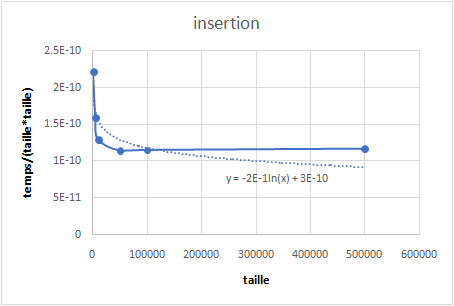
**

**Test de puissance - stdSort**

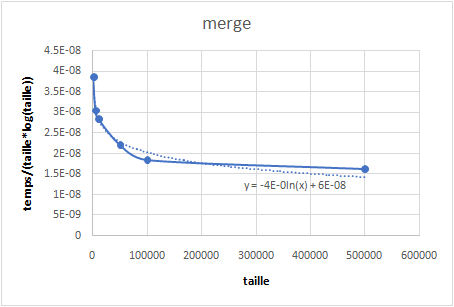
**

**Test du rapport**

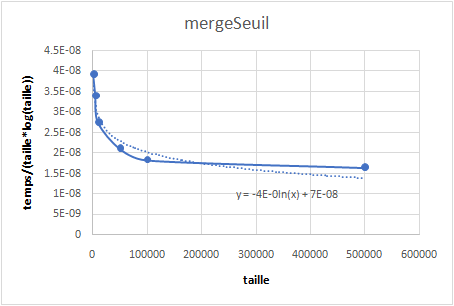
**Test du rapport - InsertionSort**

**

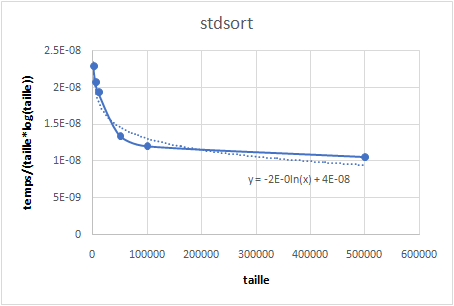
**Test du rapport - Merge**

**

**Test du rapport - MergeSort avec seuil**

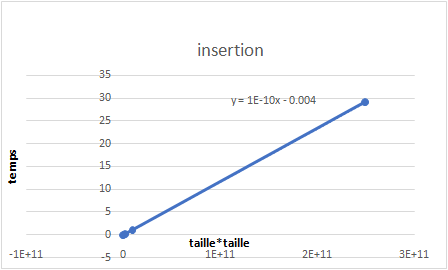
**

**Test du rapport - stdSort**

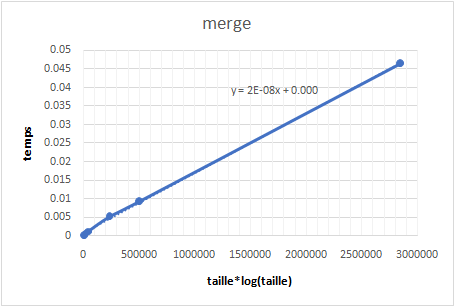
**

**Test des constantes**

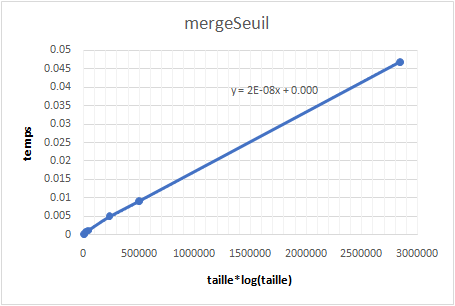
**Test des constantes - InsertionSort**

**

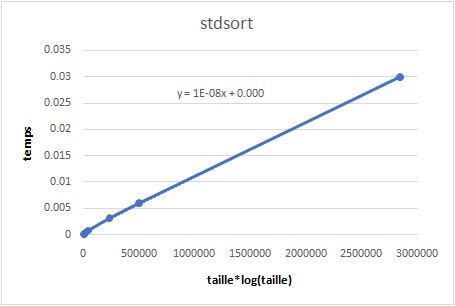
**Test des constantes - MergeSort**

**

**Test des constantes - MergeSort avec seuil**

**

**Test des constantes - stdSort**

**

**Analyse et discussion**

|  |  |
| --- | --- |
| 0 | / 6 pt |

**Que pouvez-vous déduire du test de puissance?**

Le test de puissance est une courbe linéaire où les axes X=taille des échantillons et Y = temps de calcul de l’algorithme pour faire le tri des échantillons. Ainsi, ces tests donnent des courbes linéaires d'équation Y=bf(x)+c. Dans cette perspective, les tests de puissance reflètent la performance de chaque algorithme en terme de complexité temporelle consommé pour effectuer le tri de chaque exemplaire, ce qui nous donne une idée sur la rapidité d’un algorithme par rapport aux autres.

