Introduction

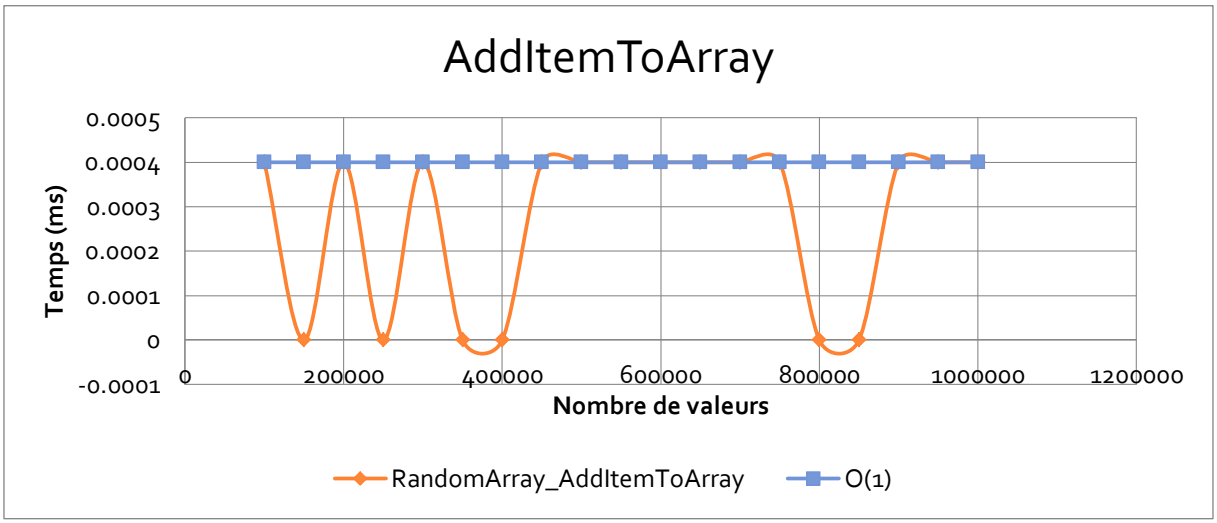
Le but de cet exercice est de déterminer la complexité (temporelle) des algorithmes dans la classe BigONotation.

Cas moyen

Pour le cas moyen, le tableau est non trié. On ne pourra pas vérifier le cas le moins favorable.

addItemToArray O(1)

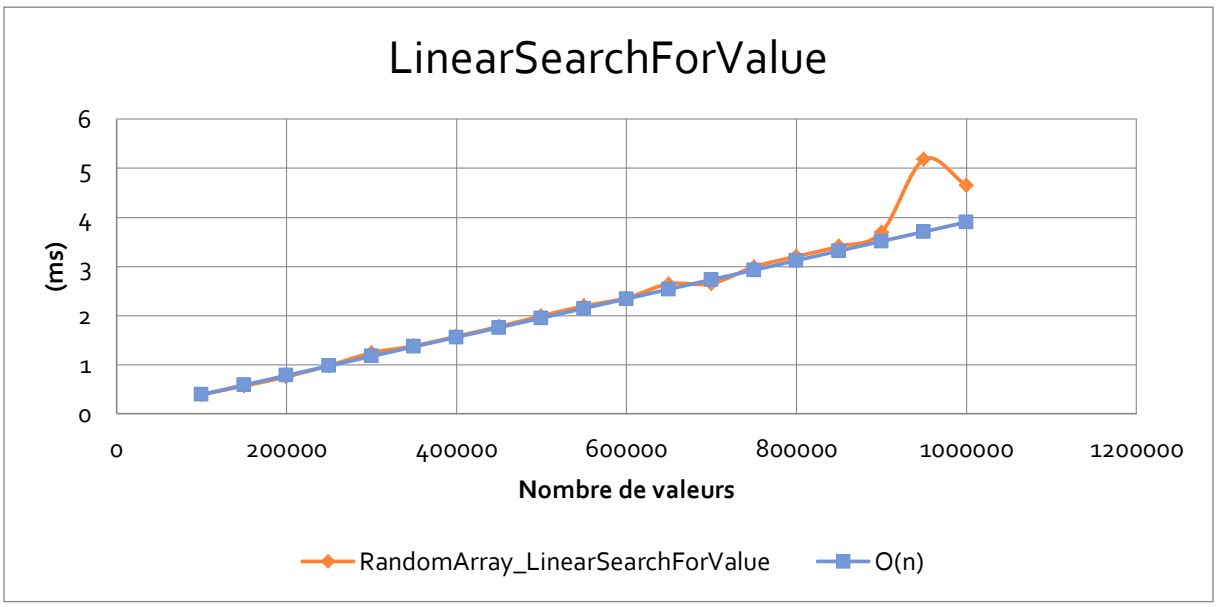
Cette méthode a une complexité constante et aura toujours le même temps d'exécution. J'ai pris le premier temps que fait la méthode et comparer à tous les autres temps d'exécution de valeurs.



linearSearchForValue O(n)

Cette méthode a une complexité linéaire. Elle fait simplement de parcourir une liste.

