Rapport - Projet d'Algorithmique et Structures de Données

Arthur LEBEURRIER, Quentin JURDIC

Pour le Dimanche 24 Avril 2022

1. Choix effectués

Dans un premier temps, nous aurions pu penser à un alogirthme classique, de type "naïf" (il est tout de même implémenté dans le programme).

Il s'agit de l'algorithme le plus banal. Bien qu'il soit efficace pour de petites valeurs de n, on se rend compte que les temps d'exécution sont très élevés pour de grandes valeurs de n (coût en $\boxed{O(n^2)}$.

Nous avons donc décidé d'améliorer notre code, en faisant appel à une méthode de type "diviser pour reigner", comme vu en cours.

En effet, ce système est très efficace pour réduire les temps d'exécution lorsque n devient grand.

L'idée est la suivante :

- Créer deux tableaux T_x et T_y, triés par abscisses et ordonnées;
- Quand $n > 3$, partager l'ensemble des points par une droite verticale, de sorte à en avoir autant de chaque côté de la droite (à 1 près);
- On utilise alors une procédure récursive, qui va récupérer le minimum des 2 distances minimales obtenues;
- Enfin, on compare ce minimum à celui obtenu parmi les couples de points qui sont de deux côtés distincts (un à gauche, et un à droite).

On obtient alors un coût en $\boxed{O(nlog(n))}$.

2. Étude des différentes fonctions

a) *recherche_naive*

Cet algorithme correspond au plus basique possible.

Il consiste à étudier les distances entre tous les couples de points possibles :

- À chaque tour de boucle, on regarde si la distance calculée entre les points i et j est plus petite que la distance minimale stockée (*distance_mini*). Si tel est le cas, on met à jour cette distance minimale, et on modifie la valeur de la liste F, composée des coordonnées des points de distance minimale.
- À la fin de la fonction, on retourne cette distance minimale d_{min}, le couple de points concerné et le nombre de points P présents au départ.

Ainsi, le coût de cette fonction est tel que :

- On a $\sum_{i=1}^{n} i = \frac{n(n-1)}{2}$ tours de la boucle entre les lignes 55 et 59.
- Les autres coûts de la fonction sont en $O(1)$.

Ainsi, le coût de la fonction *recherche_naive* est en $\boxed{O(n^2)}$ (complexité quadratique).

Vérification graphique :

On effectue le test_naif() du code.

Celui-ci permet de mesurer le temps d'exécution (en secondes), en fonction de n, pour n variant de 10 en 10.

Voici le résultat obtenu pour un nombre de mesures égal à 2000 (au-delà, les temps d'exécution deviennent vraiment très longs) :

[Mesure de performance dans le premier cas]

Au vu de cette courbe (même si l'on a quelques perturbations), on se rend bien compte que la courbe a une forme de parabole.

Ainsi, la complexité est, comme prévu, en $\boxed{O(n^2)}$

b) $fusion$ **et** tri_fusion

La fonction tri_fusion correspond à celle étudiée en cours.

Elle construite (par récursivité), ainsi :

- Cas de base : lorsque notre tableau a une taille inférieure à 1, on le renvoie;
- Cas récursif : on divise notre tableau en 2 autres de taille égale à 1 près (gauche et droite), auxquels on va appliquer à nouveau les programmes de fusion.
- Puis, on applique l'algorithme de fusion à ces deux tableaux créés.

La fonction $fusion$, elle, permet de rassembler les différents sous-tableaux créés.

Son coût est de l'ordre de $\boxed{O(n)}$ au pire cas.

Coût de cette fonction :

On appelle deux fois la fonction, sur des tableaux de taille deux fois plus petite. Et on rajoute un coût en $O(n)$.

Ainsi :

$$\boxed{C(n) = 2 \cdot C\left(\frac{n}{2}\right) + O(n)}$$

Or, $\log_2(2) = 1$, donc on est dans le cas n°2 du Master Theorem, et :

$$\boxed{C(n) = O(nlog(n))}$$

On retrouve bien le résultat du cours.

