bo

|  |
| --- |
| Projet de complexité |
| Pour le 18 Avril 2017 |
| Lieffroy Mervine, Lemaitre Martin, Calley Nicolas, Keller Rowane, Cominolo Théo |

# Introduction

Ce document a pour but de présenter notre travail pour le projet de Complexité de l’année 2016/2017. Une description du programme réalisé y est présente, ainsi que nos réponses à toutes les questions du sujet.

# Description du programme

## La classe Balayage :

Cette classe permet de réaliser la comparaison par balayage. Elle possède une liste de cercle sur laquelle la comparaison va se réaliser.

* Les méthodes importantes :

**public** **void** addCercle(Cercle c) : Permet d’ajouter les cercles dans le liste de cercle présentes dans la classe.

**public** **double** distance(Cercle c1, Cercle c2) : Permet de calculer la distance entre le centre des deux cercles passés en paramètre.

**public** **void** trier () : Permet de trier la liste de cercle par rapport à leur coordonnée x.

**public** **void** comparaison() : Permet de réaliser la comparaison des cercles comme indiqué dans l’énoncé. Au début de l’exécution de cette méthode, la méthode trier() est appelé pour trier la liste de cercle.

## La classe ToutesLesPaires :

Cette classe permet de réaliser la comparaison par ToutesLesPaires.

* Les méthodes importantes

**public** **void** addCercle(Cercle c) : même chose que Balayage.

**public** **double** distance(Cercle c1, Cercle c2) : même chose que Balayage.

**public** **void** comparaison() : Permet de réaliser la comparaison des cercles en les comparants tous entre eux, un par un.

## La classe Cercle :

Cette classe modélise le cercle à partir de coordonnées x et y, d’un rayon, ainsi que d’un numéro permettant l’identification des cercles lors des tests. Les cercles sont comparables par rapport à la coordonnée du point le plus à gauche du cercle.

## La classe Main :

La classe Main nous a permis de réaliser nos différents tests.

* Les méthodes importantes :

**public** **static** ArrayList<Integer> jeu (**int** n, **int** type) : Cette méthode permet de mettre en place chaque liste de cercle présent dans Balayage et ToutesLesPaires (les deux listes étant identiques) en fonction du nombre de cercles désiré (n) et du type de jeu choisis (jeu1, jeu2 ou jeu3). Après la mise en place des listes de cercles en fonction du type de jeu, on applique la comparaison pour les deux classes, puis on récupère les résultats dans une liste que l’on retourne (voir ci-dessous).

**public** **static** String testGlobal(**int** n) : Cette méthode permet de réaliser trois tests pour chaque type de jeu, pour un nombre n de cercle. Les résultats sont récupérés sous la forme d’un String que l’on ajoute par la suite dans un fichier texte « resultat »  pour la réalisation futur des courbes avec R.

# Question 1

Pour notre tri, nous avons utilisé un tri par fusion implémenté dans la classe Collections de java et de complexité nlog(n) d'après la documentation.

<https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html#sort(java.util.List>)

La complexité au pire cas  de ces deux algorithmes est n² car

* Pour ToutesLesPaires, on compare à chaque fois un cercle avec tous les autres cercles. On a une double boucle (« pour » et « tant que ») qui dépend de n, la complexité est donc n² au pire cas (celui où on vérifie chaque disque)
* Pour Balayage, on observe également une double boucle qui dépend de n. La complexité au pire cas est donc, également, n² dans le cas où chaque disque est vérifié.

# Question 2

* 1. K= O(n²), car au pire cas on compare tous les cercles entre eux (n²/2). Par conséquent, k peut être majoré par n².
  2. Le calcul de la liste triée dépend de la coordonnée x de chaque disque. Or, k représente le nombre de croisements entre nos disques et ce facteur n’est pas pris en compte du tout cet algorithme. Par conséquent, la complexité du calcul de la liste triée ne dépend pas de k.
  3. On a la complexité du tri de notre liste L qui est de n log n. D’après l’énoncé, la complexité de l’algorithme Balayage est de

=

Sachant que la somme des ki est égale à k, on obtient la formule suivante pour la complexité de l’algorithme Balayage : n (log(n)) + k + (n-1)

* 1. La complexité obtenue en 1) ne dépend pas de k. Par conséquent, elle ne varie pas si k est grand ou petit.

