**INF8775 – Analyse et conception d’algorithmes**

Rapport TP1 – Hiver 2022

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Pucci-Barbeau, Vincenzo, 1948994  Bossert, Matthieu, 2161168 |
| **Note finale / 30** | 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 23 Février à 23h59 en suivant les instructions suivantes :
  + Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
    - Ce rapport au 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.
    - Si le langage que vous utilisez nécessite une phase de compilation, veuillez joindre un Makefile afin que nous puissions le compiler en cas de problème avec vos exécutables. Si nous ne sommes pas en mesure de tester votre code, vous perdrez des points de respect d’interface et de qualité de code !
* 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 compatibles 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.

# Présentation des résultats

|  |  |
| --- | --- |
|  | / 1,5pts |

## Tableau des résultats

Tableau 1 : Algorithme Bruteforce

|  |  |
| --- | --- |
| Nombre de buildings | Temps moyen en millisecondes |
| 1000 | 1.31598 |
| 5000 | 6.43138 |
| 10 000 | 12.81298 |
| 50 000 | 78.6941 |
| 100 000 | 164.60524 |
| 500 000 | 954.7173 |

Tableau 2 : Algorithme Diviser-pour-régner

|  |  |
| --- | --- |
| Nombre de buildings | Temps moyen en millisecondes |
| 1000 | 6.11326 |
| 5000 | 34.83088 |
| 10 000 | 72.57494 |
| 50 000 | 422.2147 |
| 100 000 | 912.17818 |
| 500 000 | 6117.4487 |

Tableau 3 : Algorithme Diviser-pour-régner avec seuil de récursivité

|  |  |
| --- | --- |
| Nombre de buildings | Temps moyen en millisecondes |
| 1000 | 3.19174 |
| 5000 | 19.27264 |
| 10 000 | 43.60108 |
| 50 000 | 300.58978 |
| 100 000 | 631.94126 |
| 500 000 | 4564.48648 |

# Analyse et discussion

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

|  |  |
| --- | --- |
|  | / 1,5 pt |

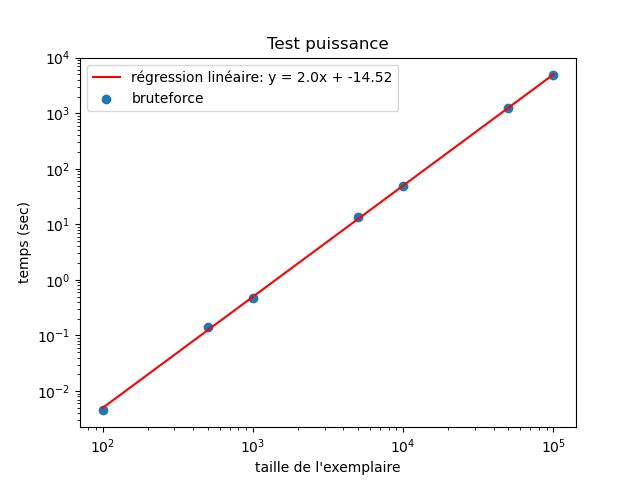
Bruteforce :

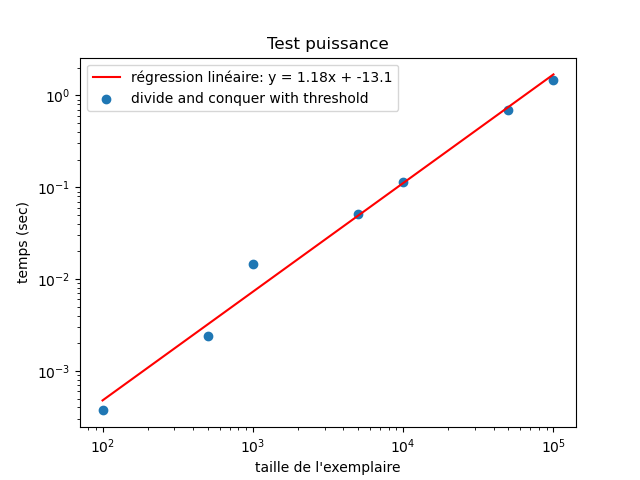
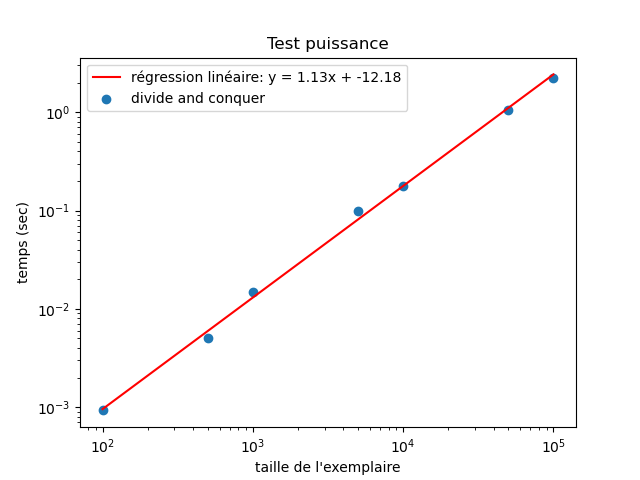
Diviser-pour-régner :

