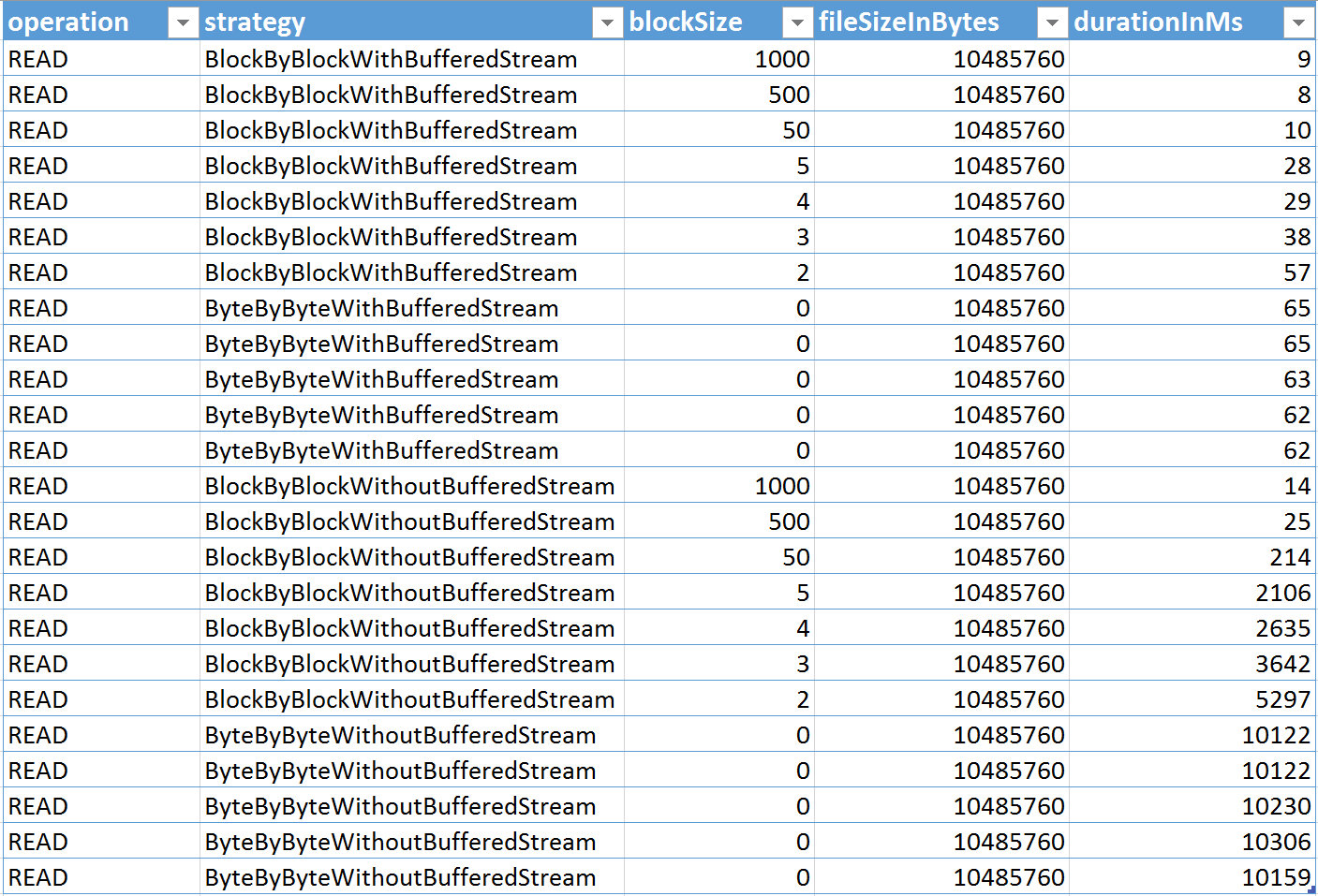
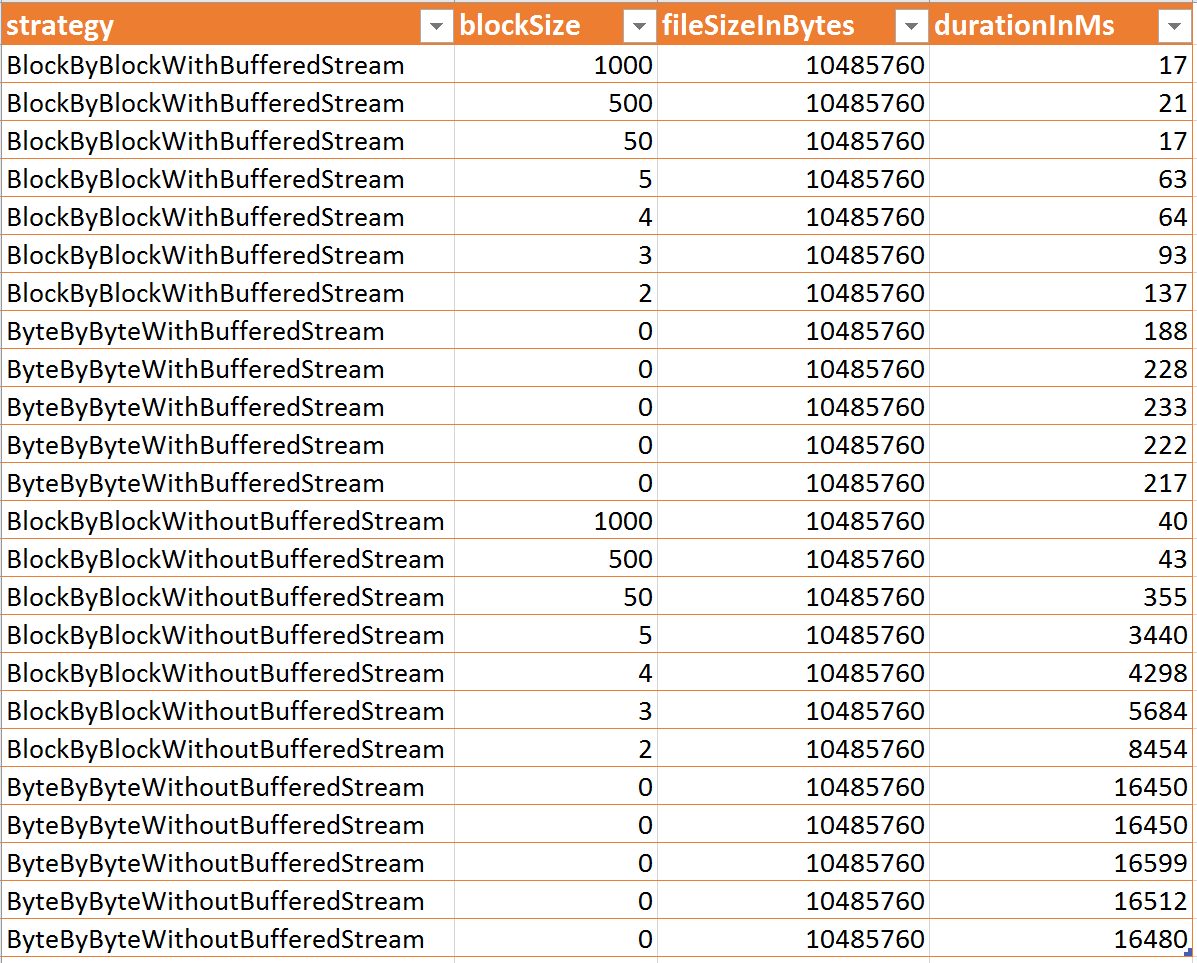
Laboratoire de la semaine 2 (29 février)

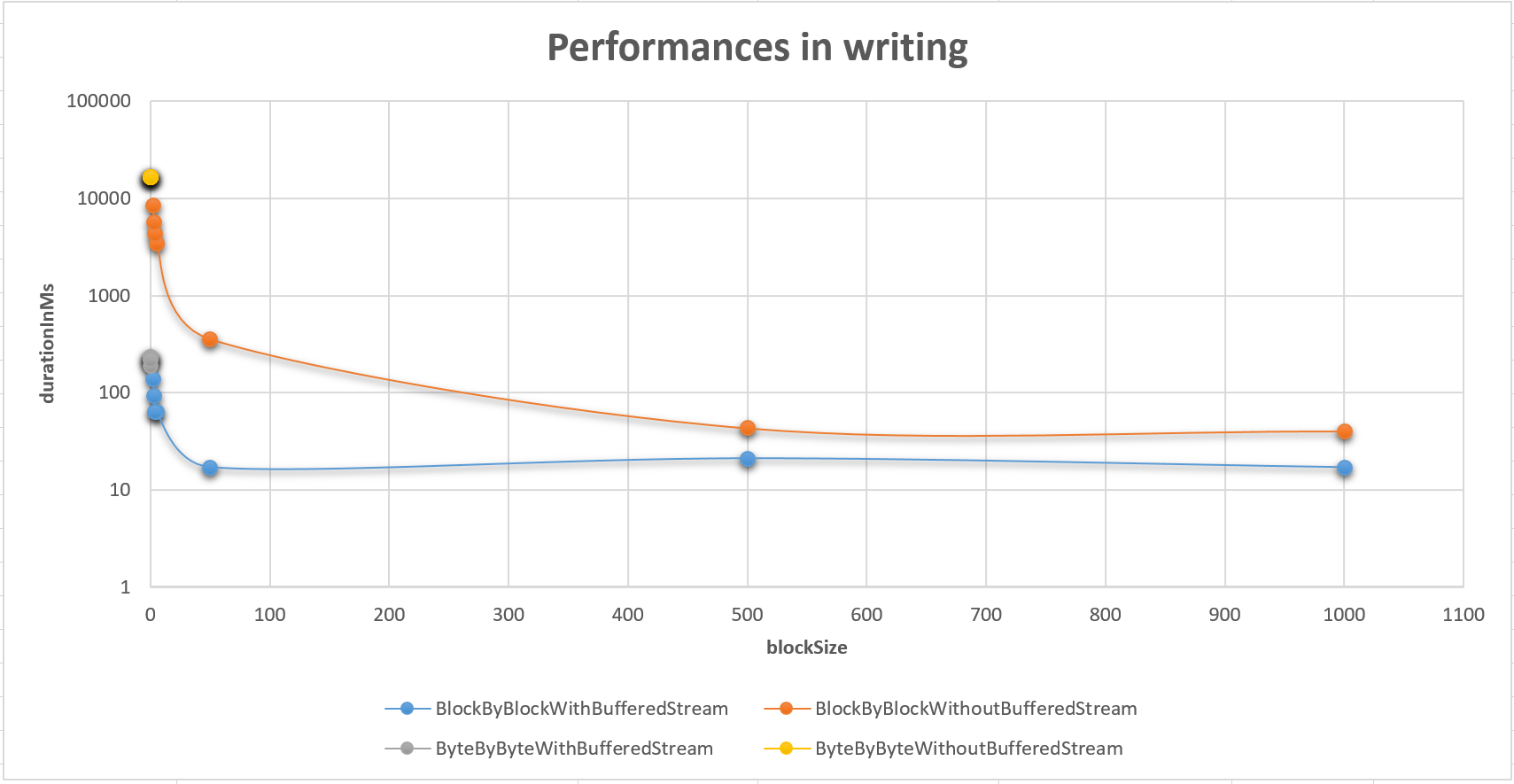
# Conditions de l’expérience

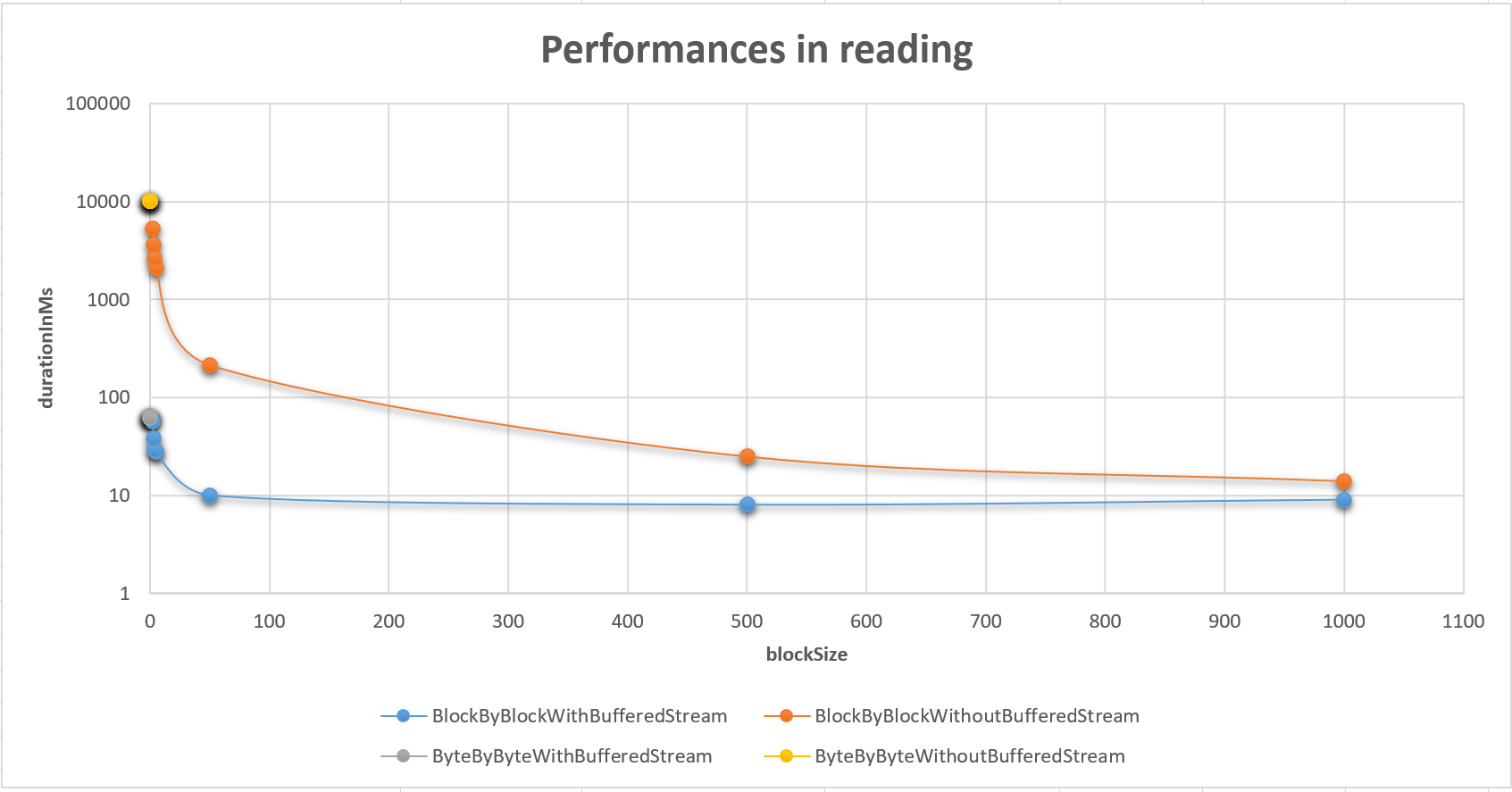
Cette expérience cherche à montrer que les performances peuvent être grandement améliorées lors d’écriture et de lecture de données en Java en utilisant simplement des tampons (« buffers » en anglais). Pour le déroulement de l’expérience, nous utilisons un programme écrit en Java (fichier « BufferedIOBenchmark.java ») qui va nous permettre d’écrire des données de tests dans différents fichiers et ensuite de relire les données de ces mêmes fichiers. Nous mettons ainsi en évidence le temps nécessaire pour l’écriture et la lecture de ces données suivant différentes stratégies. Une première stratégie consiste à utiliser des tampons avec une taille de bloc définie (nombre d’octets par bloc), une deuxième consiste à utiliser des tampons mais avec une taille de bloc égale à 0 (les données sont écrites et lues octet après octet), la troisième et quatrième stratégies consistent en la même chose que les deux premières mais en n’utilisant pas de tampons. L’expérience permet de mettre ainsi en évidence que l’écriture et la lecture des données est bien plus rapide en utilisant des tampons.

