**PARTIE 1**

200 different valeur

2000000 fois chaque

(360,360)

(min max et median en milliseconde)

pour CP2 - min: 0 - max: 905 - median: 316

pour CP3 - min: 206 - max: 356 - median: 321

pour CP6 - min: 39 - max: 443 - median: 252

pour CP2 - min: 0 - max: 912 - median: 322

pour CP3 - min: 71 - max: 367 - median: 333

pour CP6 - min: 20 - max: 835 - median: 278

pour CP2 - min: 0 - max: 912 - median: 304

pour CP3 - min: 72 - max: 368 - median: 324

pour CP6 - min: 20 - max: 1028 - median: 255

pour CP2 - min: 0 - max: 1013 - median: 321

pour CP3 - min: 72 - max: 394 - median: 331

pour CP6 - min: 21 - max: 803 - median: 288

pour CP2 - min: 6 - max: 930 - median: 338

pour CP3 - min: 75 - max: 366 - median: 330

pour CP6 - min: 0 - max: 938 - median: 231

pour CP2 - min: 39 - max: 929 - median: 340

pour CP3 - min: 100 - max: 380 - median: 340

pour CP6 - min: 40 - max: 380 - median: 337

pour CP2 - min: 20 - max: 798 - median: 351

pour CP3 - min: 82 - max: 497 - median: 345

pour CP6 - min: 40 - max: 480 - median: 342

Différence entre les Designs

Design 2:

Très facile à coder parce que nous assumons que l’utilisateur utilisera le système polaire au lieux de cartésien la plupart du temps, ce code est plus vite que Design 3, la plupart du temps, lorsque un des nombre est en dessous de 100 et un deuxième nombre en dessous de 175. Autre que cela elle est plus lent.

Design 3:

Très facile à coder parce que nous assumons que l’utilisateur utilisera le système cartésien, ce code est plus vite, la plupart du temps, lorsque les 2 numéro sont dessus 100 et 175 respectivement.

Design 6:

Très difficile a coder parce que nous savons pas lequel de design utilisés, lorsque le person insère les coordonnées, le program trouvera laquelle des design sont meilleur. Ce code est plus compliqué à écrire mais est plus vite que seulement design 2 ou 3.

Pourquoi design 2 est des fois plus vite que design 3 et les tests que j’ai fait:

La raison pour ceci est parce que lorsque les designs font des changement de système de coordonnées, le calcule de design 2 à 3 est différent que de 3 à 2. Parce que de cela, un des design est meilleur que l’autre si nous assumons que plusieur changement seront fait. Avec l'essai erreur, j’ai trouver que en dessous de 100 et 175 respectivement, design 2 est plus vite. (cela a montrer ci dessous avec les essais). Le test était fait par des nombre aléatoire de 0 à 360, car il y a 360 degrés dans un tourne complet. J’ai fait le teste plusieurs fois pour que la médiane du design 6 soit le plus bas.

**PARTIE 2**

(ArrayList et Vecteur commence a 10, donc default)

11338 milliseconde - 400000000 - Array

Array Summation time - 194

9726 milliseconde - 400000000 - Array

Array Summation time - 156

30247 milliseconde - 400000000 - ArrayList

ArrayList Summation time - 486

29817 milliseconde - 400000000 - ArrayList

ArrayList Summation time - 467

28367 milliseconde - 400000000 - Vectors

Vector Summation time - 9773

29153 milliseconde - 400000000 - Vectors

Vector Summation time - 9942

(ArrayList et Vecteur commence a 40000000, donc max size)

9979 milliseconde - 400000000 - Array

Array Summation time - 162

13445 milliseconde - 400000000 - ArrayList

ArrayList Summation time - 440

26973 milliseconde - 400000000 - Vectors

Vector Summation time - 10154

Array:

Array était le plus vite entre les 3, cependant, cela est parce que sa taille était prédéterminé. L’array est la méthode de base lorsque la taille ne changeras pas, et elle est plus vite que les 2 autres méthodes de cette situation.

ArrayList:

ArrayList est un array qui change sa taille lorsqu'elle devient plein. Lorsqu’elle devient pleine, elle agrandit de 50% de sa taille en ce moment. Ceci est plus vite que les 2 autres méthodes lorsque la taille n’est pas définie mais le montant de données devrait pas augmenter trop. La raison pour cela est qu’elle sauvent plus de mémoire que les vecteurs parce qu’elle croit de seulement 50%. (elle augmente de moins de mémoire à chaque fois).Donc avec le default size de 10, pour se rendre à 400000000 éléments, elle utiliserait assez de memoire pour 55977744 elements.

Vector:

Le Vecteur est un array qui change sa taille lorsqu’elle devient pleins aussi. Cependant elle double sa taille chaque fois et donc est moins efficace avec la mémoire si la situations montre une augmentation de données minuscules. De l’autre côté, les vecteurs sont vraiment efficace en termes de temps lorsque le montant de donné n’est pas déterminé et augmente de beaucoup. La raison pour cela est parce que le vecteur va doubler sa taille moins que le arrayList (parce que le arraylist augmente de 50% au lieux de 100%). Donc avec le default size de 10, pour se rendre à 400000000 éléments, elle utiliserait assez de memoire pour 671088640 elements.

Addition et Suggestion:

Entre les 3, le array normale semble accomplir les additions les plus vites, suivi par le ArrayList ensuit les Vecteurs. La raison pourquoi les vecteurs seront toujours moins vite que les arrays et arrayLists est parce que les vecteurs sont synchroniser. C’est a dire, si il y a plus qu’un thread qui appelle à un vecteur, elle n’aura pas de problème. Cependant, un array ou arrayList lancerai l’exception ConcurrentModificationException si un thread essaye de change l’information sur le array durant un appelle à une autre thread. Pour cette raison, les vecteurs devrait seulement être piger lorsque vous savez que le nombre d’élément vas changer constamment à grand proportions (de plus nous pouvons ajuster le coefficient de croissance). Le arrayList devrait être piger pour tous autres occasions, parce que la différence en efficacité est petit comparer au possibilité que l’array normal overflow. Il est par contre important a noté que les arrays sont plus facile à lire et comprendre que les arraylists. De plus, il est plus vite à écrire des arrays que arrayList, donc si le nombre d’éléments ne changeront pas, les arrays sont une très bonne façons de sauver du temps durant la production de code.