INF8775 – Analyse et conception d'algorithmes

TP1 – Automne 2023

Nom, prénom, matricule des membres	Mouton, Hugo, 2310478 Deloumeau, Nicolas, 2086143
Note finale / 13	0

Informations techniques

- Répondez directement dans ce document .docx. Veuillez ne pas inclure le texte en italique servant de directive. La correction se fait sur ce même rapport.
- Vous devez faire une remise électronique sur Moodle avant le 10 octobre à 23h55 pour le groupe 1, 3 octobre à 23h55 pour le groupe 2 et 5 octobre à 23h55 pour le groupe 3.
- Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve:
  - Ce rapport sous format docx.
  - Un script nommé tp.sh servant à exécuter les différents algorithmes du TP. L'interface du script est décrite à la fin du rapport.
  - 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. Notez que le code et les exécutables soumis seront testés sur les ordinateurs de la salle L-4714 et doivent être compatibles avec cet environnement. En d'autres mots, tout doit fonctionner correctement lorsque le correcteur exécute votre script tp.sh sur un des ordinateurs de la salle.
- Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.
- On vous encourage à lire le guide intitulé « guide bash » sur Moodle pour faire vos graphiques. C'est un guide qui a été conçu pour un ancien TP, mais il contient beaucoup d'informations utiles.

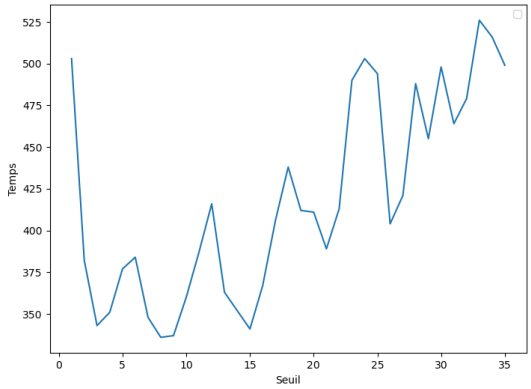
Mise en situation

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 les algorithmes de tri par dénombrement et de tri rapide (quicksort). Un rappel de ces algorithmes sera fait lors de la séance de laboratoire. Vous ferez trois versions de l'algorithme de tri rapide :

1. Pivot sur le 1er élément et seuil de récursivité à 1
2. Pivot sur le 1er élément et seuil de récursivité déterminé expérimentalement
3. Pivot sur un élément aléatoire et seuil de récursivité déterminé expérimentalement

Les exemplaires dont la taille est en deçà du seuil de récursivité ne sont plus résolus récursivement mais plutôt directement avec un algorithme de tri de votre choix prenant un temps dans $\Theta(n^2)$ en pire cas (tri par insertion, par sélection, à bulles, etc.)



A: Nous avons tracé le temps d'exécution de notre algorithme en fonction du seuil de récursivité imposé (taille de l'ensemble à partir de laquelle on change de sort). Nous avons donc choisi de fixer notre seuil de récursivité à 9 dans la mesure où après différents tests, c'est autour de 9 que le temps d'exécution semble être le plus faible (malgré les oscillations présentes sur la figure).

Jeu de données

Vous trouverez dans l'archive du TP un script python *gen.py* servant à générer les exemplaires. Ce script s'exécute de la manière suivante :

gen.py -S TAILLE [-n NB_EXEMPLAIRES] [-m MAGNITUDE] [-r RANDOM_SEED] [-p PSEUDO-TRI]

TAILLE correspond à la taille des exemplaires.

NB_EXEMPLAIRES correspond au nombre de suites que vous voulez générer pour chaque taille

MAGNITUDE correspond à la valeur maximale possible dans les exemplaires. Par exemple, pour -m 100000, les exemplaires comportent des nombres de 0 à 100000.

RANDOM_SEED correspond à la *seed* utilisée pour la génération aléatoire des listes de nombres.

PSEUDO-TRI est une variable binaire qui détermine si l'exemplaire sera presque trié.

Présentation des résultats

0	/ 4 pt

Tableau des résultats

Série 1

Algorithme	Temps (us)				
Taille	1000	3000	10000	30000	100000
Dénomb.	41.5	118.8	398.8	1090	3422.5
QS s=1	276.3	980.9	3652.9	11777.4	40775.5
QS s=9	257.2	891.7	3129.5	10204.1	35135
QS s=9, al.	271.3	827.1	3034	8533.7	35212.4

Série 2

Algorithme	Temps (us)				
Max	1000	3000	10000	30000	100000
Dénomb.	393.1	414.8	580.3	1056.1	2229

Série 3

Algorithme	Temps (us)				
Taille/Max	1000/10	3000/100	10000/1000	30000/10000	1e5/1e5
Dénomb.	393.1	414.8	580.3	1056.1	2229

