PROJET COMPLEXITE ALGORITHMIQUE

L3 Miage

BESSON Léonard

BURTEAUX Pierre

CHABOISSIER Maxime

PAPELIER Romain

SCHWEITZER Victorien

SOMMAIRE

[1. Implantation des algorithmes 3](#_Toc416096678)

[2. Complexité de Balayage en fonction de n et 4](#_Toc416096679)

[3. Test des algorithmes 4](#_Toc416096680)

[4. Passage aux jeux de test 5](#_Toc416096681)

[Jeu de test 1 : coordonnées tirées dans [0,n] 5](#_Toc416096682)

[Toutes les paires 5](#_Toc416096683)

[Balayage 6](#_Toc416096684)

[Balayage 6](#_Toc416096685)

[Jeu de test 2 : x1 y1 tirés sur [0,n], de longueur 1 7](#_Toc416096686)

[Toutes les paires 7](#_Toc416096687)

[Balayage 8](#_Toc416096688)

[Jeu de test 3 : coordonnées tirées sur [0,√n] 9](#_Toc416096689)

[Toutes les paires 9](#_Toc416096690)

[Balayage 10](#_Toc416096691)

# 1. Implantation des algorithmes

ToutesLesPaires(X, n)

K=0 Ө(1)

Pour i=0 à n-1 faire Ө(n)

Pour j=i+1 à n-1 faire Ө(n-i-1), O(n), Ө(n.(n-1) / 2)

interX=(intersection en x…) Ө(1)

interY=(intersection en y…) Ө(1)

si interX et interY alors Ө(1)

k=k+1 O(1)

Retourner k Ө(1)

Pour i et j fixés, on a O(4)

Pour i fixés, on a j itéré n-i-1 fois donc Ө(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 Ө(1) + O(n²) = O(n²)

Balayage(X, n)

TriRapide(X,0,n) O(n.log n)

K=0 Ө(1)

Pour i=0 à n-1 faire Ө(n)

J=i+1 Ө(1)

Tant que j < n et X[j].x1 < X[i].x2 faire O(n-j)

interY=(intersection en y…) Ө(1)

si interY alors Ө(1)

k=k+1 O(1)

J=j+1 Ө(1)

Retourner k Ө(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) + = O( n + ) = 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. Complexité de Balayage en fonction de n et

Balayage(X, n)

TriRapide(X,0,n) O(n.log n)

K=0 Ө(1)

Pour i=0 à n-1 faire Ө(n)

J=i+1 Ө(1)

Tant que j < n et X[j].x1 < X[i].x2 faire Ө()

interY=(intersection en y…) Ө(1)

si interY alors Ө(1)

k=k+1 O(1)

J=j+1 Ө(1)

Retourner k Ө(1)

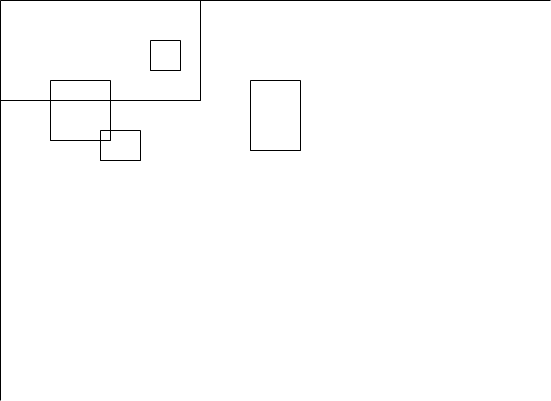
On a le nombre total de paires de rectangles qui se coupent en x. L’algorithme de balayage consiste à, pour chaque rectangle à l’indice i dans X, parcourir les autres rectangles qui le coupent en x.

Pour i fixé, on a une complexité sur la boucle tant que de Ө( )

Pour i allant de 1 à n, on a une complexité égale à = Ө()

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 . On a donc une complexité totale de Ө(n + ) + O(n.log n)

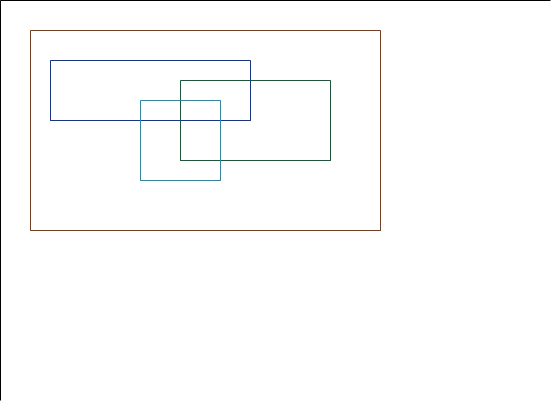
# 3. Test des algorithmes

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 :

