PROJET COMPLEXITE ALGORITHMIQUE

L3 MIAGE

BESSON Léonard

BURTEAUX Pierre

CHABOISSIER Maxime

PAPELIER Romain

SCHWEITZER Victorien

SOMMAIRE

UTILISATION DU PROGRAMME

Le programme est sous forme d'une simple archive JAR, lançable dans un terminal. Aucun argument n'est à fournir, le programme est entièrement interactif.

Le premier menu vous propose parmi 4 actions possibles :

- 1. Comparer le temps d'exécution pour les algos ToutesLesPaires et Balayage, pour un nombre de rectangles n et un numéro de jeu de test donnés.
- 2. Obtenir le nombre de paires de rectangles qui se coupent pour un algorithme, un nombre de rectangles n et un numéro de jeu de test donnés.
- 3. Tracer le graphique du temps d'exécution en fonction de n, pour un algo, un numéro de jeu de test et un n donnés.
- 4. Tracer le graphique du nombre de paires de rectangles qui se coupent en fonction de n, pour un algo, un numéro de jeu de test et un n donnés.

/ ! \ Lorsque n est très grand, les actions 3 et 4 peuvent s'avérer très longues. Dans ce cas il est préférable de recourir aux actions 1 et 2, et reporter les résultats dans un tableur.

DESCRIPTION DES OBJETS JAVA

Nous avons dans un premier temps créé la classe Rectangle, qui possède les coordonnées de son coin inférieur gauche et de son coin supérieur droit en attributs.

Nous avons ensuite implanté les algorithmes ToutesLesPaires et Balayage en tant que méthodes statiques dans la classe Main, qui contient la méthode statique « main ». D'autres méthodes sont venues se greffer pour offrir plus de choix d'utilisation à notre programme, ainsi qu'un module de rendu de graphique à partir d'un tableau de valeurs. Ce module, afin de rester un minimum performant, enregistre des valeurs par pas de n/100 à partir de n > 100.

1. IMPLANTATION DES ALGORITHMES

ToutesLesPaires(X, n)

K=0	Θ(1)
Pour i=0 à n-1 faire	Θ(n)
Pour j=i+1 à n-1 faire	Θ(n-i-1), O(n)
interX=(intersection en x…)	Θ(1)
interY=(intersection en y…)	Θ(1)
si interX et interY alors	Θ(1)
k=k+1	O(1)
Retourner k	Θ(1)

Pour i et j fixés, on a O(4)
Pour i fixés, on a j itéré n-i-1 fois donc Θ(n-i-1) = O(n)
I est itéré n fois donc on a n.O(n) = O(n²)
A la fin on retourne k, on a donc Θ(1) + O(n²) = O(n²), ou Θ(n.(n-1) / 2) (formule d'une somme des i allant de 1 à n)

Balayage(X, n)

TriRapide(X,0,n)	O(n.log n)
K=0	Θ(1)
Pour i=0 à n-1 faire	Θ(n)
J=i+1	Θ(1)
Tant que j < n et X[j].x1 < X[i].x2 faire	O(n-j)
interY=(intersection en y…)	Θ(1)
si interY alors	Θ(1)
k=k+1	O(1)
J=j+1	Θ(1)
Retourner k	Θ(1)

Pour i et j fixés, on a O(3)
Pour i fixé, on a n-j itérations dans le pire des cas, ce qui donne O(n-j), or j=i+1.
On peut donc ramener ça à O(n-i-1) = O(n-i)
La complexité totale est de O(n.log n) + $\sum_{i=1}^{n-1} O(n-i)$ = O(n + $\frac{n(n-1)}{2}$) = O(n²) + O(n.log n)

Dans le pire des cas, l'algorithme Balayage a une complexité en nombre d'opération identique à l'algorithme ToutesLesPaires si on écarte la partie TriRapide, étant donné que le pire des cas pour le Balayage est le cas où tous les rectangles se coupent en X, ce qui veut dire que la boucle tant que se répétera systématiquement pour j allant de i+1 à n-1, ce qui revient à un algorithme de ToutesLesPaires.

2. COMPLEXITE DE BALAYAGE EN FONCTION DE N ET k_x

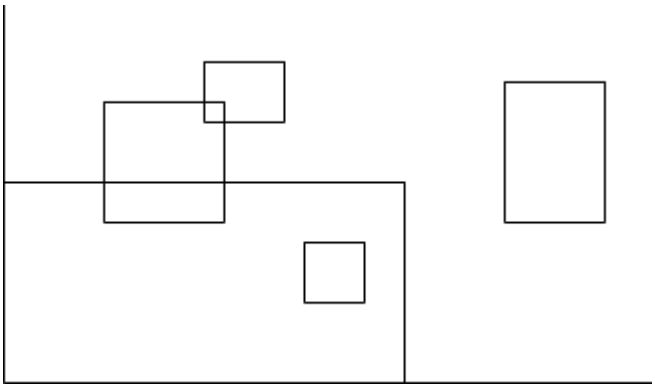
Balayage(X, n)

TriRapide(X,0,n)	O(n.log n)
K=0	Θ(1)
Pour i=0 à n-1 faire	Θ(n)
J=i+1	Θ(1)
Tant que j < n et X[j].x1 < X[i].x2 faire	Θ($k_{x\,i}$)
interY=(intersection en y…)	Θ(1)
si interY alors	Θ(1)
k=k+1	O(1)
J=j+1	Θ(1)
Retourner k	Θ(1)

On a k_x le nombre total de paires de rectangles qui se coupent en x. L'algorithme de balayage consiste à, pour chaque rectangle à l'indice i dans X, parcourir les $k_{x\,i}$ autres rectangles qui le coupent en x.
Pour i fixé, on a une complexité sur la boucle tant que de Θ($k_{x\,i}$)
Pour i allant de 1 à n, on a une complexité égale à $\sum_{i=1}^{n}$ Θ($k_{x\,i}$) = Θ(k_x)
Dans tous les cas, i est itéré de 0 jusque n. Ensuite la somme totale des itérations de j pour chaque i est égale à k_x . On a donc une complexité totale de Θ(n + k_x) + O(n.log n)

Considérons k_x comme étant « petit » lorsque k_x < n² par exemple. Dans ce cas, la complexité totale est inférieure à Θ(n + n²) + O(n.log n).
Or n + n² = O(n²).
La complexité totale du Balayage en fonction de n et k_x , pour k_x « petit », est donc inférieure à la complexité O(n²) + O(n.log n) trouvée en 1.