La complexité obtenue en 2)d dépend de k. Si k est petit, il y aura peu de « croisements » de cercles et k sera inférieur à n log n. Au contraire, si k est grand beaucoup de croisements auront lieu et k sera supérieur à n log n, il pourra même aller jusqu’à n².

# Question 3

Testons nos deux algorithmes pour n=6 :

C1

C4

C5

C2

C3

C6

**Résultat de la comparaison par paire :**

Le cercle [1] croise [2]

Le cercle [1] croise [4]

Le cercle [2] croise [3]

Le cercle [2] croise [4]

Le cercle [2] croise [5]

Le cercle [3] croise [4]

Le cercle [3] croise [5]

Le cercle [5] croise [6]

**Résultat du balayage :**

Le cercle [1] croise [2]

Le cercle [1] croise [4]

Le cercle [2] croise [4]

Le cercle [2] croise [5]

Le cercle [2] croise [3]

Le cercle [4] croise [3]

Le cercle [5] croise [3]

Le cercle [5] croise [6]

C1

C4

C5

C2

C3

C6

C7

**Résultat de la comparaison par paire :**

Le cercle [1] croise [2]

Le cercle [1] croise [3]

Le cercle [1] croise [4]

Le cercle [1] croise [7]

Le cercle [3] croise [5]

**Résultat du balayage :**

Le cercle [1] croise [2]

Le cercle [1] croise [7]

Le cercle [1] croise [3]

Le cercle [1] croise [4]

Le cercle [3] croise [5]

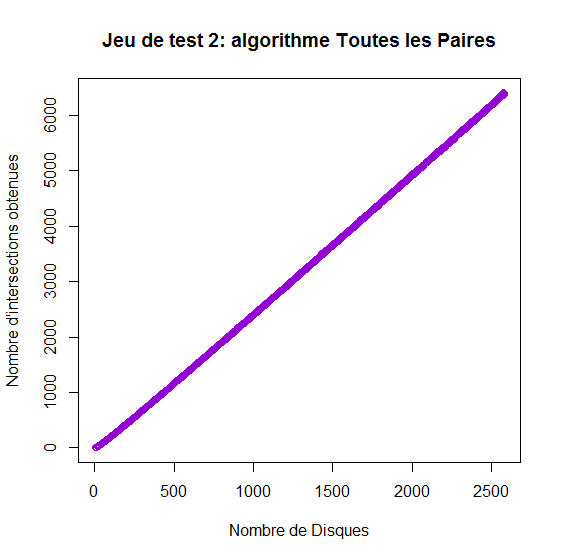
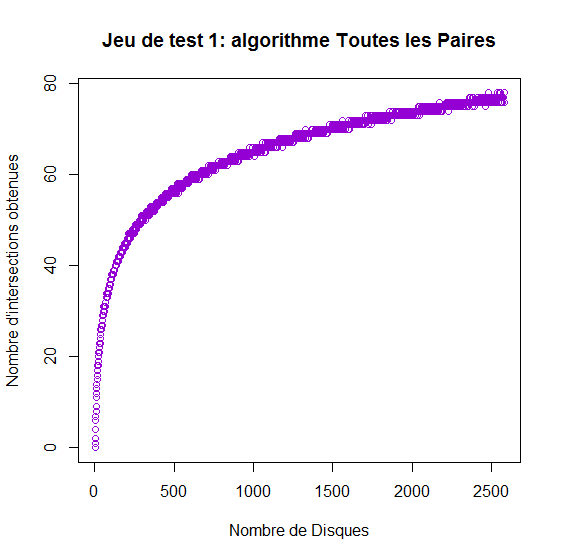
# Question 4