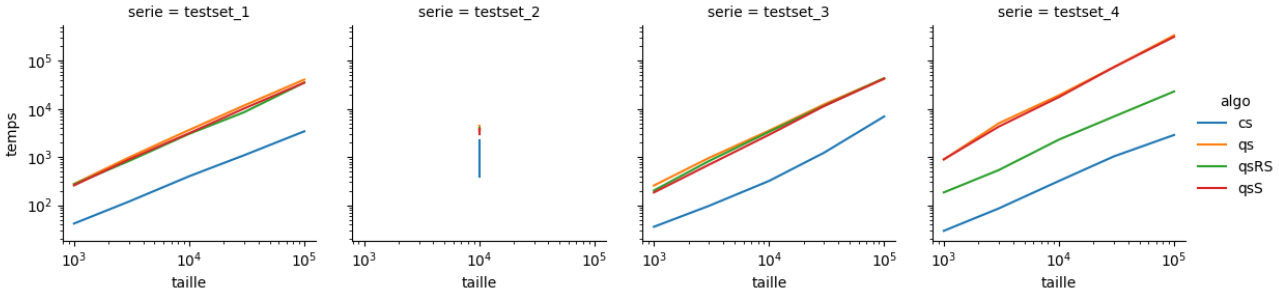
Série 4

Algorithme	Temps (us)				
Taille	1000	3000	10000	30000	100000
QS s=1	886.3	5027	19161.7	74148.5	337678.9
QS s=9	896.6	4283.6	17629.3	73589	315308.8

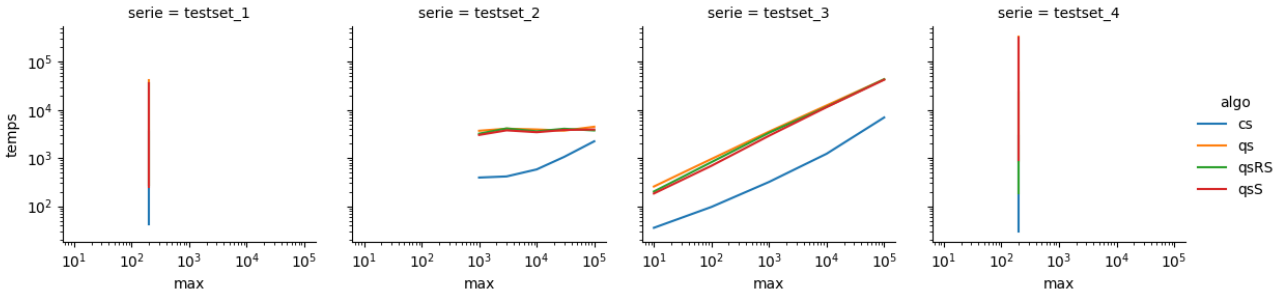
QS s=9, al.	184	536	2305	6942.7	23055.9

Tests de puissance

Pour chacun des algorithmes et chacune des séries d'exemplaires, appliquez le test de puissance et rapportez les graphiques ici.



Test de puissance: Temps en fonction de Taille



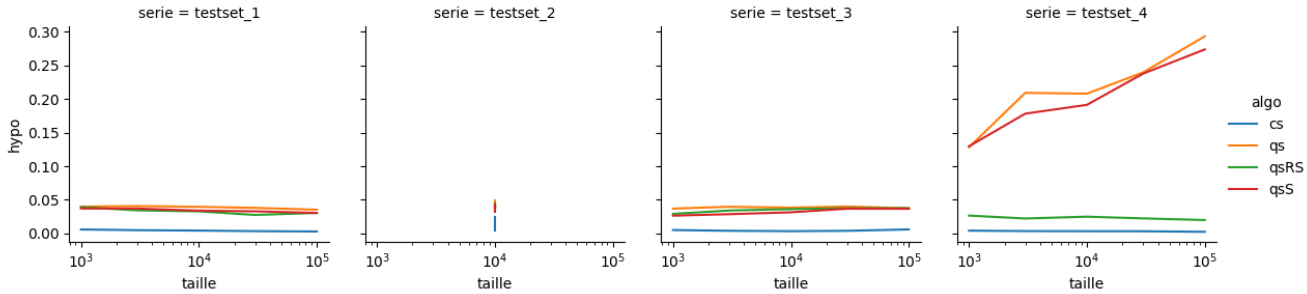
Test de puissance: Temps en fonction de Max

Test du rapport

Pour chacun des algorithmes et chacune des séries d'exemplaires, appliquez le test du rapport en utilisant chacune des analyses en meilleur cas, en pire cas et en moyenne et rapportez les graphiques ici.