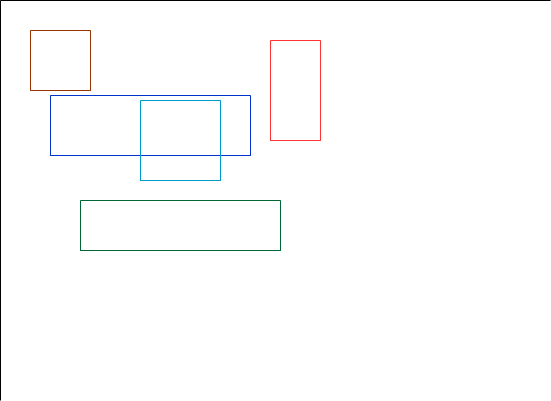
* ToutesLesPaires : 3
* Balayage : 3

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

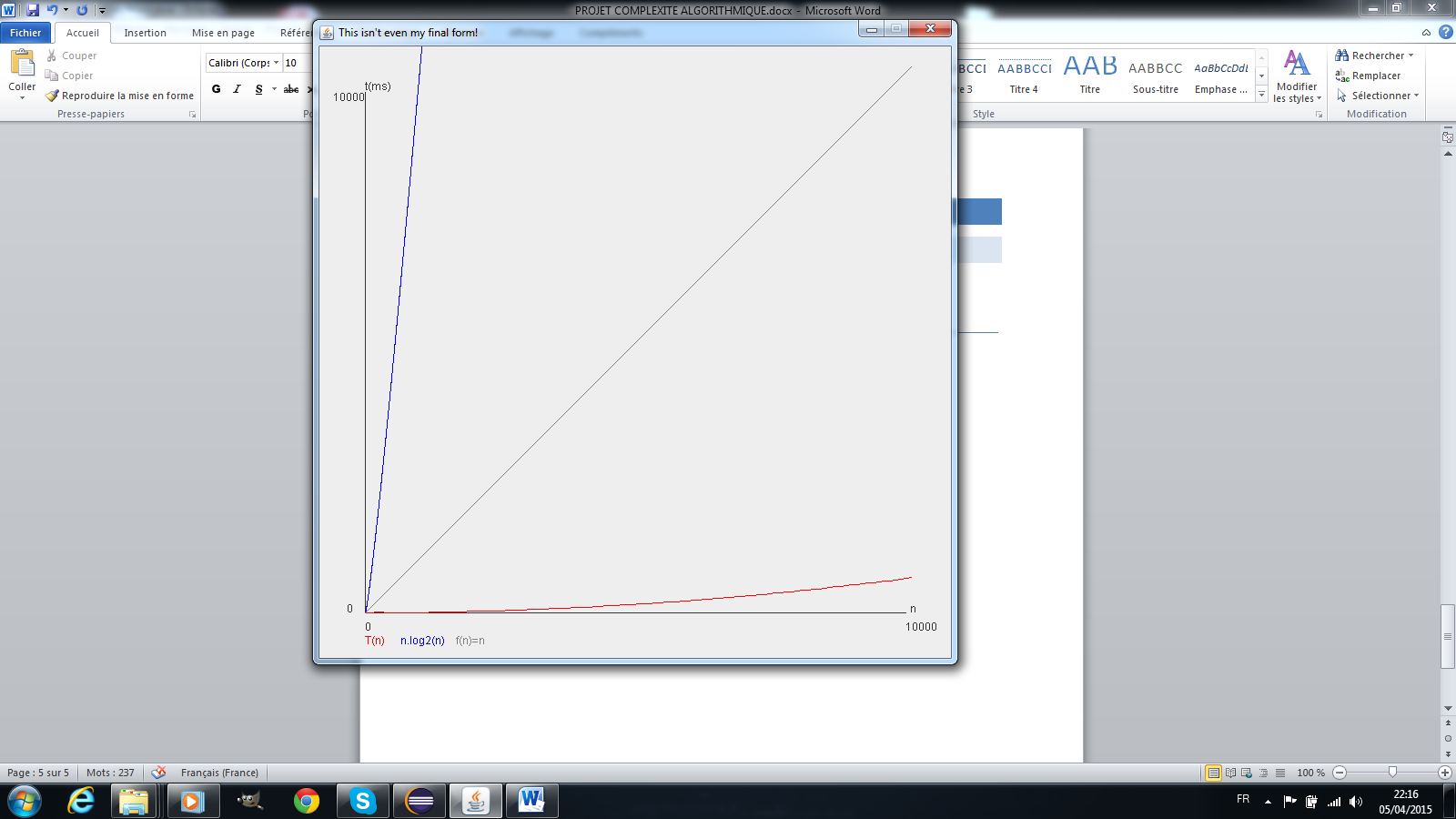
# 4. Passage aux jeux de test

Nous avons passé les algorithmes ToutesLesPaires et Balayage sur les 3 