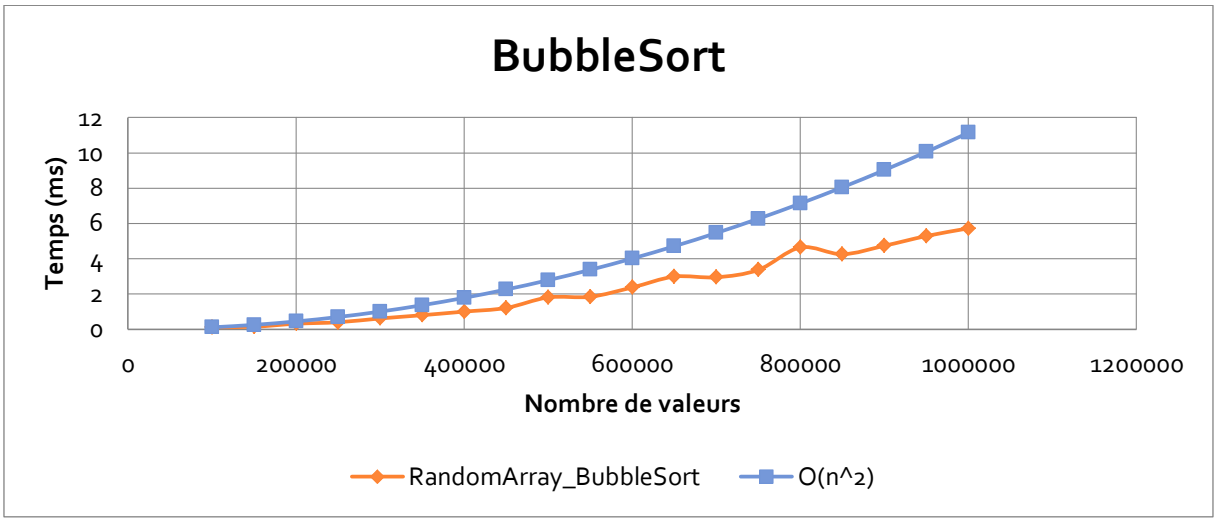
Equation en bleu : (Temps pour 100'000 /100'000) * Valeurs



bubbleSort O(n^2)

Cette méthode a une complexité quadratique.

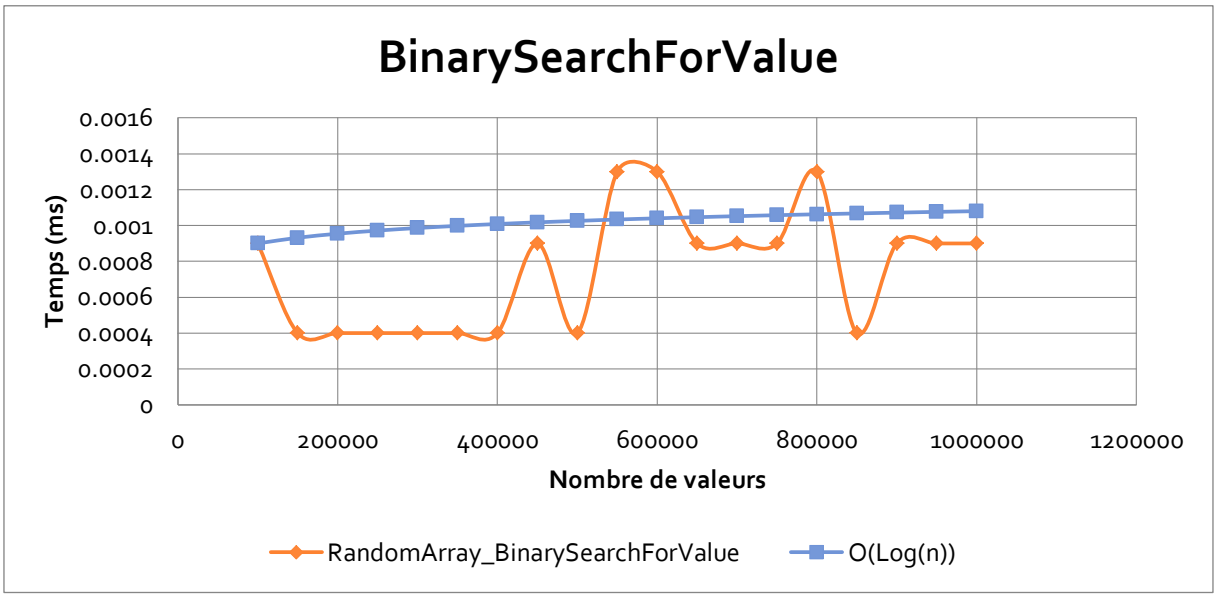
Equation en bleu : (Temps pour 100'000 / 100'000^2) * Valeurs^2



binarySearchForValue O(log n)