3. TEST DES ALGORITHMES

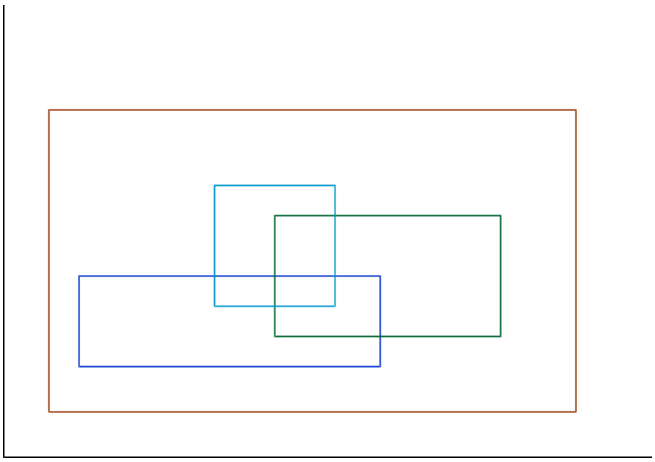


5 rectangles :

- (0,0,200,100)
- (150,40,180,70)
- (50,80,110,140)
- (100,130,140,160)
- (250,80,300,150)

Résultat du test :

- ToutesLesPaires : 3
- Balayage : 3

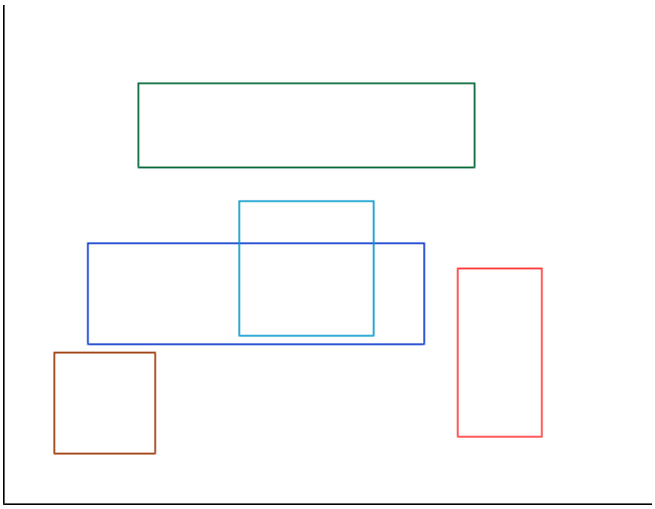


4 rectangles :

- (30,30,380,230)
- (50,60,250,120)
- (180,80,330,160)
- (140,100,220,180)

Résultat du test :

- ToutesLesPaires : 6
- Balayage : 6



5 rectangles :

- (30,30,90,90)
- (50,95,250,155)
- (140,100,220,180)
- (80,200,280,250)
- (270,40,320,140)

Résultat du test :

- ToutesLesPaires : 1
- Balayage : 1

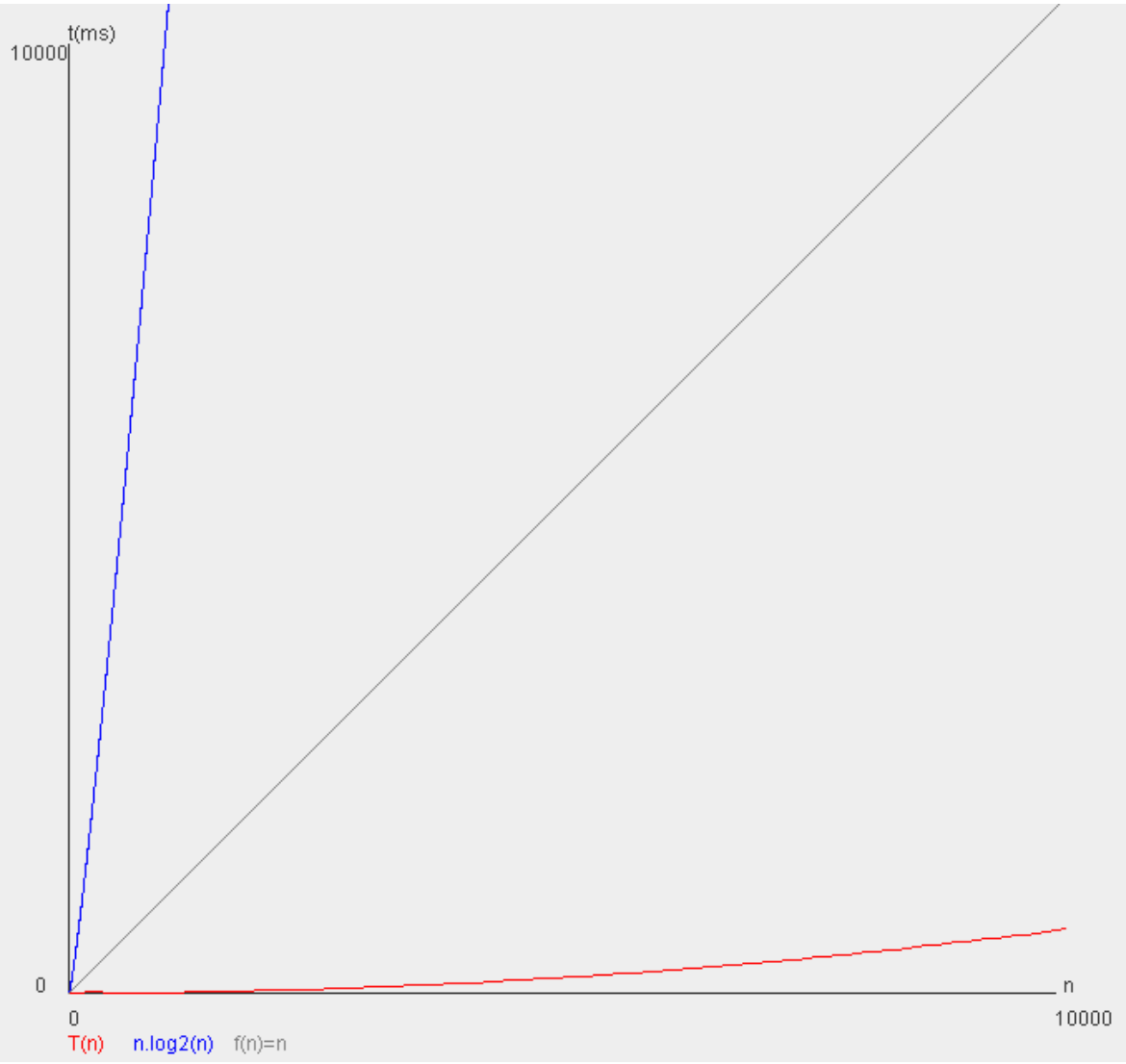
4. ESTIMATION EMPIRIQUE DU TEMPS DE CALCUL

Nous avons passé les algorithmes ToutesLesPaires et Balayage sur les 3 types de jeu de test pour un ensemble de n=10000 rectangles.
Tous les tests qui suivent ont été réalisé sur un ordinateur équipé d'un processeur Intel Core i7.