types de jeu de test pour un ensemble de n=10000 rectangles.

## Jeu de test 1 : coordonnées tirées dans [0,n]

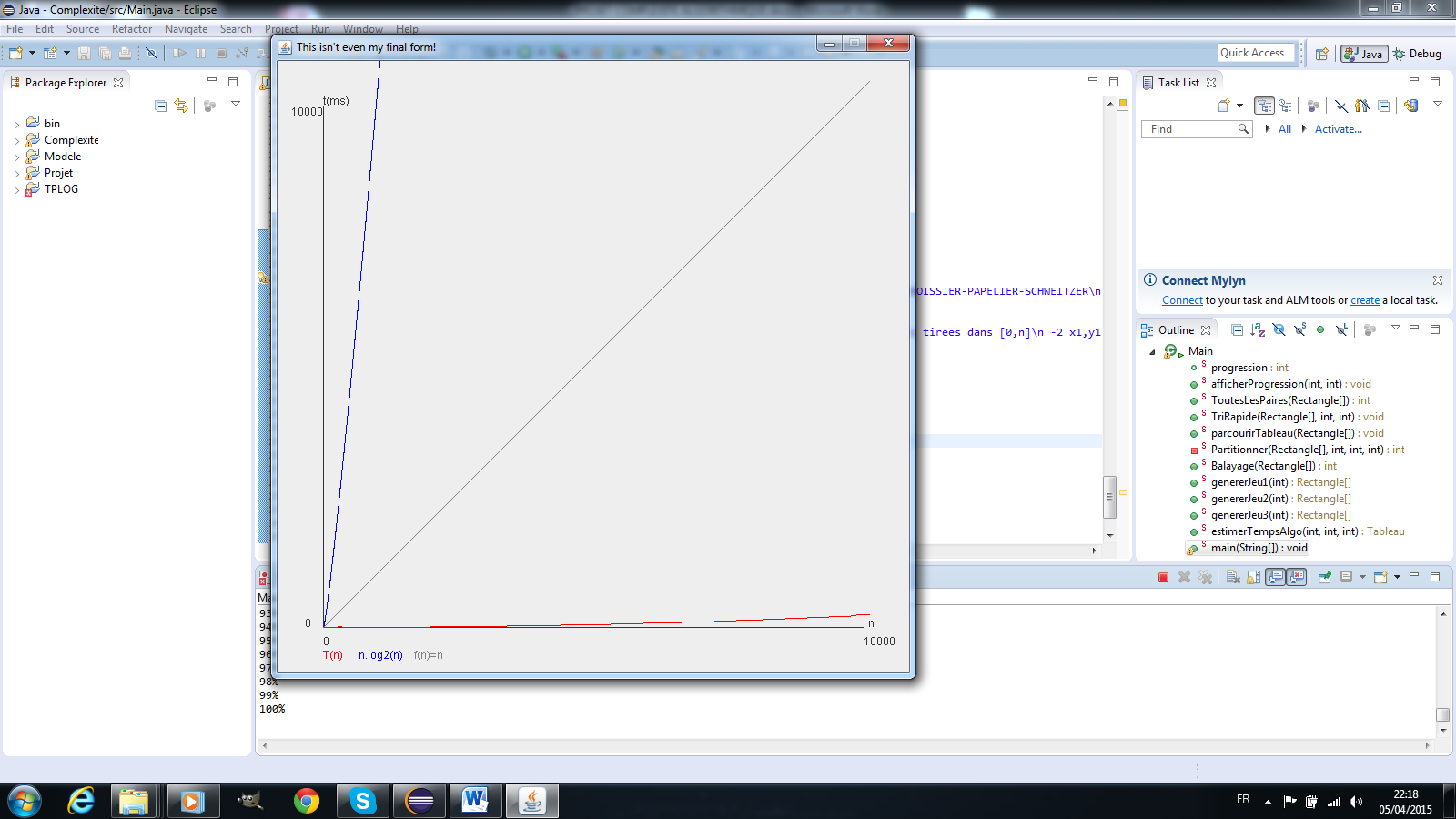
### Toutes les paires

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.(n) et celle en gris la fonction f(n) = n.