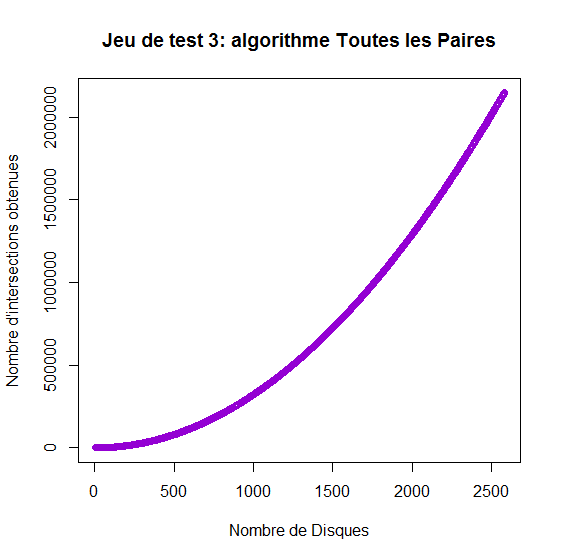
Soit H(n) le nombre d’intersections obtenues par nos deux algorithmes en fonction de n. Commençons par tracer les résultats obtenus par ces derniers en fonction du jeu de test choisi.

Pour rappel :

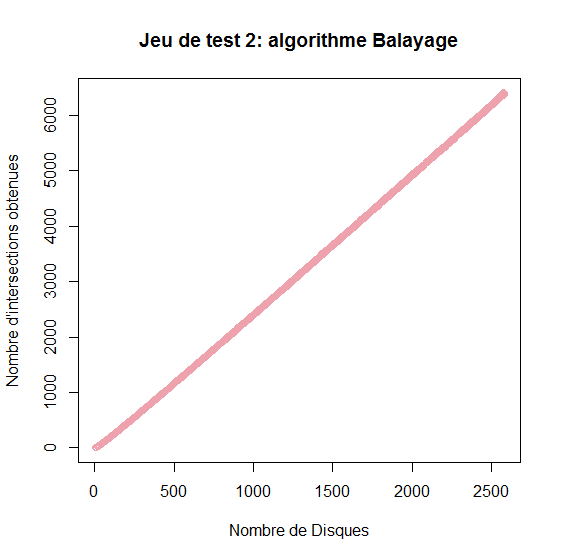
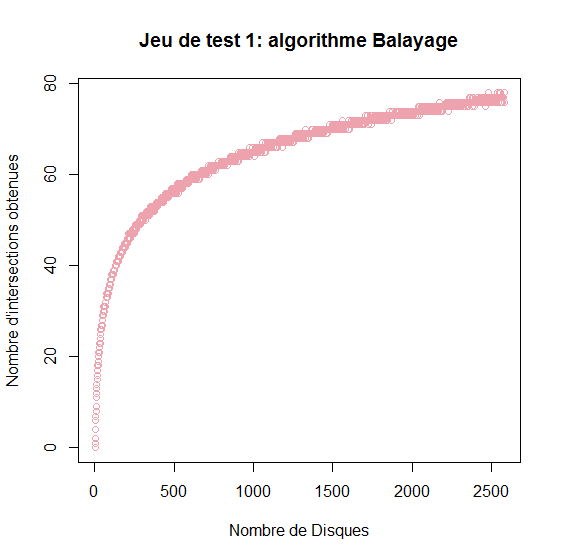
* Dans le jeu de test 1, le rayon de nos cercles est toujours de 1.
* Dans le jeu de test 2, le rayon est déterminé aléatoirement dans l’intervalle [0,].
* Dans le jeu de test 3, le rayon est déterminé aléatoirement dans l’intervalle [0,n].

