17

Algorithmique IV - Dichotomie

Extrait du programme

Thème : Algorithmique

- **Contenus :**
  Recherche dichotomique dans un tableau trié

Capacités attendus :
Montrer la terminaison de la recherche dichotomique à l'aide d'un variant de boucle.

Commentaires :
Des assertions peuvent être utilisées.
La preuve de la correction peut être présentée par le professeur.

Introduction

Jouons à un jeu.
J'ai choisi un nombre entre 1 et 1000 et vous devez le retrouver en me proposant des nombres. A chacune de vos propositions, je répondrai soit « plus haut », « plus bas » ou « gagné ».
Vous pourriez proposer des nombres totalement au hasard, mais il y a fort à parier que le dialogue sera le suivant, en supposant que j'ai choisi le nombre 666.
- 500 !
- plus haut.
- 750 !
- plus bas.
- 625 !
- plus haut.
- 687 !
- plus bas.
- 656 !
- plus haut.
- 671 !
- plus bas.
- 663 !
- plus haut.
- 667 !
- plus bas.
- 665 !
- plus haut.
- 666 !
- Gagné ! ! ! ! !

Félicitation mon ami. Tu devais retrouver un nombre choisi entre 1 et 1000. Si tu avais choisi tes nombres totalement au hasard, tu aurais pu le trouver entre 1 coup [1] et 1000 coups [2]. Grâce à ta méthode, tu as trouvé le nombre en seulement 10 coups !
Mais quelle est cette méthode ?

Le principe est, à chaque étape, de ne garder que la moitié des nombres. C'est ce qu'on appelle la méthode de la **dichotomie** [3].

1. La chance ! tu as trouvé le nombre totalement au hasard en un seul coup. Tu dois t'appeler Nosferatu.
2. Autant le dire, que tu n'as pas eu de chance.
3. Du grec ancien dikhotomia « division en deux parties »

I. Algorithme de recherche dichotomique

a) Le principe

Pour rechercher une valeur dans un tableau par l'algorithme de recherche dichotomique **il faut au préalable avoir un tableau de valeurs classées** dans un ordre croissant ou décroissant. Dans la suite nous choisirons un tableau rangé dans l'ordre croissant. Le principe consiste à couper le tableau en 2. Avec la valeur médiane, on peut déterminer si le nombre recherché est dans la partie inférieure ou supérieure du tableau. On ne garde alors plus que cette partie et on recommence le processus.

Reprenons notre exemple introductif. On cherche un nombre entre 1 et 1000. On ne le saura qu'à la fin, mais ce nombre est 666. A chaque étape, nous définissions le minimum, le maximum des valeurs restantes dans notre recherches ainsi que la médiane, obtenu en effectuant la division entière par 2 de la somme du minimum par le maximum. Nous utiliserons le symbole `//` qui en Python représente la division entière [4] [5]. La médiane est donc obtenue en faisant (mini + MAXI)//2.

minimum : 1	Médiane : $(1+1000)//2 = 500$	MAXIMUM : 1000	Comme le nombre recherché est plus grand que 500, on ne va garder que les valeurs entre 500 et 1000.
minimum : 500	Médiane : $(500+1000)//2 = 750$	MAXIMUM : 1000	Comme le nombre recherché est plus petit que 750, on ne va garder que les valeurs entre 500 et 750.
minimum : 500	Médiane : $(500+750)//2 = 625$	MAXIMUM : 750	Comme le nombre recherché est plus grand que 625, on ne va garder que les valeurs entre 625 et 750.
minimum : 625	Médiane : $(625+750)//2 = 687$	MAXIMUM : 750	Comme le nombre recherché est plus petit que 687, on ne va garder que les valeurs entre 625 et 687.
minimum : 625	Médiane : $(625+687)//2 = 656$	MAXIMUM : 687	Comme le nombre recherché est plus grand que 656, on ne va garder que les valeurs entre 656 et 687.
minimum : 656	Médiane : $(656+687)//2 = 671$	MAXIMUM : 687	Comme le nombre recherché est plus petit que 671, on ne va garder que les valeurs entre 656 et 671.
minimum : 656	Médiane : $(656+671)//2 = 663$	MAXIMUM : 671	Comme le nombre recherché est plus grand que 663, on ne va garder que les valeurs entre 663 et 671.
minimum : 663	Médiane : $(663+671)//2 = 667$	MAXIMUM : 671	Comme le nombre recherché est plus petit que 667, on ne va garder que les valeurs entre 663 et 667.
minimum : 663	Médiane : $(663+667)//2 = 665$	MAXIMUM : 667	Comme le nombre recherché est plus grand que 665, on ne va garder que les valeurs entre 665 et 667.