JEU DE TEST 1 : COORDONNEES TIREES DANS [0,N]

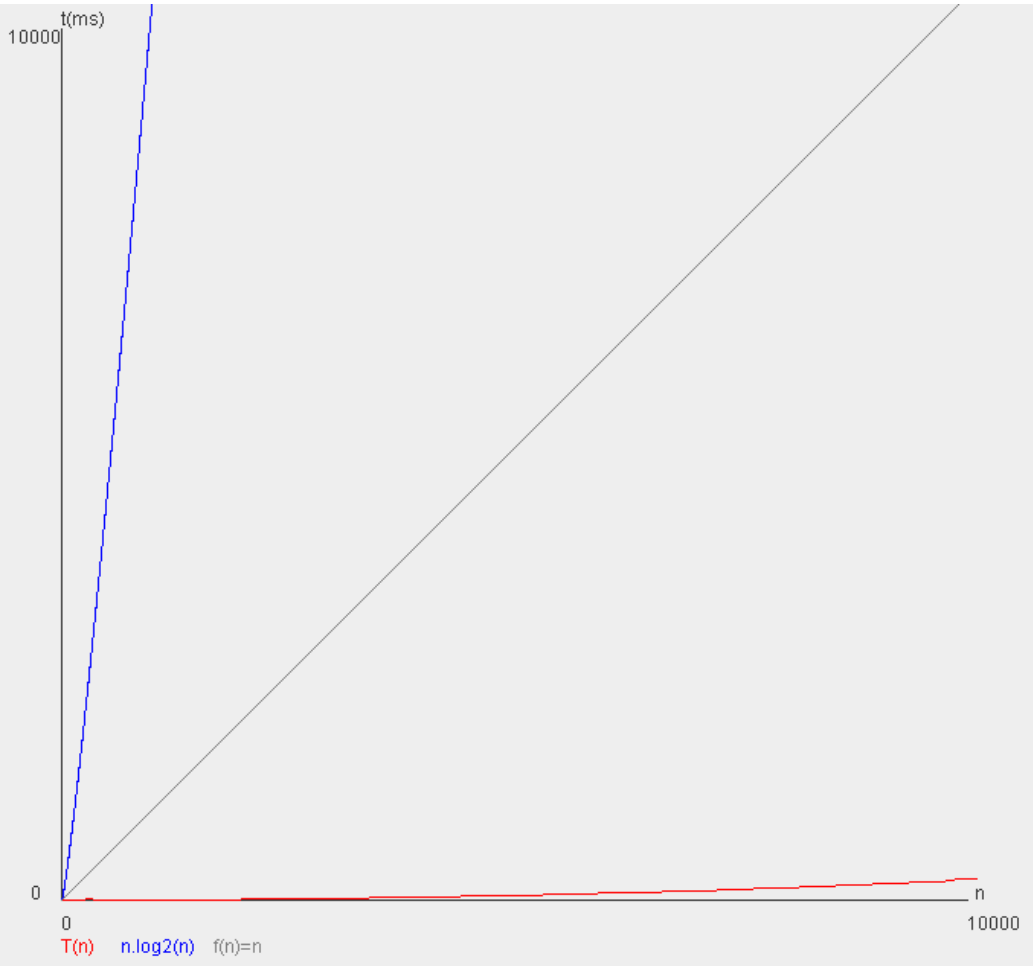
TOUTES LES PAIRES

Voici le graphique du temps d'execution de l'algorithme ToutesLesPaires en millisecondes en fonction de n, sur un ensemble de n rectangles aux coordonnées x1,y1,x2,y2 tirées aléatoirement sur [0,n]. La courbe rouge représente le temps d'execution en fonction de n, celle en bleu la fonction $n.log_2(n)$ et celle en gris la fonction f(n) = n.



BALAYAGE

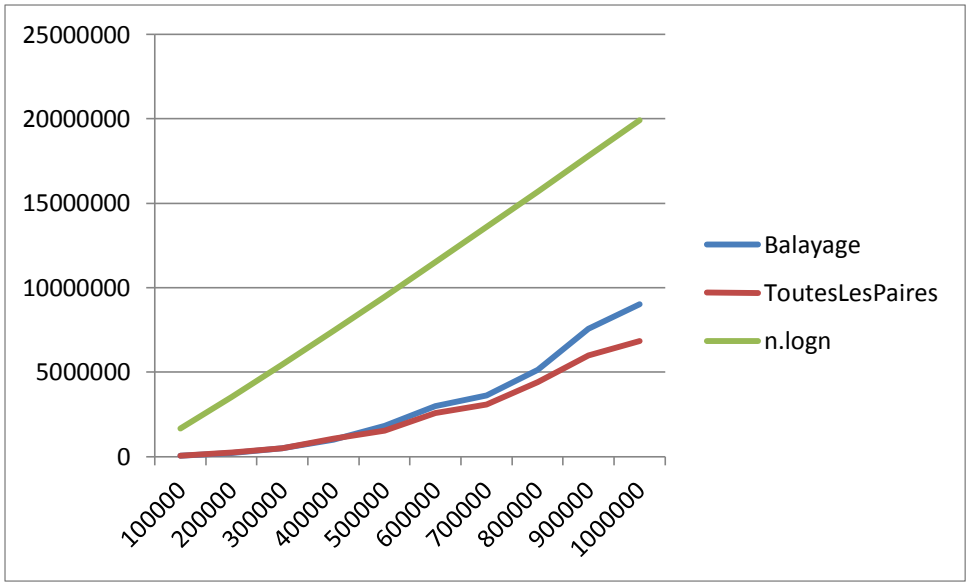
Voici le graphique du temps d'execution de l'algorithme Balayage en millisecondes en fonction de n, sur le même type de jeu de test. On remarque clairement qu'ici l'algorithme de Balayage est plus efficace en terme de temps d'execution que l'algorithme ToutesLesPaires.