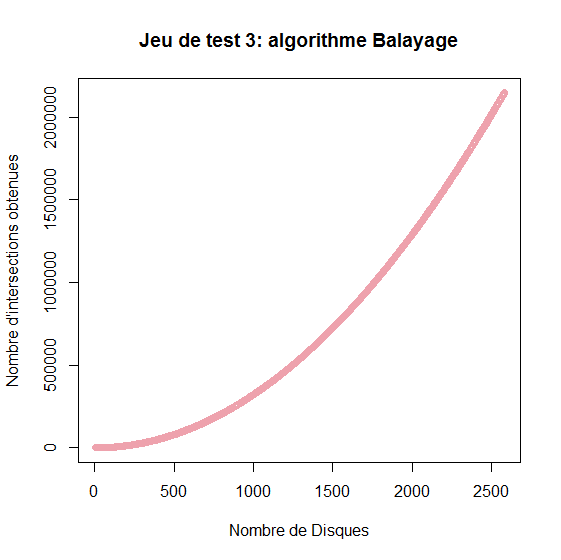
1. Algorithme Toutes Les Paires





1. Algorithme Balayage



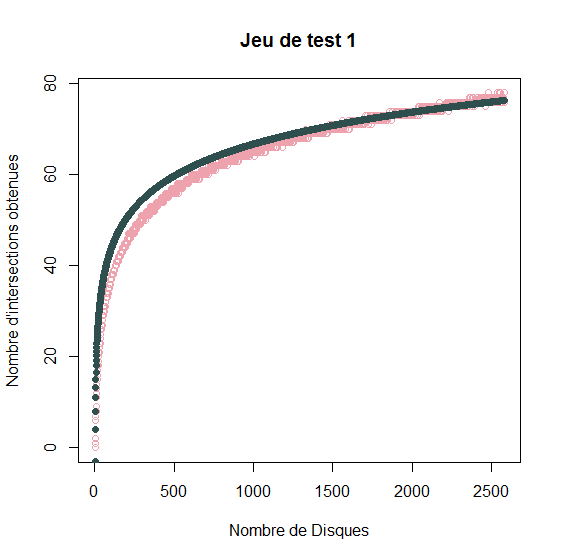


Sur les graphes des trois jeux de tests, les résultats des deux algorithmes sont identiques.

Cherchons maintenant, pour chaque jeu de test, une fonction de type c1F(n)+c2 qui se rapproche le plus de la courbe obtenue. Sur chaque graphique, la courbe obtenue lors de nos tests sera en rose, tandis que la fonction qui s’en rapproche le plus (que nous aurons déterminée) sera en gris.

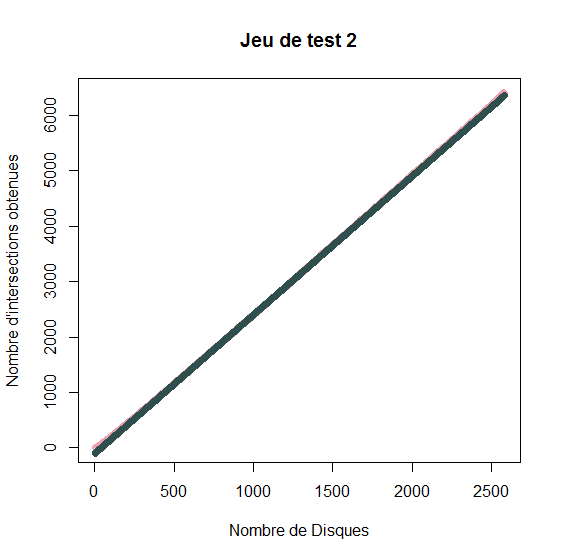
1. Jeu de test 1 : Rayon=1

* F(n)=log(n)
* C1=7
* C2=-3



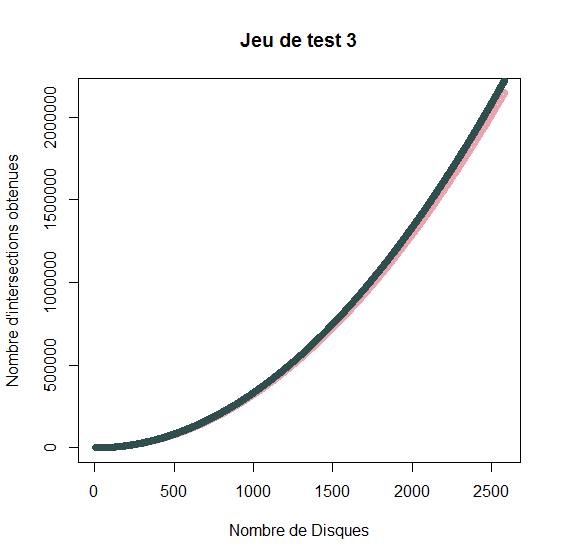
1. Jeu de test 2 : Rayon [0,].