Remarque : *on cherche ici à avoir un coût faible pour de grandes valeurs de n.*

Si l'on avait voulu réduire au maximum le coût, et ce y-compris pour de petites valeurs de n, on aurait pu proposer un algorithme sous forme itérative, qui aurait alors été plus efficace que la procédure récursive pour les petites valeurs de n.

c) $reigne$ **et** fct

la fonction reigne est construite par récursivité:

- Cas de base: lorque le tableau de points contient 3 points ou moins on applique l'algorithme de recherche naive.
- Cas récursif, on divise notre tableau de point initiale en 2 tableaux de tailles egales (a un point pret) sur lequels on applique notre algorithme fct.

La fonction fct prend en argument deux triplés (dsitance minimale, couple de pointt, ensemble de point) de deux ensembles A et B et renvoie le nouveau triplé de AUB. Ainsi la plus petite distance entre deux points est atteinte, soit entre deux points de A, soit entre deux points de B, soit entre un point de A et un point de B.

On note $\delta = \min(\text{distance\_min(A)}, \text{distance\_min(B)})$ et on sintersse uniquement aux points situés dans la bande verticale de largeur 2δ situés au centre du plan. Il suffit alors de calculer la distance minimale entre les points et leurs 7 suivants (les points etant triés par ordonné croissante)

En effet pour un point p quelquonque de la zone considéré, il y a au plus 7 points situés a un distance inferieur a δ de p.

Ainsi le coup de la fonction est en O(7n) = O(n) dans le pire des cas.

On appelle deux fois la fonction, sur des tableaux de taille deux fois plus petite. Et on rajoute un coût en $O(n)$.

Ainsi :

$$\boxed{C(n) = 2 \cdot C\left(\frac{n}{2}\right) + O(n)}$$

Or, $\log_2(2) = 1$, donc on est dans le cas n°2 du Master Theorem, et :

$$\boxed{C(n) = O(nlog(n))}$$

On retrouve bien le résultat théorique.