COMPARAISON

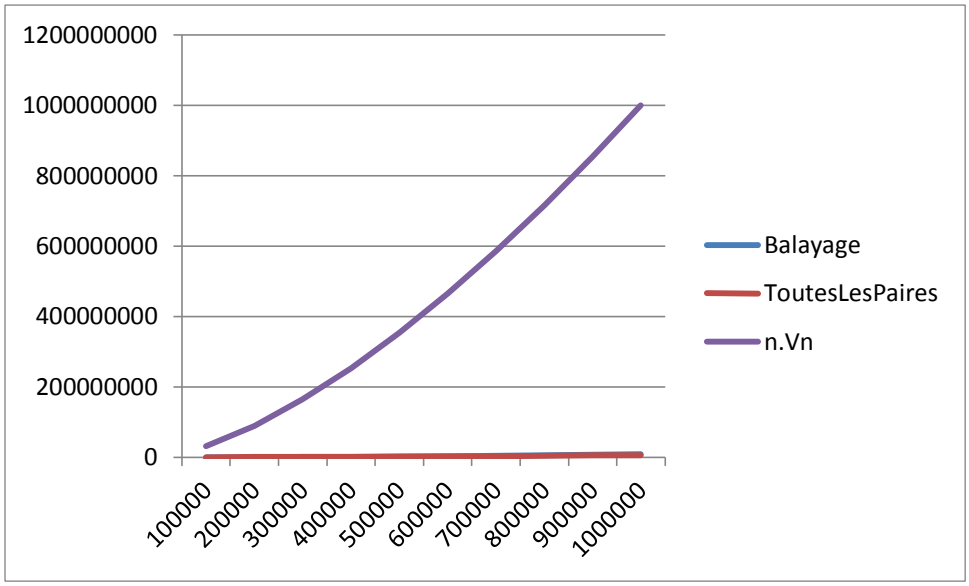
On remarque très bien que pour ce jeu de test, et avec n=10000, l'algorithme de Balayage est plus rapide que l'algorithme ToutesLesPaires. D'après ce graphique, la complexité de nos deux algorithmes semble se rapprocher d'un O(n.log n). Pour nous en assurer, nous avons testé séparément des jeux de 100 000,200 000, 300 000, … , 1 000 000 rectangles, que nous avons enregistré dans un tableau à partir duquel nous avons tracé un graphique Excel, afin de gagner du temps, car lancer directement notre fonction de traçage de courbes pour de telles valeurs aurait été beaucoup trop long.

On obtient le graphique suivant :



Dans un premier temps, il s'avère que l'algorithme de Balayage soit plus lent que l'algorithme ToutesLesPaires à partir de 500 000 rectangles. Cela est dû au fait que l'algo de Balayage doit d'abord trier la liste des rectangles. Pour n < 500000, cela permet un gain de temps, mais au-delà, l'algo ToutesLesPaires est plus performant.

Dans un second temps, on remarque que d'après la tendance des deux courbes la complexité de Balayage et ToutesLesPaires est un Ω(n.log n) à partir d'un certain n (plus grand qu'un million).

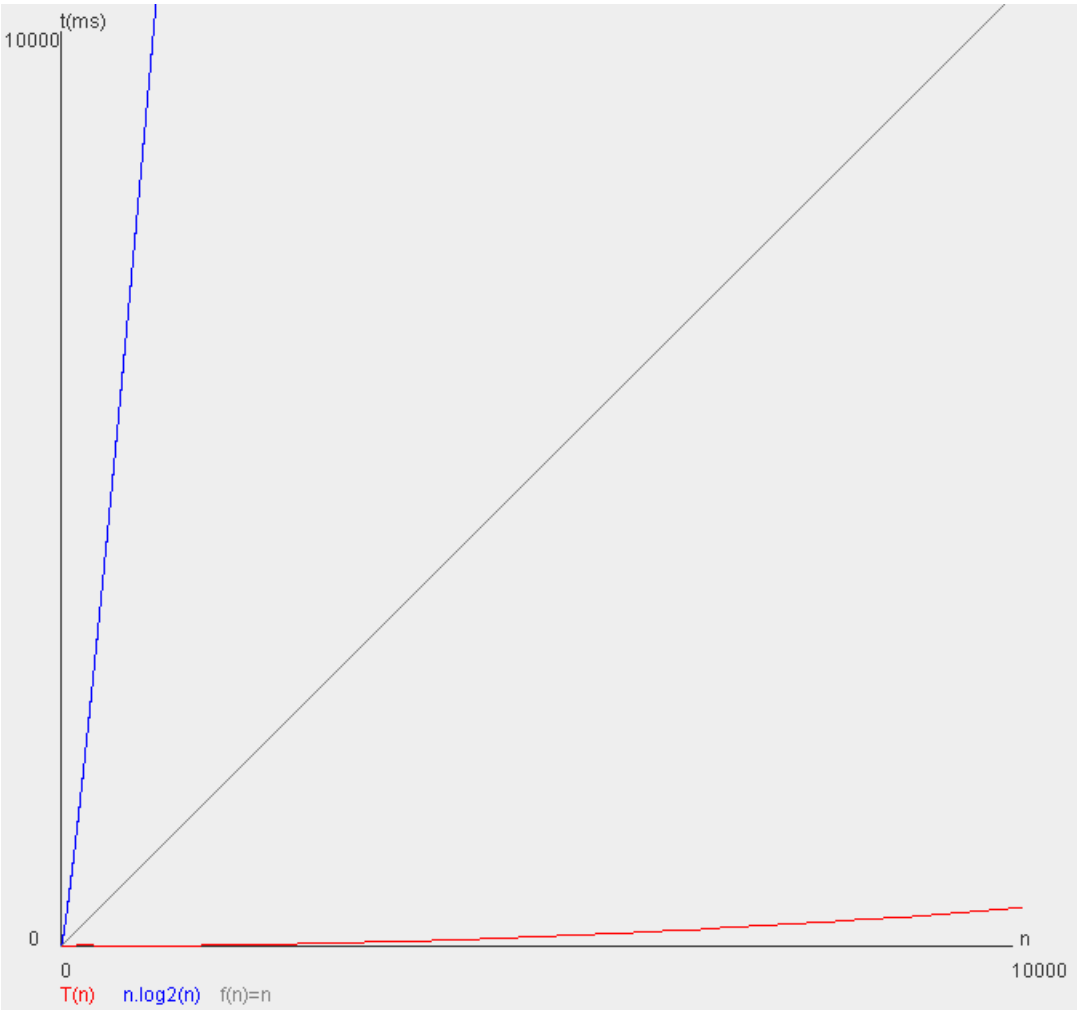


Ici, nous avons comparé la complexité de Balayage et ToutesLesPaires à la fonction n.√n. Cette fonction croit beaucoup plus vite que n.log n et que Balayage et ToutesLesPaires. Il semble donc assez clair que la complexité en terme de temps d'exécution de ces algorithmes est un O(n.√n).