* F(n)=n
* C1=2.5
* C2=-100



1. Jeu de test 3 : Rayon [0,n]

* F(n)=n²
* C1=1/3
* C2=0



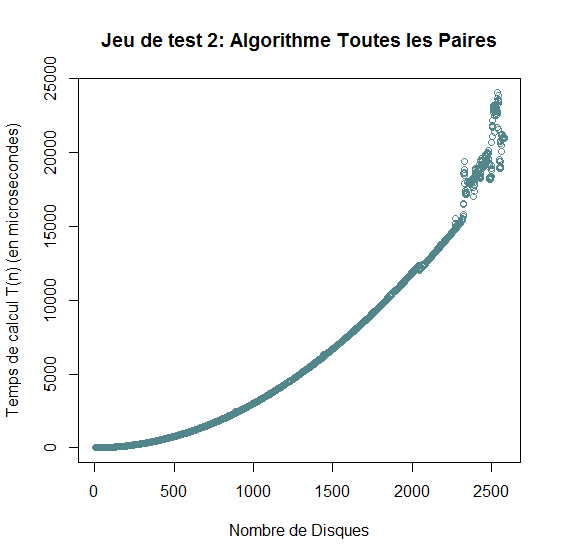
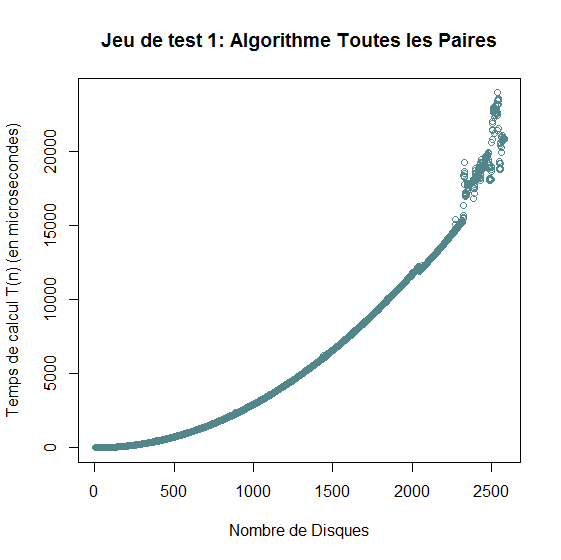
**Conclusion sur la complexité expérimentale de H(n) :**

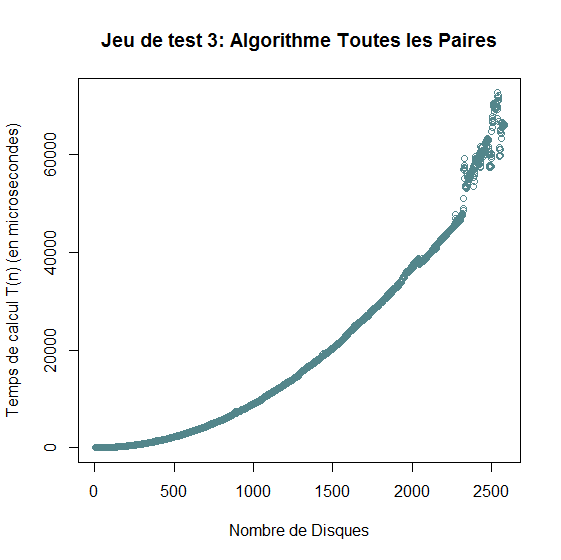
1. Dans le premier jeu de test, H(n)= O(log(n))
2. Dans le deuxième jeu de test, H(n)=O(n)
3. Dans le troisième jeu de test, H(n)=O(n²)