Taux de croissance s'apparente à: *f(taille) = taille*log(taille)*

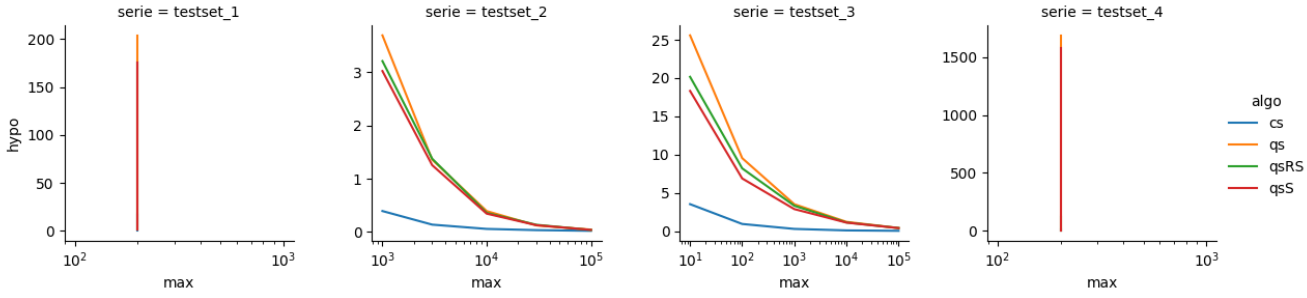
***hypo** correspond à: y / f(taille)*



Test du rapport: Hypo en fonction de Taille

Taux de croissance s'apparente correspond ici à: *f(max) = max*

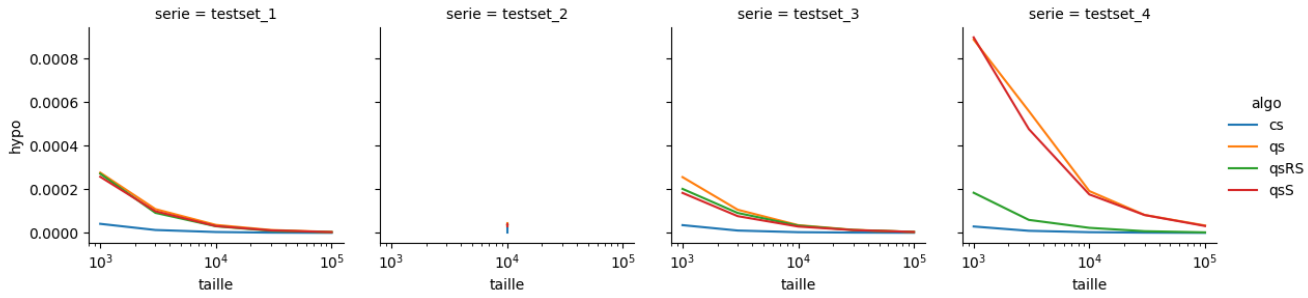
***hypo** correspond à: y / f(taille)*



Test du rapport: Hypo en fonction de Max

Taux de croissance s'apparente correspond ici à: *f(taille) = taille²*

***hypo** correspond à: y / f(taille)*



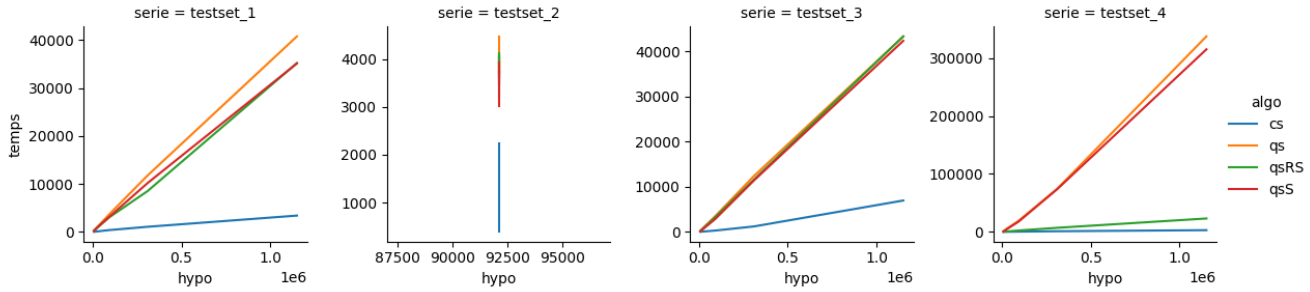
Test du rapport: Hypo en fonction de Max

Test des constantes

Pour chacun des algorithmes et chacune des séries d'exemplaires, appliquez le test des constantes et rapportez les graphiques ici. Dans chaque cas, choisissez l'analyse (meilleur cas, pire cas, cas moyen) la plus appropriée selon les résultats du test précédent (test du rapport).