JEU DE TEST 2 : X1 Y1 TIRES SUR [0,N], DE LONGUEUR 1

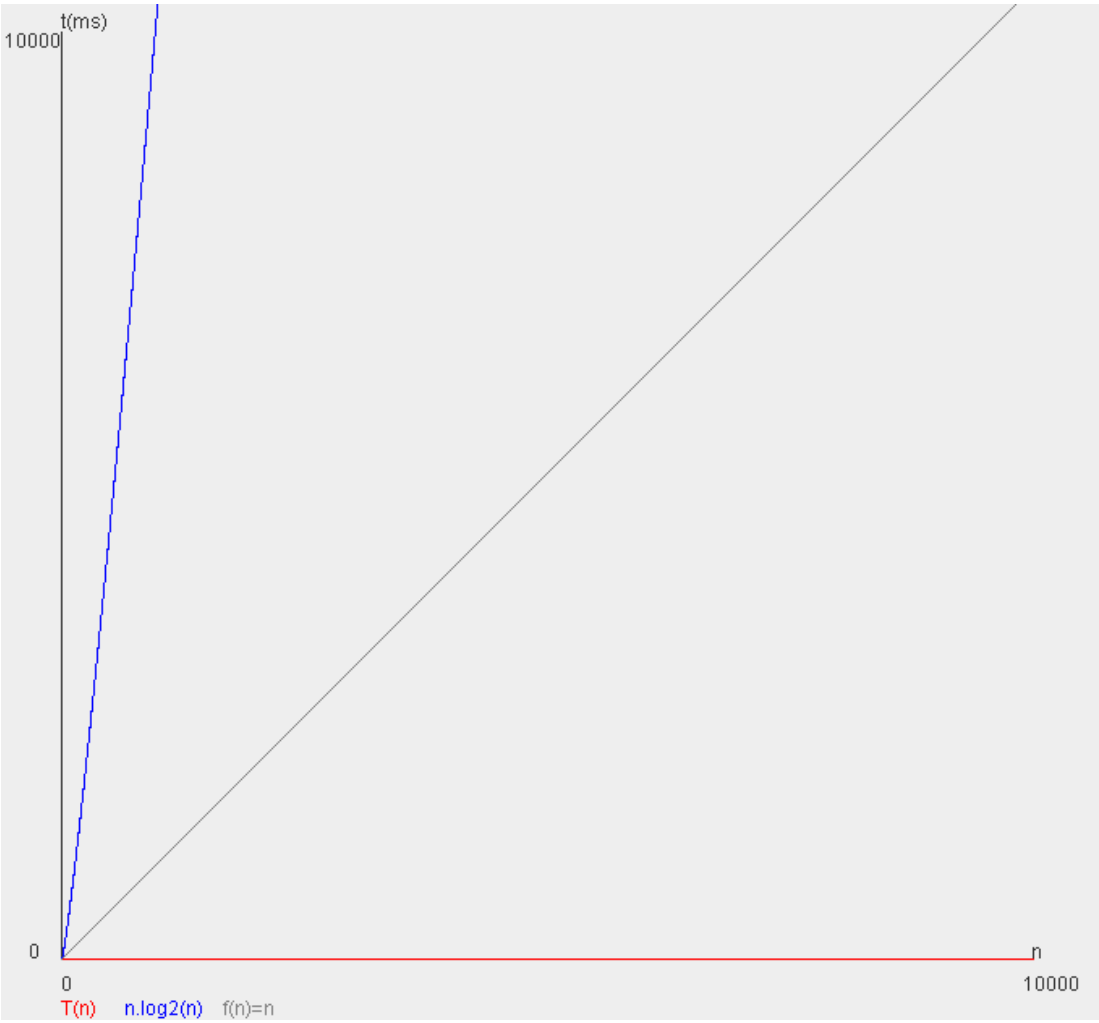
TOUTES LES PAIRES

Voici le graphique du temps d'execution de l'algorithme ToutesLesPaires en millisecondes en fonction de n, sur un ensemble de n rectangles de largeur et hauteur égales à 1 et aux coordonnées x1,y1 tirées aléatoirement sur [0,n]. On peut voir que le temps d'execution est plus rapide que sur le test 1, ceci est du au fait que sur un intervalle [0,10000] avec une longeur et largeur de 1, il y a très peu de chance pour avoir des rectangles qui se coupent, on rentre donc rarement dans le bloc « if » qui incrémente k, ce qui évite une opération.