# Question 5

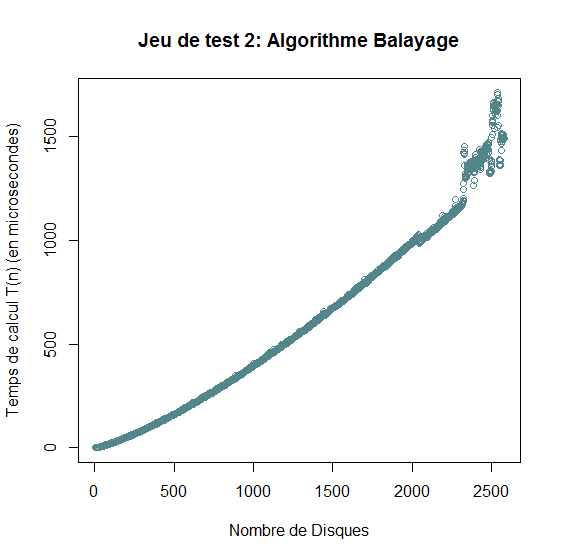
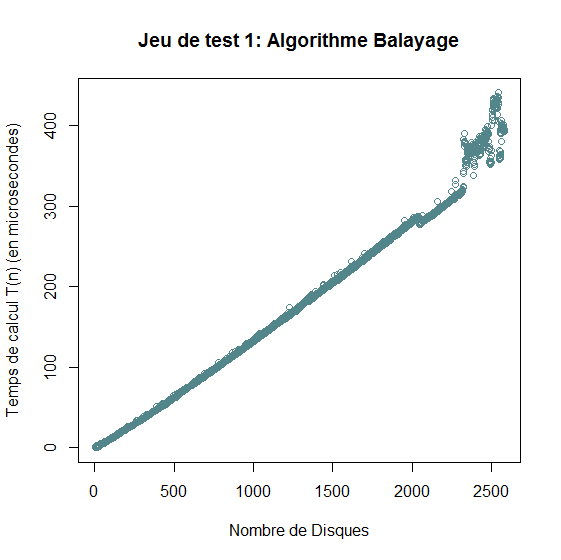
Pour estimer le temps de calcul T(n) pour chaque algorithme sur chacun des jeux de test, traçons les résultats obtenus sur ces derniers.

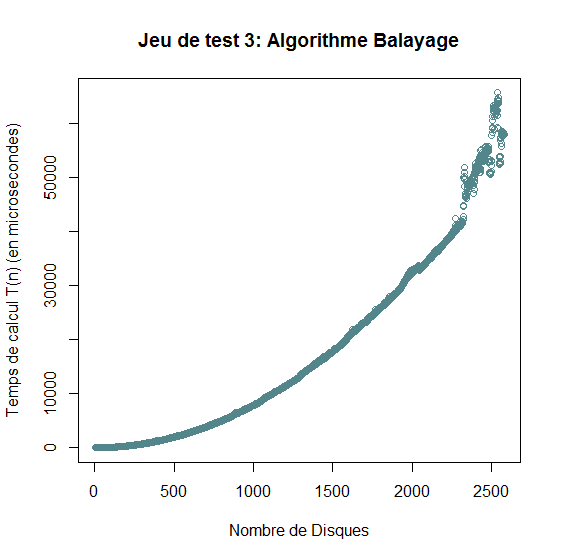
1. Algorithme Toutes Les Paires





1. Algorithme Balayage



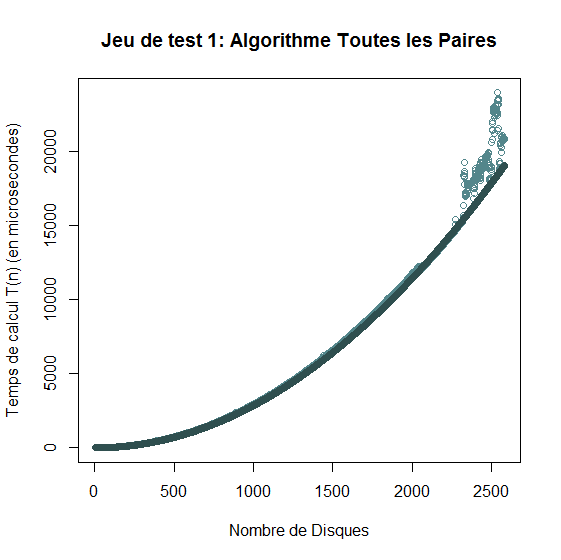


A présent, trouvons comme nous l’avons fait pour la question précédente, une fonction de type c1F(n)+c2 qui se rapproche des courbes obtenues pour chaque algorithme et chaque jeu de test. La courbe grise est notre fonction et la courbe bleue est le résultat de nos tests.