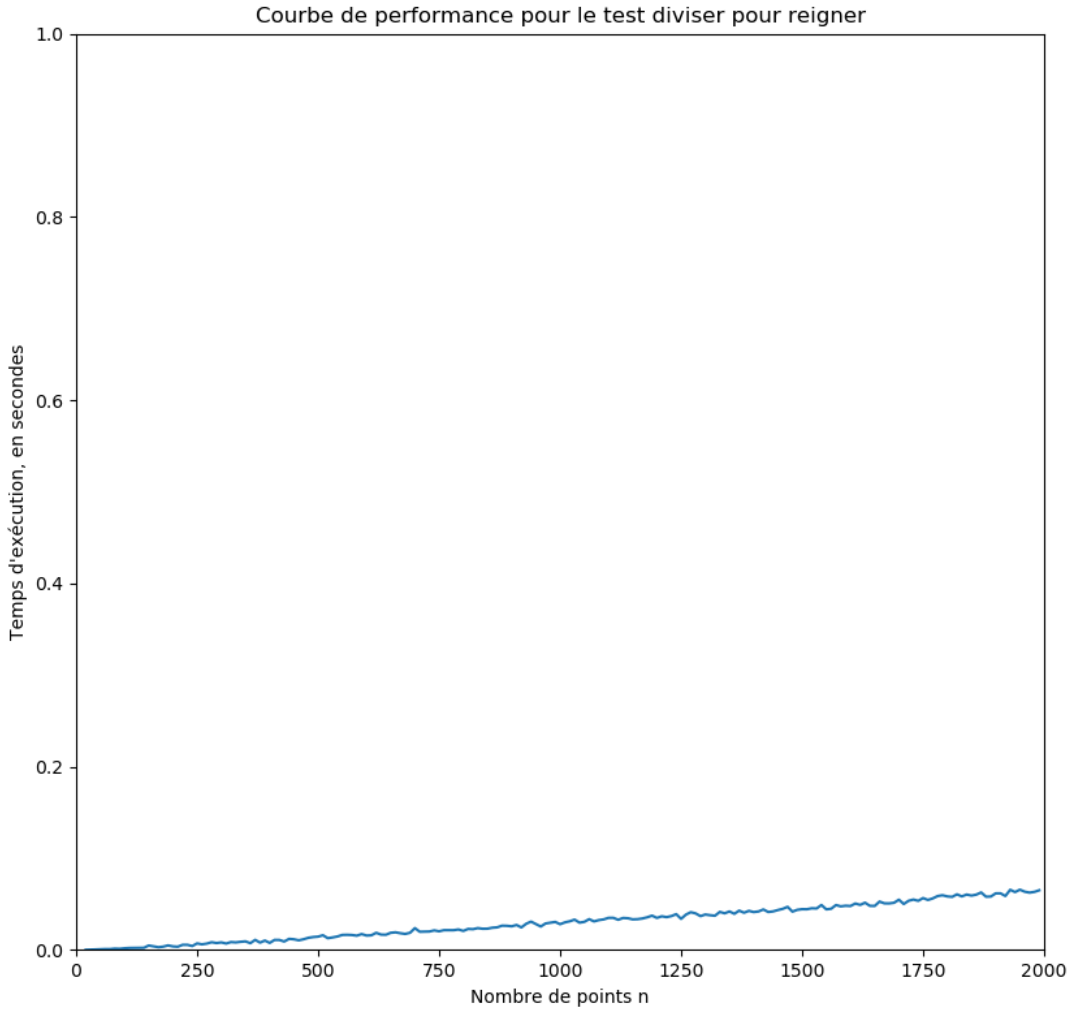

Figure 2: Mesure de performance dans le cas diviser pour reigner

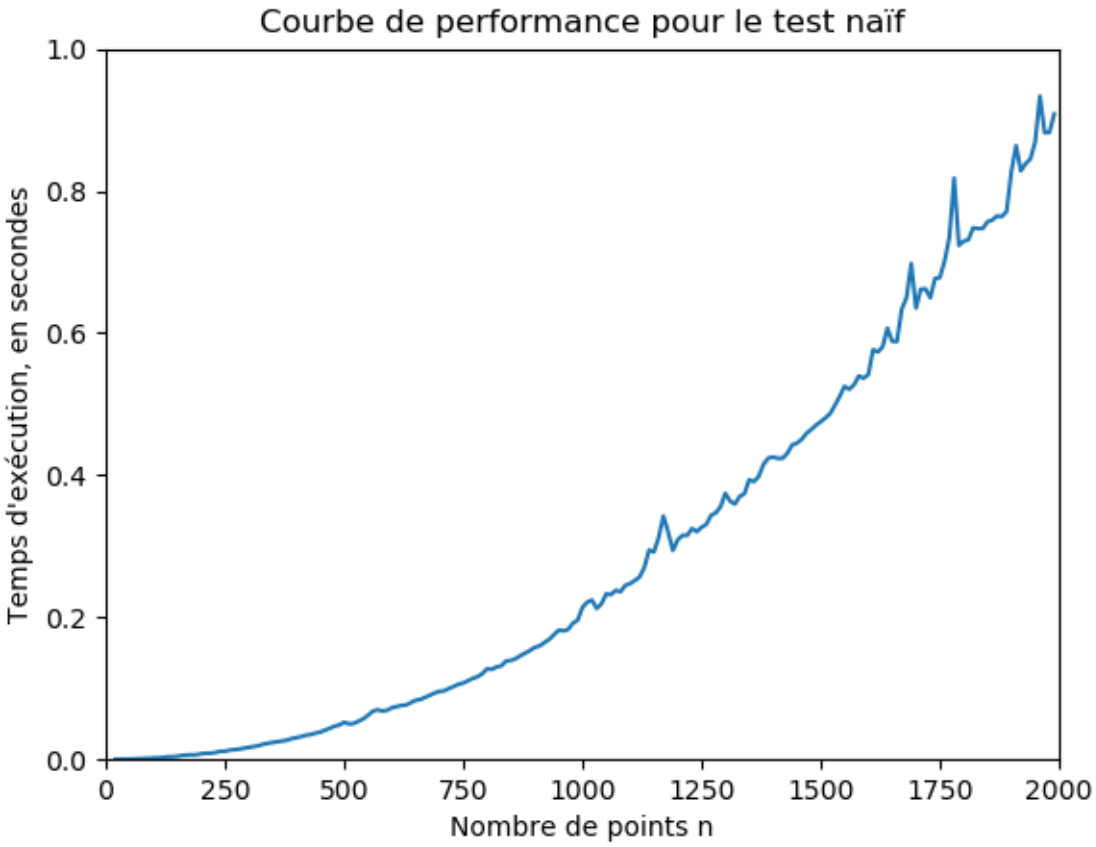

Figure 1: Mesure de performance dans le premier cas