BALAYAGE

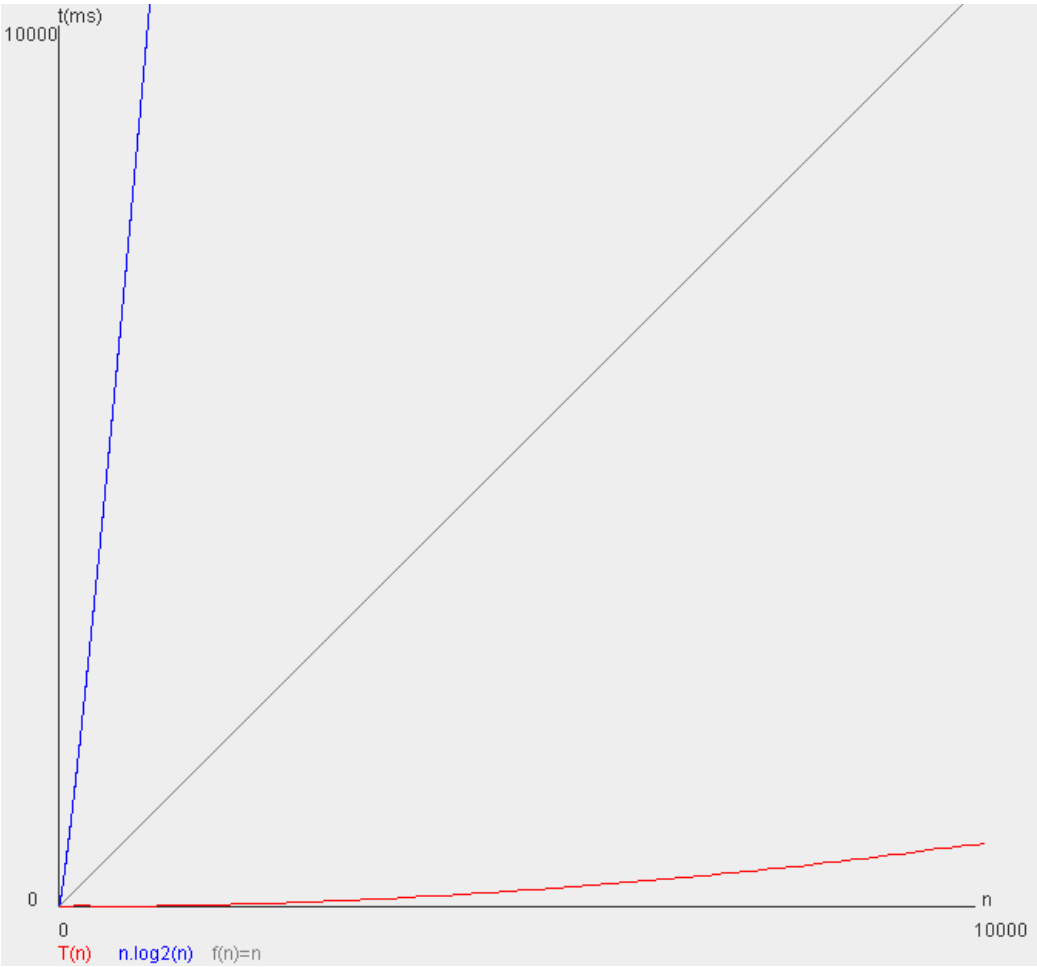
Voici le même graphique pour l'algorithme de Balayage. Etant donné qu'on a très peu de chance d'avoir des couples de rectangles qui se coupent, l'algorithme de balayage se contente d'itérer la première boucle n fois, sans jamais (ou presque) entrer dans la seconde boucle. Le temps d'execution est donc inférieur ou égal au temps qu'il faut pour effectuer entre 0 et 10000 itérations simples, ce qui est très rapide sur un ordinateur actuel.



JEU DE TEST 3 : COORDONNEES TIREES SUR [0,√N]

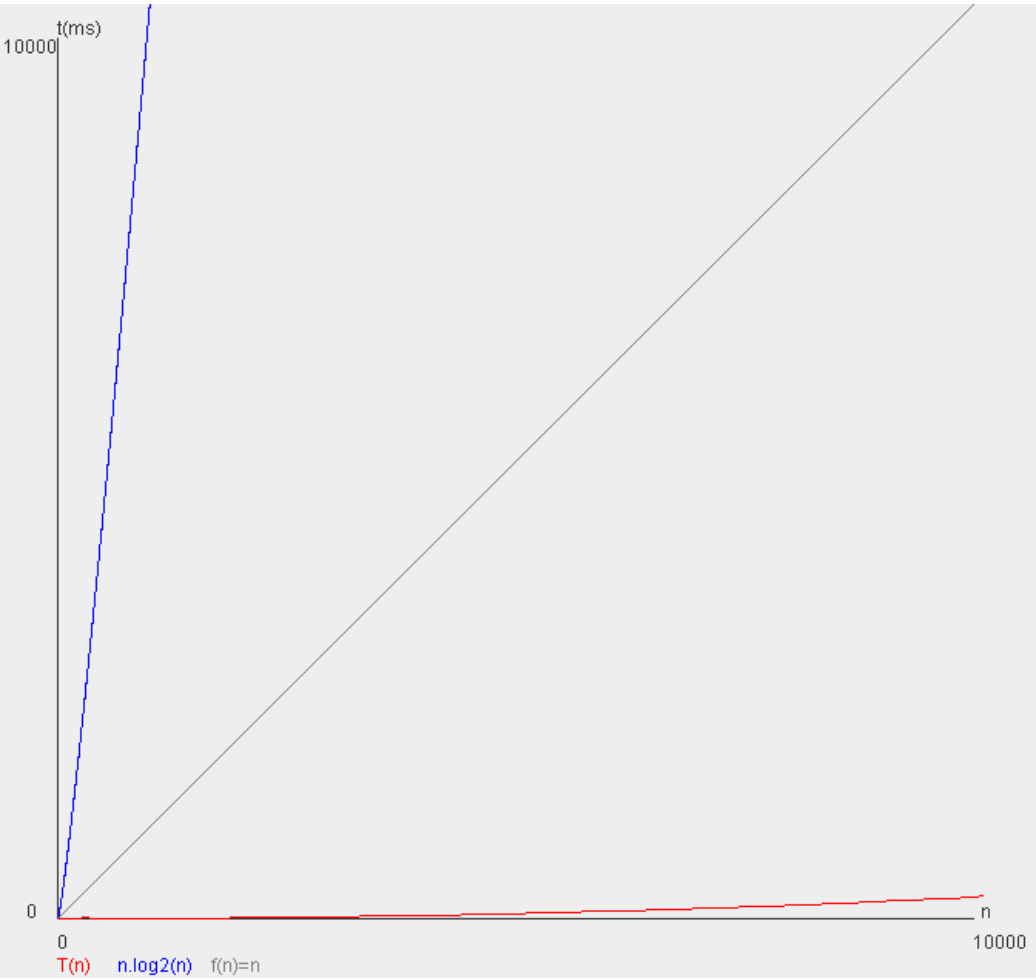
TOUTES LES PAIRES

Voici le graphique du temps d'execution de l'algorithme ToutesLesPaires en millisecondes en fonction de n, sur un ensemble de n rectangles aux coordonnées x1,y1,x2,y2 tirées aléatoirement sur [0, √n]. Le temps d'execution ici est sensiblement identique à celui obtenu au jeu de test 1.