Taux de croissance s'apparente à: *f(taille) = taille*log(taille)*

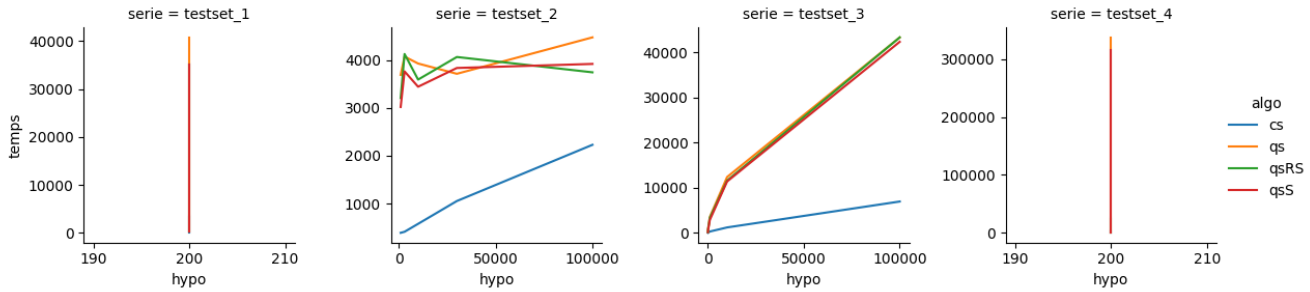
***hypo** correspond à: f(taille)*



Test des constantes: Temps en fonction de Hypo

Taux de croissance s'apparente à: *f(max) = max*log(max)*

***hypo** correspond à: f(max)*



Test des constantes: Temps en fonction de Hypo

Analyse et discussion

0	/ 6 pt

Que pouvez-vous déduire du test de puissance?

Le test de puissance nous indique que les algorithmes sont d'une complexité polynomiale. Le test de puissance du testset 2 n'est pas concluant, car l'algorithme dépend à la fois de la taille et de la valeur maximum.

Citez la consommation théorique en meilleur cas, en pire cas et en moyenne du temps de calcul pour les algorithmes, en notation asymptotique.

Nul besoin de faire une preuve, on demande seulement de citer.

	Pire Cas	Moyen Cas	Meilleur Cas
Counting	Θ(n+m)	Θ(n+m)	Θ(n+m)
Quick el=1,r=1	Θ(n²)	Θ(n log n)	Θ(n log n)
Quick el=1,r=9	Θ(n²)	Θ(n log n)	Θ(n log n)
Quick el=random,r=9	Θ(n²)	Θ(n log n)	Θ(n log n)

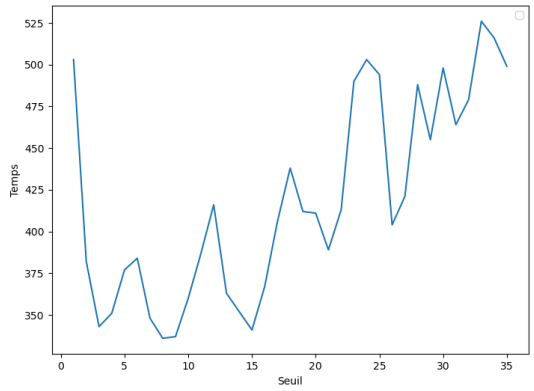
Que pouvez-vous déduire du test du rapport?

Le test du rapport permet de déterminer que le taux de croissance de l'algorithme s'apparente à n*log(n), car celui-ci converge. Il n'est cependant pas possible de déterminer celui du counting sort, car celui-ci dépend à la fois de la valeur maximum et de la taille de l'exemplaire.

Que pouvez-vous déduire du test des constantes?

On peut déduire que l'algorithme quicksort évolue de manière linéaire à la fonction n*log(n). Il est aussi possible de voir que dans le testset 2, l'algorithme counting sort évolue de manière linéaire par rapport à la valeur maximum, ce qui est attendu, car il doit parcourir un plus grand tableau.

Discutez de l’impact du seuil de récursivité.



Les performances du quicksort dépendent beaucoup de la quantité d’appels récursifs que nous autorisons. Si le seuil est fixé trop bas, les mauvaises performances du bubble sort prennent le dessus. Si le seuil est trop élevé, c’est le grand nombre de comparaisons du quicksort qui réduisent les performances de notre algorithme.

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

Dans la majorité des cas, l’algorithme quicksort est le meilleur pour effectuer un tri dans un espace mémoire et temps raisonnable. Avec un bon réglage du seuil de récursivité et un bon choix du pivot, ce tri est le plus performant. Par contre, le tri par dénombrement est seulement avantageux lorsque l’on manipule des valeurs entières. Cependant, plus l’intervalle de valeurs est grand et plus la consommation mémoire est importante.

Autres critères de correction

Respect de l'interface tp.sh

0	/ 1 pt

Utilisation

tp.sh -a [counting | quick | quickSeuil | quickRandomSeuil] -e [path_vers_exemplaire]

Arguments optionnels

-p affiche les nombres triés en ordre croissant sur une ligne, 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.