### Algorithme Toutes les Paires

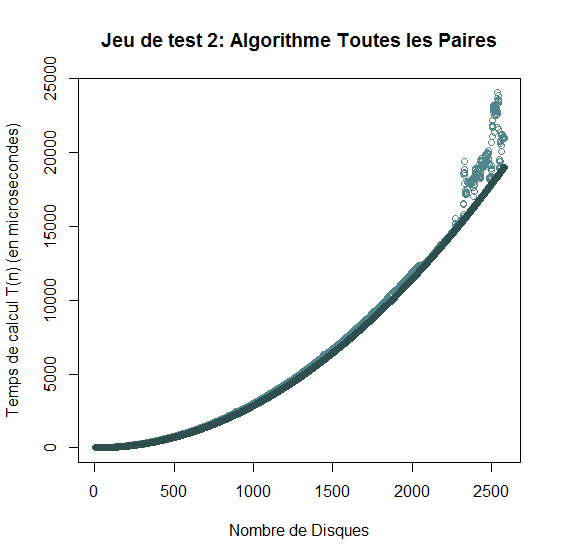
1. Jeu de test 1 : Rayon=1

* F(n)=n²
* C1=1/350
* C2=0



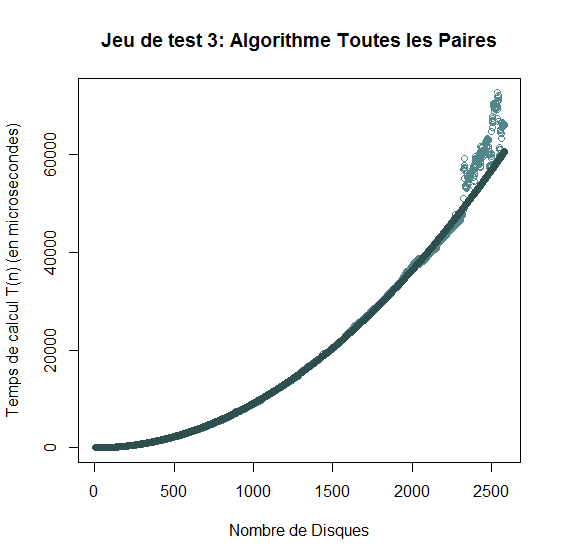
1. Jeu de test 2 : Rayon [0,].

* F(n)= n²
* C1=1/350
* C2=0



1. Jeu de test 3 : Rayon [0,n]

* F(n)=x²
* C1=1/110
* C2=0



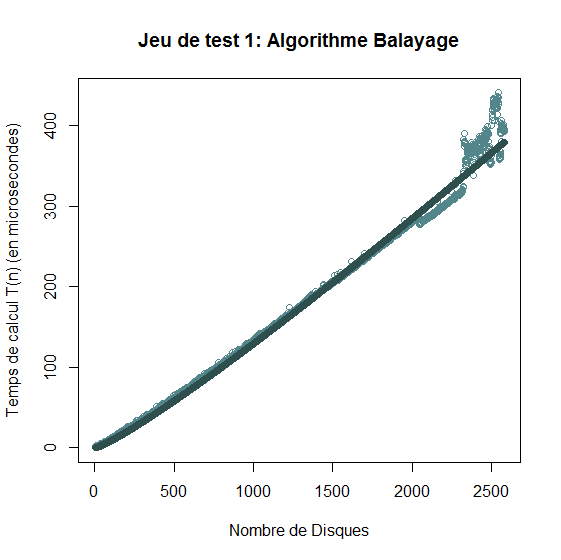
Pour l’algorithme Toutes Les Paires, nous pouvons donc déduire les complexités suivantes pour T(n) :

1. Dans le premier jeu de test, T(n)= O(n²)
2. Dans le deuxième jeu de test, T(n)=O(n²)
3. Dans le troisième jeu de test, T(n)=O(n²)

