Preuve d'algorithmes

I Qu'est ce qu'une preuve d'un algorithme ?

Définition : Réaliser la preuve d'un algorithme, c'est :

- Prouver qu'il se termine : On parle de **terminaison.**
- Prouver qu'il fait bien ce que l'on attend de lui : On parle de **correction.**

Terminaison : Il s'agit de vérifier ici que les calculs effectués par l'algorithme s'arrêtent bien. Notamment, lorsqu'une boucle conditionnelle est effectuée par l'algorithme, il est primordial de vérifier que l'on sort bien de cette boucle. On parle aussi de **variants de boucle**.

Correction : Il s'agît ici de prouver que l'algorithme fait bien ce qu'on lui demande. Pour cela, on va chercher **un invariant de boucle.** C'est une propriété qui est vérifiée avant l'entrée dans la boucle, qui est vérifiée à chaque itération de la boucle et qui amène au résultat escompté à la sortie de la boucle.

II Exemple

On va réaliser la preuve d'un algorithme qui à un entier n donné associe sa puissance de 2.

- Préciser ce qui est donné en entrée, ce que l'on obtient en sortie.
- Ecrire un algorithme réalisant la tache demandée.
- Montrer que quelque soit l'entier naturel donné, l'algorithme se termine bien.
- Essayer de déterminer un invariant de boucle, c'est-à-dire une condition qui est vérifiée à toutes les étapes de l'algorithme.

Vérifier que cet invariant est vérifié à l'étape initiale, écrivez le au début de la boucle et vérifiez qu'il est toujours valable à la fin de l'exécution de cette boucle.

Exercice

Reprendre l'algorithme de calcul de la somme des n premiers entiers naturels, prouvez sa terminaison et trouvez l'invariant de boucle.

III L'algorithme de dichotomie

On cherche une valeur dans un tableau trié.

Principe de la méthode : On compare la valeur cherchée à celle de l'indice du milieu de la liste (approximativement le milieu : On prend le rang correspondant à la partie entière de la moyenne de la borne inf (a) et sup(b) : (a+b)//2 .

Si les deux valeurs sont égales, c'est terminé, sinon, on ne va conserver que la partie du tableau où peut se trouver la valeur cherchée et ainsi de suite jusqu'à rencontrer la valeur ou obtenir une liste vide .

Exemple : Voici une liste de nombres : [1 ;2 ;5 ;8 ;12 ;15 ;21 ;24]. On cherche la valeur 3 (oui, c'est facile)

- On divise la liste en deux : On tombe sur l'indice 4, dont la valeur 8 est plus grande que 2. On garde donc le début de liste plus exactement les rangs 1,2 et 3. (attention en python, ce serait 0,1 et 2) . On compare avec la valeur du milieu ,2. On ne garde que la partie de la liste supérieure à 2. Soit 5. Qui n'est pas égale à 3. On sort de la boucle, on a testé toute la liste .

Etape	Rang inf	Rang sup	Rang du milieu	Valeur rang milieu
1	1	8	4	8 >2
2	1	3	2	2<3
3	3	3	3	5>3

Exercice

Voici une liste de 12 entiers : [-5,-2,0,1,2,5,8,9,14,21,52,100].

1. Combien faudra-t-il au maximum d'étapes pour pouvoir assurer de la présence ou non d'un entier dans ce tableau ?
2. On cherche l'entier 1 . Réaliser le tableau comme vu précédemment.
3. Ecrire un algorithme qui réponde au problème. En entrée, on aura une liste triée, en sortie la présence ou non de l'élément cherché.
4. Implémentez cet algorithme en langage python dans un programme appelé dichotomie.py
5. Application : Télécharger le fichier english_words et enregistrez dans le même dossier que votre programme.

6. Recherchez la présence d'un mot anglais de votre choix.
7. Modifiez votre programme pour connaître le nombre d'étapes nécessaires pour répondre à votre problématique (présence ou non d'un élément). Choisissez 5 mots et comparez le nombre d'étapes nécessaires.
8. Prouvez la terminaison de votre algorithme : Pour cela montrez que l'on va sortir de la boucle, quelle que soit la longueur du tableau initial. (Hum, on dirait des maths !). La variant de boucle peut être la taille de l'intervalle étudié.
9. Et la complexité ?
   Bon ici , il fait un peu d'aide : A chaque étape, on divise le tableau en 2. Dans le pire des cas, on cherche un entier i tel que $\frac{n}{2^i}$ = 1 , ce qui revient à cherche i tel au bout de i divisions par 2 d'un tableau de taille n, on aura un tableau de taille 1.