# Présentation des mesures

Notre programme permet également d’enregistrer dans un fichier au format « csv » les différents temps obtenus pour toutes les stratégies selon l’écriture et la lecture des données. Ci-dessous, nous avons les deux tableaux contenus dans le fichier « csv » généré ainsi que les deux graphiques, se rapportant aux deux tableaux, que nous avons dessinés en utilisant un tableur (Excel). Il est important de préciser que l’échelle utilisée pour représenter la durée en ms est logarithmique (en base 10) pour avoir de meilleurs graphiques. De plus, la taille du fichier dans lequel les données sont écrites est fixée et ne change pas (pour pouvoir effectuer les comparaisons).







# Analyse des mesures

Avec ces différentes mesures, nous remarquons bien sûr que l’utilisation de tampons permet d’améliorer significativement les temps d’écriture et de lecture des données. Que ce soit avec ou sans tampons, nous remarquons également que l’utilisation de blocs de données de plus en plus grand (en écriture et en lecture), nous donne des performances de plus en plus grandes. Si nous comparons l’écriture et la lecture des données, il y a une plus forte tendance à ce que les temps se rapprochent en lecture avec ou sans utilisation de tampons. Enfin, nous remarquons que l’utilisation de tampon même avec une taille de bloc égale à 0 donne des temps bien plus petits que sans l’utilisation de tampons que ce soit en écriture ou lecture.

# Modification du code du programme de base

Le programme qui nous était fourni au début du laboratoire ne permettait pas encore d’enregistrer les différents temps obtenus lors de l’expérience dans un fichier au format « csv ». Nous avons dû le modifier pour pouvoir le faire. Nous avons donc rajouté trois interfaces supplémentaires ainsi que trois classes implémentant ces trois interfaces. La première classe « ExperimentData » permet d’enregistrer les données que nous voulons écrire dans le fichier « csv » (comme le temps, la taille du fichier, la stratégie, …) dans une map (avec une association entre une clé et une valeur). Ces données correspondent donc aux données générées durant l’expérience. La deuxième classe « FileRecorder » permet de créer le fichier de sortie au format « csv » ainsi que le flux de sortie (avec tampon) connecté au fichier de sortie. Enfin la troisième classe « CsvSerializer » permet de sérialiser les données de l’expérience (récupérées depuis l’objet de la première classe) au format « csv » et de les écrire dans notre fichier de sortie. Le format « csv » correspond à différentes données séparées par des « , ». Après avoir créé ces trois classes, nous les avons utilisées dans la classe principale du programme. Dans cette dernière, nous avons rajouté un constructeur dans lequel nous initialisons un objet pour la sérialisation (troisième classe) et un autre pour créer le fichier « csv » voulu (deuxième classe). Finalement, nous créons un nouvel objet de données (première classe) à chaque fois que nous appelons la méthode pour écrire des données de test dans des fichiers et celle pour lire ces mêmes données dans ces mêmes fichiers.