Diviser-pour-régner avec seuil :

## Tests de puissance

|  |  |
| --- | --- |
|  | / 5 pt |



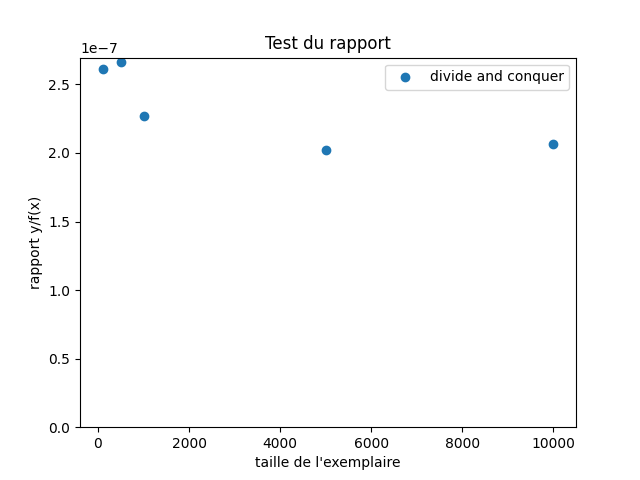
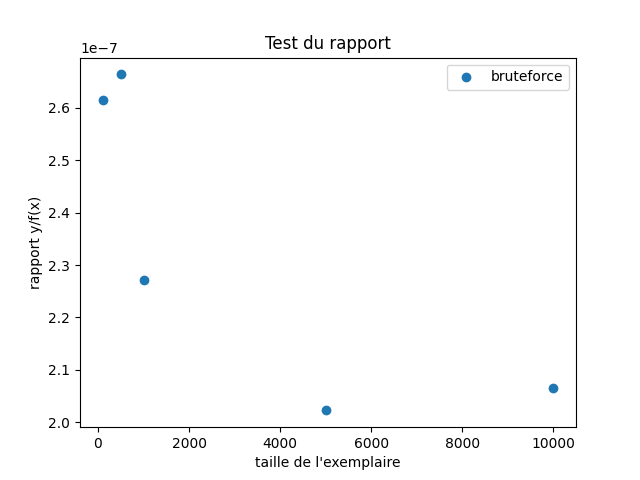


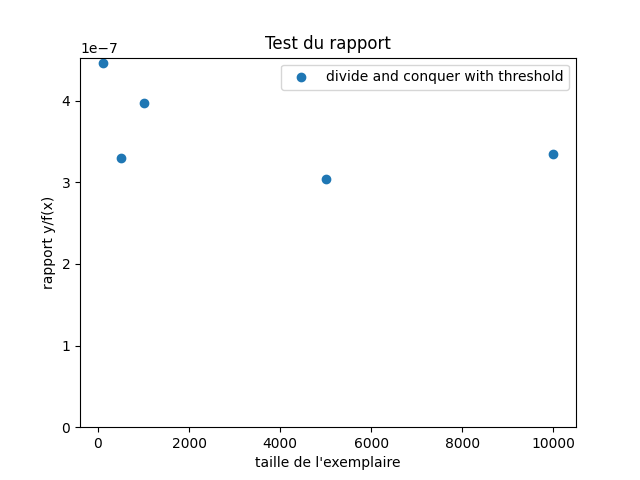
## Que pouvez-vous déduire du test de puissance ?

1. La pente de la courbe affiche le degré de complexité d’un algorithme polynomial.
2. On vérifie bien ici que la méthode bruteforce est en O(n^2) car la pente de la régression est environ égale à 2.
3. Pour les méthodes récursives O(n\*lg n), la pente est légèrement supérieur à 1 car le terme log(n) apporte une croissance non négligeable. En effet, la régression n’est pas tout à fait une droite.

## Test du rapport

|  |  |
| --- | --- |
|  | / 5 pt |



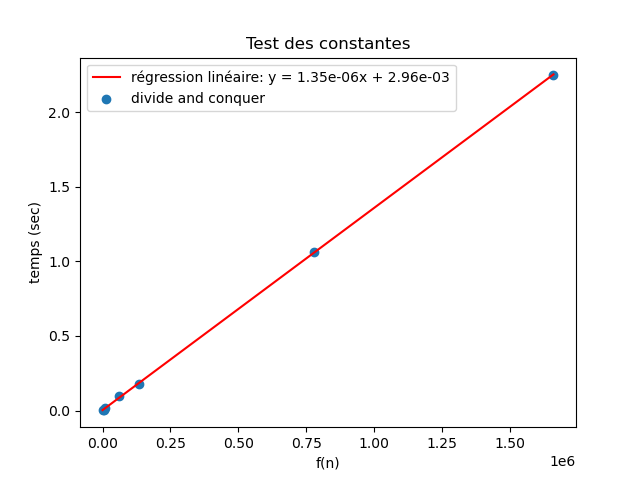
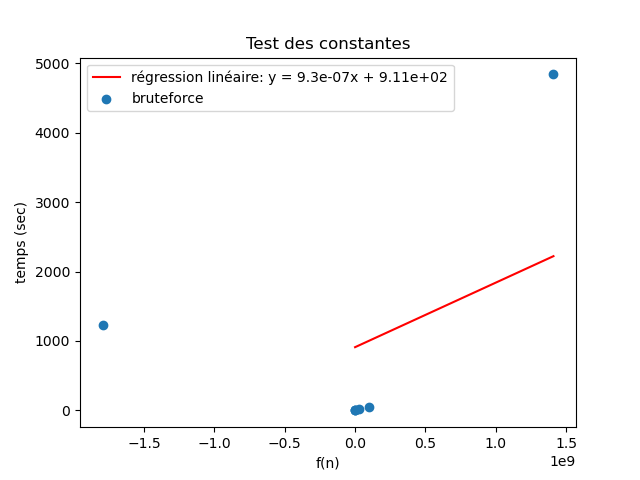