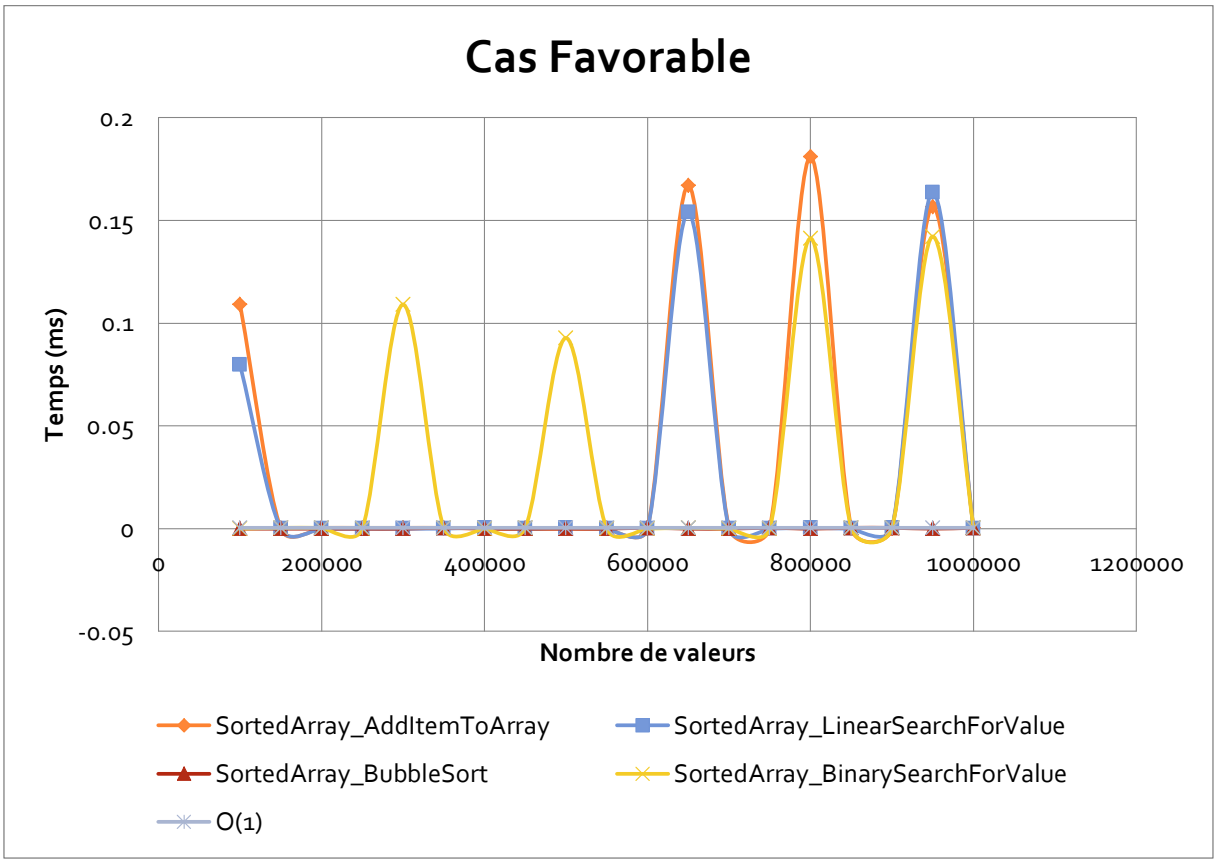
Cette méthode a une complexité logarithmique.

Equation en bleu : (Temps pour 100'000 / Log(100'000)) * Log(Valeurs)



Cas Favorable

Pour les cas favorable on ne peut pas déduire une complexité car la complexité de la methode d'ajout d'un élément dans un tableau sera toujours constante alors qu'ici il est montré qu'à 800'000 valeurs elle prend environ 0.18 (ms).



Conclusion

La complexité d'un algorithme est très utiles pour savoir comment va réagir notre méthodes face à des données qui seront conséquent. Dans une autre manière elle peut nous pousser à trouver toujours une solutions qui mieux que la précédente.