Question 1 :

En réalisant plusieurs tests aléatoires, nous sommes parvenus au résultat suivant contenu dans la figure 1 ci-bas. En regardant le graphique, ce dernier nous convainc fortement que la structure de données utilisées occupe un espace linéairement proportionnel à la quantité de données.

Figure 1: Taille utilisée par la LinearSpacePerfectHashing selon le nombre d'éléments différents contenus à l’intérieur

Ensuite, quant à l’utilisation de la valeur p de 46 337, celle-ci possède plusieurs utilités. Tout d’abord, comme ce nombre est un nombre premier, celui-ci permet de minimiser les collisions lorsque nous utilisons cette valeur comme modulo. De plus, la valeur p limite la taille de la QuadraticSpacePerfectHashing. En effet, comme m < p et m = n2, alors n2 < p et donc n < sqrt(p). Ainsi, le nombre d’éléments maximal d’une QuadraticSpacePerfectHashing est égal à la racine carrée de p, soit 215 éléments. Quant à la LinearSpacePerfectHashing, comme elle utilise la QuadraticSpacePerfectHashing, cela limite donc pour elle aussi sont nombre maximal d’éléments. Néanmoins, il est improbable qu’il y ait plus de 215 éléments dans une seule table quadratique lorsqu’il y a autant de tables quadratiques qu’il y a d’éléments dans la LinearSpacePerfectHashing, surtout lorsque l’utilisation du nombre premier p dans le module diminue de beaucoup les nombres de collisions. Ainsi, comme n < p pour la table linéaire, le nombre d’éléments contenus dans celle-ci est alors limité à 46 336. Notre code reflète cette limite, car si on dépasse celle-ci, les méthodes allouant la mémoire affectent un pointeur nul aux données. Il n’est alors pas possible de créer les tables.