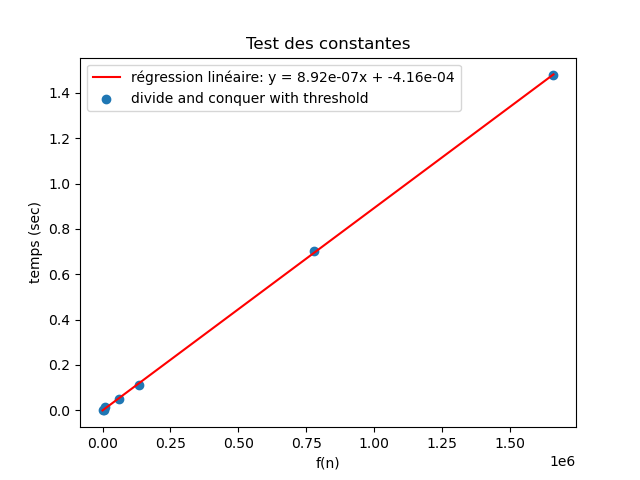
## Que pouvez-vous déduire du test du rapport ?

* 1. La fonction bruteforce converge vers une valeur de 2. Ainsi, on peut en déduire que notre constante multiplicative pour notre équation de complexité est égale à 2.
  2. La fonction diviser pour régner converge vers une valeur de 2. Ainsi, on peut en déduire que notre constante multiplicative pour notre équation de complexité est égale à 2.
  3. La fonction diviser pour régner avec un seuil converge vers une valeur de 3. Ainsi, on peut en déduire que notre constante multiplicative pour notre équation de complexité est égale à 3.

## Test des constantes

|  |  |
| --- | --- |
|  | / 5 pt |





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

Formule de base pour le test des constantes :

Dans le cas de bruteforce, on a une constante multiplicative approximativement égale à :

Dans le cas de diviser pour régner et de diviser pour régner avec seuil, le comportement asymptotique est similaire. Cependant la constante de la version avec seuil est environ moitié moindre que la version sans seuil. La version avec seuil est donc en pratique environ deux fois plus rapide

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

|  |  |
| --- | --- |
|  | / 2 pt |

* 1. L’impact théorique est nul. Peu importe le seuil, tant que celui-ci est fini, la consommation de l’algorithme de force brute sur un échantillon de taille inférieure ou égale au seuil demeure constante. La complexité asymptotique n’est donc pas impactée.

Dans la pratique cependant, l’appel de fonction récursif peut s’avérer moins efficace en dessous d’une certaine taille d’échantillon (environ 20). La version avec seuil donne des performance expérimentalement deux fois meilleures par rapport à la version sans seuil

## Suite à cette analyse, indiquez sous quelles conditions (taille d’exemplaire ou autre) vous utiliseriez chacun de ces algorithmes. Prenez en compte la complexité spatiale et temporelle, le temps de calcul et la difficulté d'implémentation. Justifiez.

|  |  |
| --- | --- |
|  | / 3 pt |

Pour un petit échantillon (<1000), l’algorithme naïf est à la fois plus simple d’implémentation, et les différences de performances ne sont pas notables. La pile n’est pas remplie par les appels récursifs de la fonction ce qui minimise le risque de manquer d’espace mémoire.

Pour un gros échantillon (>1000) la complexité temporelle de l’algorithme diviser pour régner est meilleure, ce qui rend l’algorithme significativement plus rapide. Cependant, il faut également faire attention à l’espace mémoire nécessaire pour les appels récursifs de la fonction qui croient en O(lg n). L’implémentation est également plus complexe car il faut traiter certains cas particuliers.

Enfin, en pratique, l’implémentation avec seuil donne les meilleurs résultats. Le seul inconvénient par rapport à diviser pour régner est la nécessité de programmer les deux méthodes. Toutefois, l’algorithme naïf reste relativement simple à implémenter. Le seuil peut également différer d’une machine à l’autre.

# Autres critères de correction

## Respect de l’interface tp.sh

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

## Qualité du code

|  |  |
| --- | --- |
|  | / 5 pt |

* + - * 1. Validité des solutions
        2. Qualité de l'implémentation

Présence de commentaires

## Présentation générale

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

* Concision
* Qualité du français

## Pénalité retard

|  |
| --- |
| 0 |

* -15% de la note / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.