### Algorithme Balayage

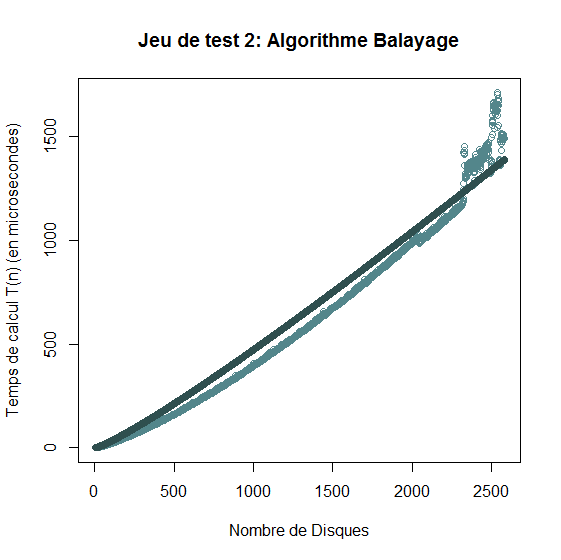
1. Jeu de test 1 : Rayon=1

* F(n)=n log n
* C1=1/77
* C2=0



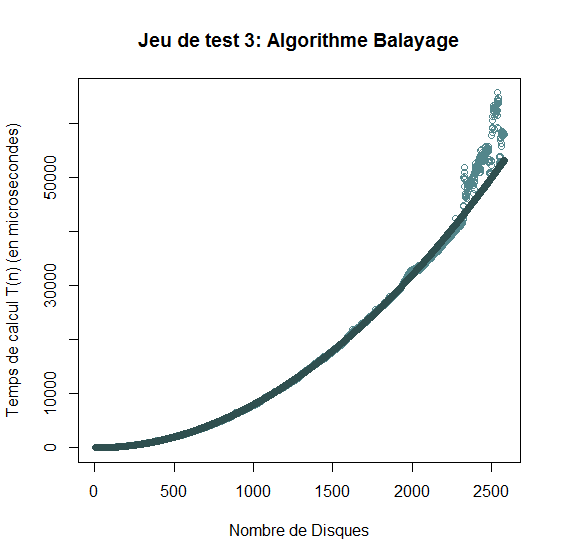
1. Jeu de test 2 : Rayon [0,].

* F(n)=n log n
* C1=4.75/100
* C2=0



1. Jeu de test 3 : Rayon [0,n]

* F(n)=n²
* C1=1/125
* C2=0



Pour l’algorithme Balayage, nous pouvons donc déduire les complexités suivantes pour T(n) :

1. Dans le premier jeu de test, T(n)= O(n log n)
2. Dans le deuxième jeu de test, T(n)=O(n log n)
3. Dans le troisième jeu de test, T(n)=O(n²)

Mise en relation avec les résultats de la question 2.c et 4 :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithme** | **Type de mesure** | **Jeu de test 1** | **Jeu de test 2** | **Jeu de test 3** |
| ***Toutes Les Paires*** | ***H(n)*** | O(log n) | O(n) | O(n²) |
| ***T(n)*** | O(n²) | O(n²) | O(n²) |
| ***Balayage*** | ***H(n)*** | O(log n) | O(n) | O(n²) |
| ***T(n)*** | O(n log n) | O(n log n) | O(n²) |

Complexité identiques – Meilleur – Moins bon

Le calcul de H(n) (en question 4) nous montre que nous avons des résultats identiques pour les deux algorithmes concernant le nombre de croisements obtenus. Nous nous servons donc de T(n) pour comparer l’efficacité de chacun.

Dans la question 2.c, nous avons estimé la complexité T(n) de l’algorithme *Balayage* à n log n+ k + (n-1). Ici, k est, pour chaque algorithme et chaque jeu de test, le résultat de H(n). Lorsque k est négligeable par rapport à n log n, la complexité sera donc de n log n. On observe cela avec les jeux de test un et deux( log n< n log n et n< n log n). Au contraire, dans le jeu de test 3, k=n². n² étant plus grand que n log n, la complexité de T(n) devient donc O(n²).

Dans tous les cas, la complexité de T(n) pour *Balayage* est inférieure ou égale à celle de *Toutes les paires.*

Nous pouvons donc affirmer que l’algorithme *Balayage* est plus efficace que *Toutes Les Paires*.

# Question 6