### Balayage

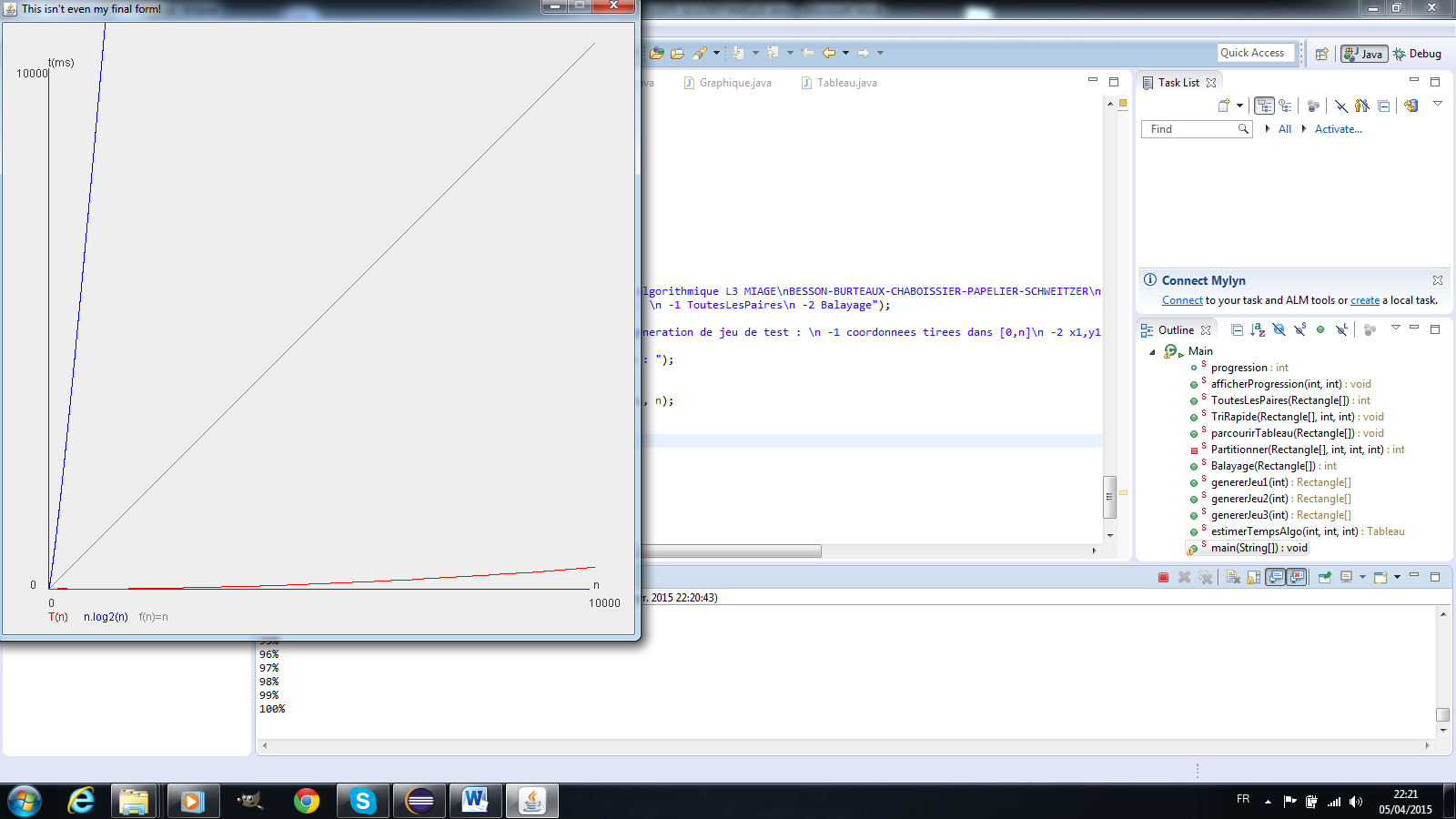
Voici le graphique du temps d’execution de l’algorithme Balayage en millisecondes en fonction de n, sur le même type de jeu de test. On remarque clairement qu’ici l’algorithme de Balayage est plus efficace en terme de temps d’execution que l’algorithme ToutesLesPaires.