BALAYAGE

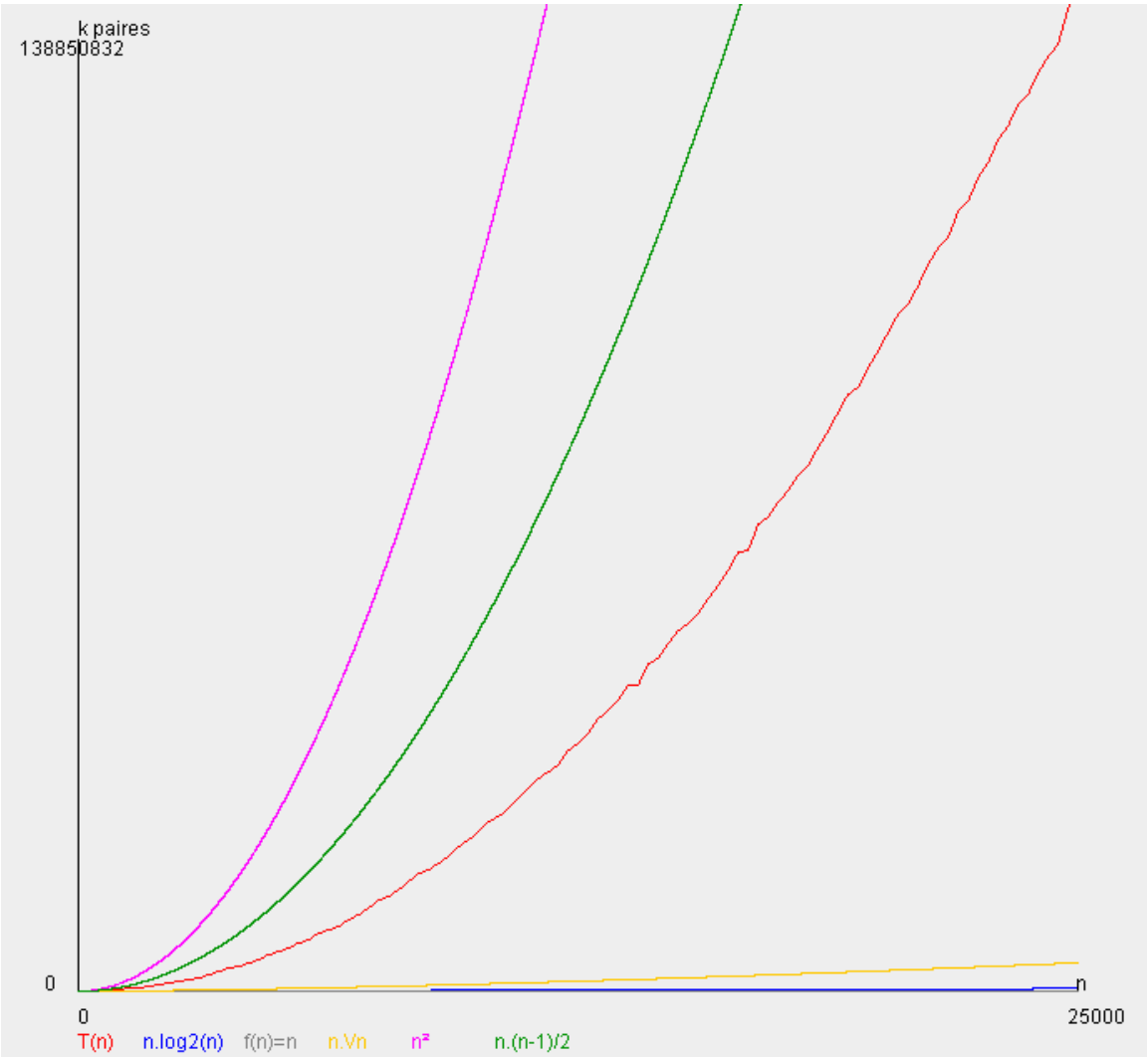
Ici on a le même graphique pour la fonction Balayage, et comme avec le jeu de test 1, l'algorithme de Balayage est plus performant que l'algorithme ToutesLesPaires pour n « petit ».



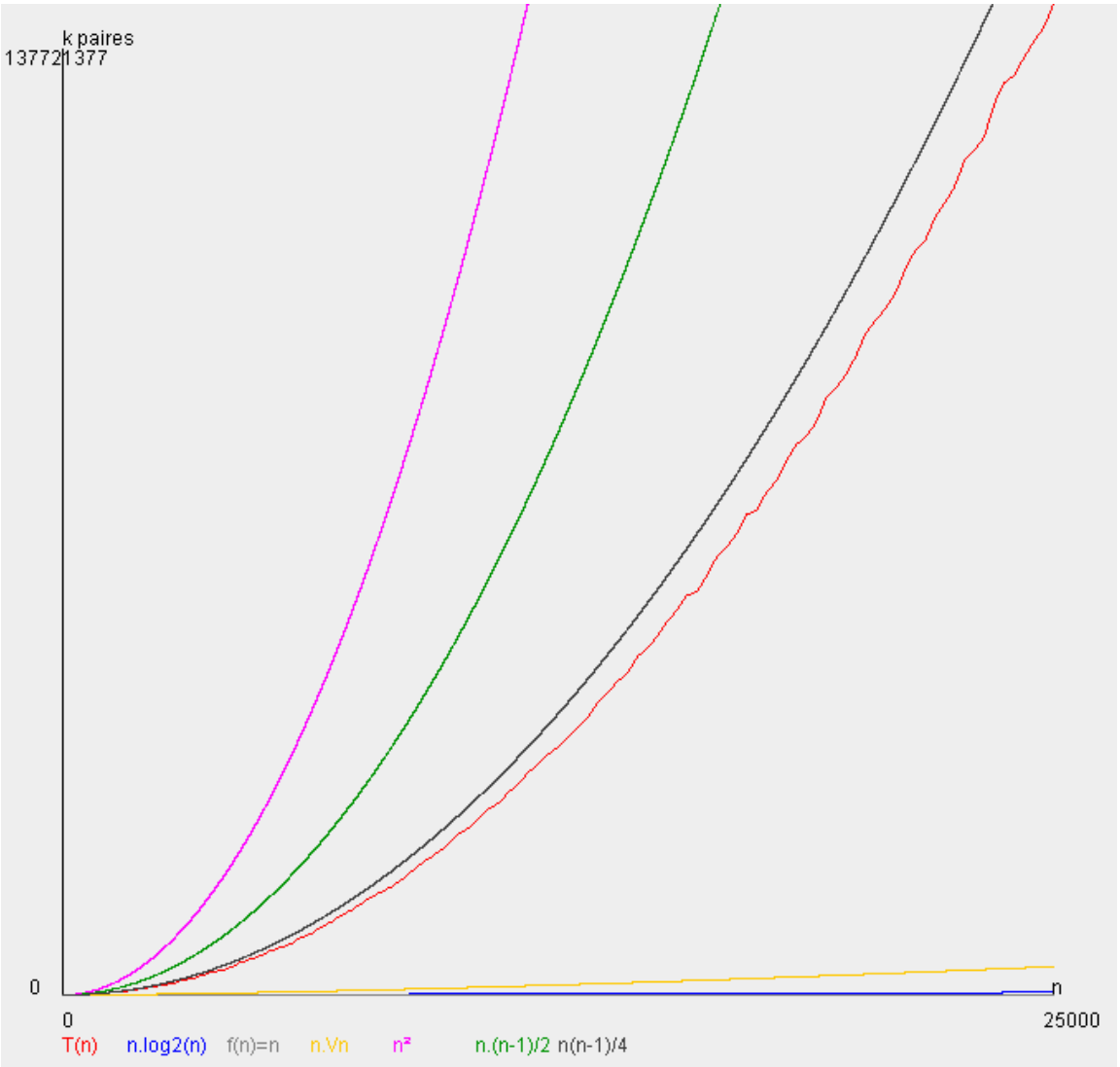
5. ESTIMATION EMPIRIQUE DU NOMBRE DE PAIRES DE RECTANGLES QUI SE COUPENT

JEU DE TEST 1 : COORDONNEES TIREES DANS [0,N]