   Mathématiquement, on a $\frac{n}{2^i}$ = 1 donc n = 2^i . Le log de base 2, noté $\log_2$ est tel que $\log_2(2) = 1$. On a donc ici :

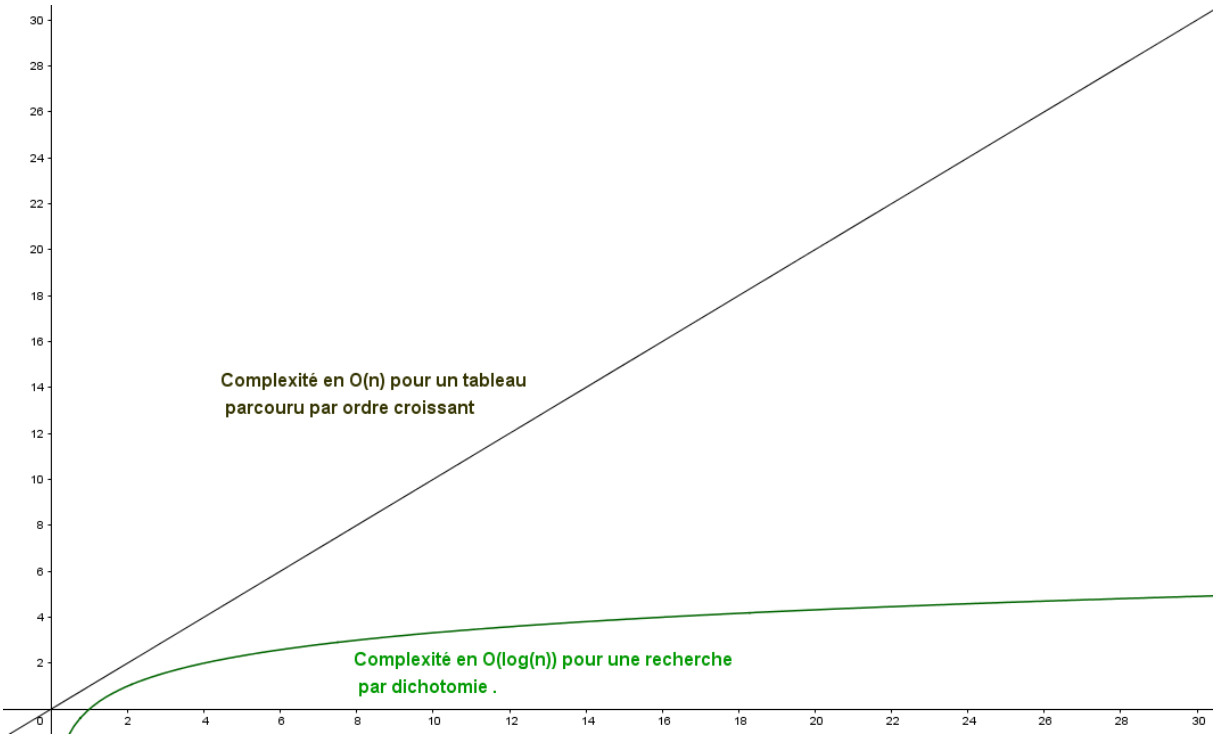
   $$\log_2(n) = i.\log_2(2) = i .$$

   Pour parcourir par dichotomie un tableau de taille n , il faut dans le pire des cas $\log_2(n)$ étapes.

   La complexité est ici en O**($\log_2(n)$)**

   Remarque . Si l'on parcourt un tableau trié (ou non d'ailleurs) à la recherche d'un élément en comparant au premier élément, puis deuxième…on peut trouver plus rapidement la solution certes mais dans le pire des cas, il nous faut parcourir tout le tableau et donc aller jusqu'à n étapes.

   Le graphique ci-dessous semble plaider pour la recherche dichotomique. Il faut cependant ajouter le coût du tri du tableau, ce dont n'a pas besoin la parcours par ordre croissant d'indices d'un tableau.


Complexité en O(n) pour un tableau
parcouru par ordre croissant
Complexité en O(log(n)) pour une recherche
par dichotomie .