## Jeu de test 2 : x1 y1 tirés sur [0,n], de longueur 1

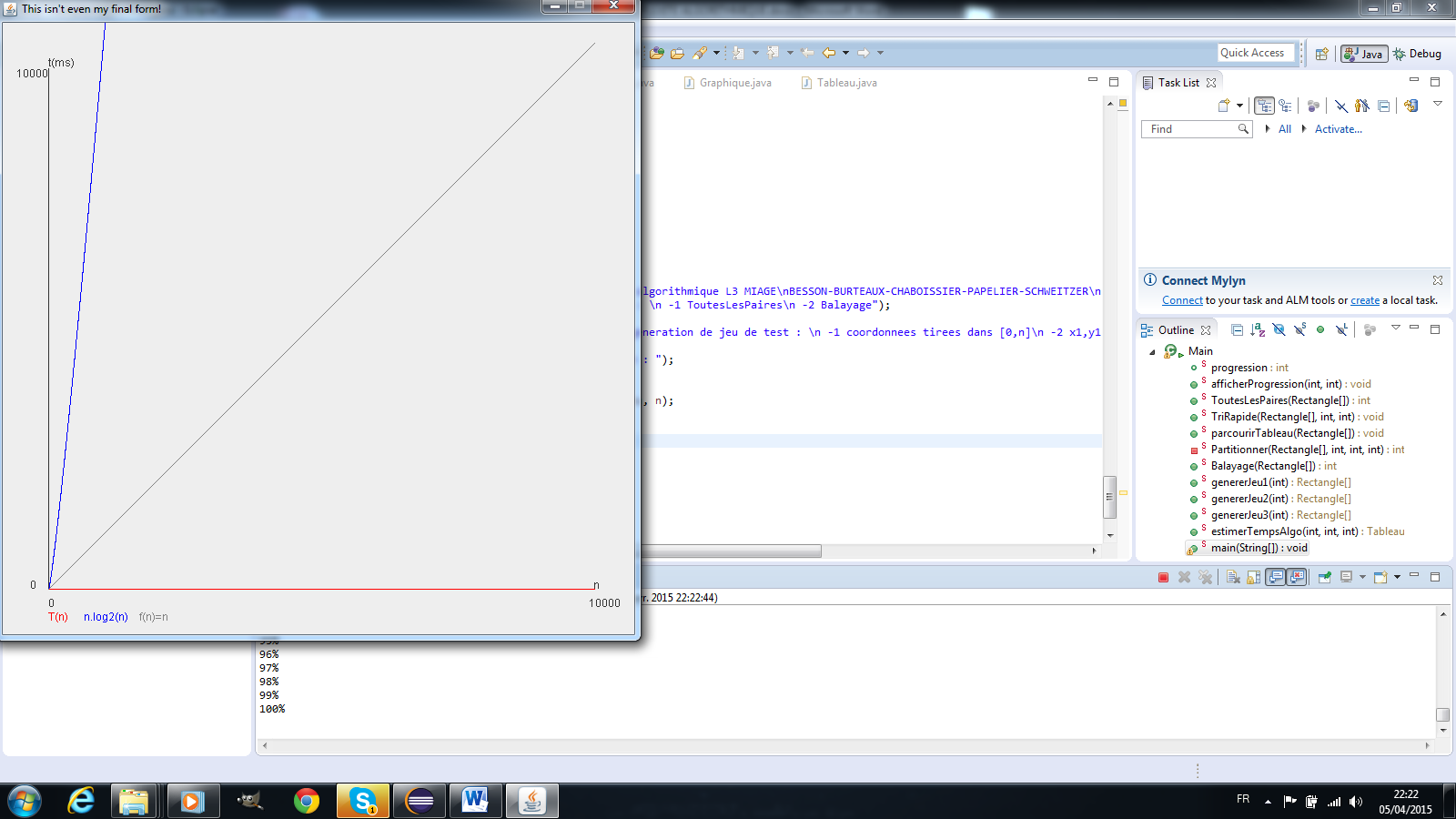
### Toutes les paires

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

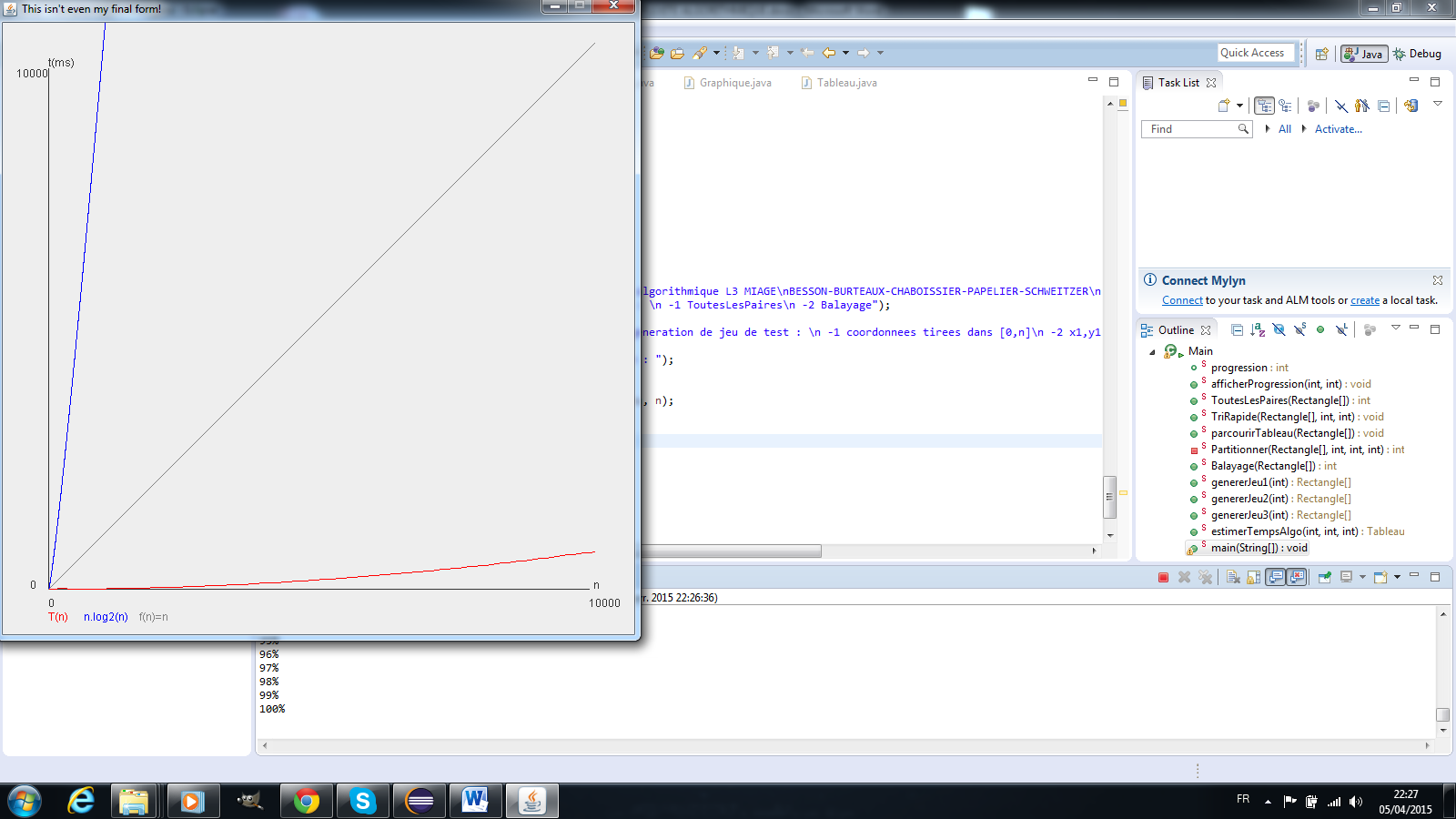
Voici le même graphique pour l’algorithme de Balayage. Etant donné qu’on a très peu de chance d’avoir des couples de rectangles qui se coupent, l’algorithme de balayage se contente d’itérer la première boucle n fois, sans jamais (ou presque) entrer dans la seconde boucle. Le temps d’execution est donc 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 : coordonnées tirées 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.