En testant nos algorithmes sur des ensembles de 0 à 25000 rectangles, on obtient le graphique suivant :



Nous y avons représenté k en fonction de n ainsi que la fonction n^2 et n.(n-1)/2 . En effet, le nombre maximal de paires de rectangles **différentes** est égal à $\sum_{i=1}^{n-1} n$, soit n(n-1)/2. Le nombre de paires de rectangles différentes qui se coupent (k) est un O(n(n-1)/2).

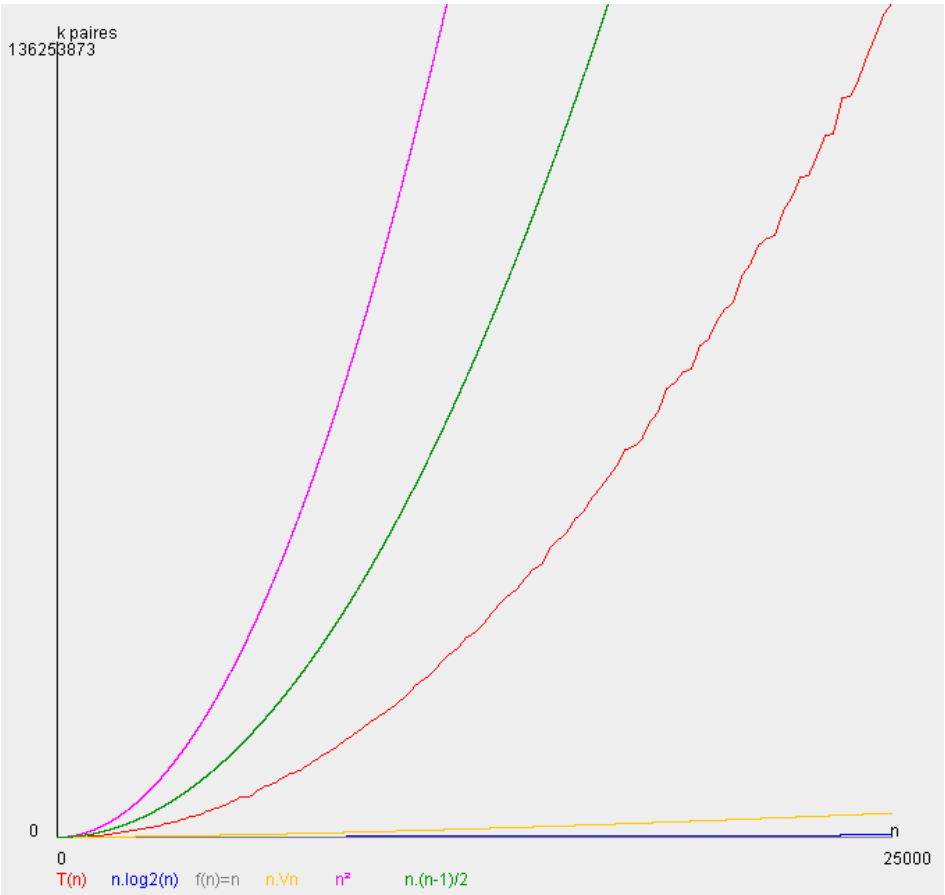


Nous avons essayé un rendu graphique en ajoutant cette fois la fonction **n(n-1)/4** (courbe noire), qui n'est autre que la moitié de **n(n-1)/2** pour chaque n. Le nombre k de paires de rectangles qui se coupent est donc en moyenne légèrement inférieur à la moitié du nombre maximal de paires possibles.

JEU DE TEST 2 : X1 Y1 TIRES SUR [0,N], DE LONGUEUR 1

Ici le nombre total de paires de rectangles qui se coupent est égal à 0. La courbe de k est donc une constante égale à 0.

JEU DE TEST 3 : COORDONNEES TIREES SUR [0,√N]



On peut remarquer que le nombre de paires k en fonction de n sur le jeu 3 est quasiment identique à celui du jeu 1.

CONCLUSION GENERALE

Voici un tableau récapitulatif des complexités étudiées tout au long du projet :

	ToutesLesPaires	Balayage
Complexité en nombre d'opérations en fonction de n	$O(n^2)$	$O(n.\log n) + O(n^2)$
Complexité en nombre d'opérations en fonction de n et k_x	$O(n^2)$	$O(n.\log n) + O(n+k_x)$
Complexité empirique en temps en fonction de n	$O(n.\sqrt{n})$	$O(n.\sqrt{n})$
Nombre k de paires de rectangles qui se coupent en fonction de n	$O(\frac{n(n-1)}{4})$	

Voici un tableau comparatif des deux algorithmes d'après les tests effectués avec notre programme :

	Comparaison en termes de temps d'exécution
Jeu 1 & 3 : n rectangles de taille [0,n]	ToutesLesPaires > Balayage pour n < 400 000
	ToutesLesPaires < Balayage pour n > 400 000
Jeu 2 : n rectangles de taille 1	ToutesLesPaires >> Balayage

L'efficacité d'un algorithme par rapport à un autre dépend donc beaucoup de l'ensemble de rectangle, de la quantité de rectangles présents et de leur disposition. S'il s'agit d'un ensemble où l'on sait à l'avance que très peu de rectangles ont une chance de se couper (jeu 2), alors l'algorithme de Balayage sera plus efficace, quel que soit n. Si il s'agit d'un jeu ou les rectangles ont de bonnes chances de se couper les uns les autres, alors Balayage est là encore plus efficace pour un certain nombre de rectangles (inférieur à 400000 par exemple), mais ToutesLesPaires prend l'avantage pour un nombre de rectangles plus grand.