Nous avons constaté qu’il ya une différence entre MergeSort et MergeSortSeuil, ce dernier est plus rapide puisque nous utilisons un seuil pour changer de *merge* vers *insertion sort* lorsque la taille de l’exemplaire à trier est plus bas que 19 éléments, ce seuil a été déterminé expérimentalement selon la rapidité entre les algorithmes Merge et InsertionSort. Nous avons constaté que insertion sort était plus rapide pour les petits échantillons. Nous avons donc choisi un seuil = 19 qui donnait les meilleurs résultats. On constate aussi que InsertionSort est beaucoup plus lent que tous les autres lorsque la taille des exemplaires augmente.

**Citez la consommation théorique du temps de calcul pour les algorithmes, en notation asymptotique.**

Soit n la taille de l’exemplaire à trier.

1. **Insertion Sort ∈** O(n2)
2. **Merge Sort ∈** O(n(log(n))
3. **Merge Sort avec seuil ∈** O(n(log(n))
4. **std Sort ∈** O(n(log(n))

**Que pouvez-vous déduire du test du rapport?**

Nous avons émis les hypothèses qui correspondent à la consommation théorique du temps de calcul des algorithmes tel que décrit ci-haut. Par conséquent, nous avons obtenues des courbes logarithmiques qui tendent vers une constante b que nous avons utilisé pour déterminer l'équation des différents algorithmes selon l'équation y=b\*f(x) + c. Ainsi, le fait que la courbe tend vers une constante est une preuve de l’exactitude de notre hypothèse de départ.

**Axes des graphiques**: x=taille, y=temps/f(x)

1. **Insertion Sort:** f(x) = taille\*taille
2. **Merge Sort:** f(x) = taille\*log(taille)
3. **Merge Sort seuil:** f(x) = taille\*log(taille)
4. **std Sort:** f(x) = taille\*log(taille)

**Que pouvez-vous déduire du test des constantes?**

Nous avons émis les hypothèses qui correspondent à la consommation théorique du temps de calcul des algorithmes tel que décrit ci-haut. Selon ces hypothèses, nous avons obtenues des courbes linéaires qui croisent l’axe y en une constante b et qui ont une pente m, selon l'équation y = m\*f(x) + b.

**Axes des graphiques**: x=f(x), y=temps

1. **Insertion Sort:** f(x) = taille\*taille
2. **Merge Sort:** f(x) = taille\*log(taille)
3. **Merge Sort seuil:** f(x) = taille\*log(taille)
4. **std Sort:** f(x) = taille\*log(taille)

**Discutez de l’impact du seuil de récursivité.**

MergeSort est généralement plus rapide pour trier des échantillons toutefois InsertionSort est plus rapide pour les échantillons de petite taille. L’idée du seuil est donc de trier les échantillons jusqu’à une taille où InsertionSort est plus rapide pour utiliser cet algorithme par la suite et terminer le tri. Nous constatons l’impact direct de l’utilisation du seuil sur la performance de l’algorithme MergeSortSeuil qui est devenu plus rapide que MergeSort. Pour atteindre ce résultat, nous avons passé de *merge* vers *insertion sort* lorsque la taille de l’exemplaire à trier est plus bas que 19 éléments. Ce seuil a été déterminé expérimentalement en regardant la rapidité entre les algorithmes MergeSort et InsertionSort. Ce dernier est plus rapide pour des échantillons entre 5 et 20. À cette fin, nous avons choisi un seuil = 19. En conséquence, l’algorithme MergeSortSeuil est devenu légèrement plus rapide que MergeSort.

**Suite à cette analyse, indiquez sous quelles conditions (taille d’exemplaire ou autre) vous utiliseriez chacun de ces algorithmes. Justifiez.**

Cette analyse nous a montré que l’algorithme *merge* et plus rapide lorsque la taille des exemplaires est élevée. Néanmoins, l’algorithme InsertionSort prend la relève pour devenir plus rapide que les autres pour des échantillons de taille minime. Dans cette optique, nous choisissons l’algorithme idéal selon la taille des exemplaires à trier. Il est par contre plus efficace de toujours utiliser MergeSort avec seuil puisque l’algorithme se charge d’utiliser l’algorithme approprié selon la taille de l’échantillon.

**Créer un exemplaire pour lequel un de vos trois algorithmes est plus rapide que** *std::sort***. Pourquoi est-il plus rapide?**

*Vous devez inclure l’exemplaire à la racine de la remise sous le nom “ex.txt”.*

**Autres critères de correction**

**Respect de l’interface tp.sh**

|  |  |
| --- | --- |
| 0 | / 1 pt |

Utilisation

tp.sh -a [insertion | merge | mergeSeuil | stdsort] -e [path\_vers\_exemplaire]

Arguments optionnels

-p affiche les nombres triés, sans texte superflu

-t affiche le temps d’exécution en ms, sans unité ni texte superflu

Important: l’option -e doit accepter des fichiers avec des paths absolus.

**Qualité du code**

|  |  |
| --- | --- |
| 0 | / 1 pt |

**Présentation générale**

|  |  |
| --- | --- |
| 0 | / 1 pt |

* Concision
* Qualité du français

**Pénalité retard**

|  |
| --- |
| 0 |

* -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.