4. Avec les « vraies » symboles mathématiques nous devrions écrire $\left\lfloor \frac{\text{mini} + \text{MAXI}}{2} \right\rfloor$.

5. Petite curiosité, dans des langages comme Java par exemple, si tous les nombres sont des entiers, le symbole / est déjà celui d'une division entière. On pourrait alors écrire `(mini + MAXI)/2`

minimum : 665	Médiane : $(663+667)//2 = 666$	MAXIMUM : 667

Nous avons trouvé le nombre 666 !

b) Le principe sur un tableau

Pour une recherche dans un tableau, nous allons utiliser le même procédé en modifiant uniquement une chose. La médiane. Puisque nous sommes dans un tableau, nous avons besoin de connaître la valeur médiane et pour cela, il nous déterminer l'**indice médian**. Voyons plutôt cela avec un exemple [6].
Nous avons le tableau `T` de nombres entiers rangé dans l'ordre croissant : `T = [1, 3, 4, 6, 7, 8, 10, 13, 14]` et nous recherchons le nombre 6.

Indice	mini=0				Med = (0+8)//2 = 4				MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus petit que `T[Med]=T[4]=7`, on ne va garder que la partie du tableau entre les indices `mini=0` et `Med=4`. On rejette la partie supérieure du tableau (en rouge dans la suite).

Indice	mini=0		Med = (0+4)//2 = 2		MAXI = 4				
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus grand que `T[Med]=T[2]=4`, on ne va garder que la partie du tableau entre les indices `Med=2` et `MAXI=4`. On rejette la partie inférieure (en rouge dans la suite).

Indice			mini=2	Med = (2+4)//2 = 3	MAXI = 4				
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est exactement le nombre `T[Med]=T[3]=6`, nous avons terminé notre travail. **Nous pouvons donc conclure que le nombre 6 est bien présent dans le tableau et qu'il se situe à la case indexé 3.**

6. Cet exemple a été honteusement inspiré de l'exemple de la fiche wikipédia ... `https://fr.wikipedia.org/wiki/Recherche_dichotomique`

c) (Légère) amélioration du principe

Vous avez bien regardé le principe, si la valeur recherchée est la médiane nous avons terminé, sinon, nous devons prendre une moitié du tableau restant. Cette moitié de tableau est soit les valeurs allant de `T[mini]` à `T[Med]` soit les valeurs allant de `T[Med]` à `T[MAXI]`. Or comme la valeur recherchée n'est pas la valeur médiane, on peut très légèrement améliorer notre algorithme en supprimant la valeur médiane. Autrement dit, on ne garde que les valeurs allant de `T[mini]` à `T[Med-1]` soit les valeurs allant de `T[Med+1]` à `T[MAXI]`.
Reprenons notre exemple précédent avec le tableau `T = [1, 3, 4, 6, 7, 8, 10, 13, 14]` et la recherche du nombre 6.

Indice	mini=0				Med = (0+8)//2 = 4				MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus petit que `T[Med]=T[4]=7`, on ne va garder que la partie du tableau entre les indices `mini=0` et `Med-1=3`. On rejette la partie supérieure du tableau (en rouge dans la suite).

Indice	mini=0	Med = (0+3)//2 = 1		MAXI = 3					
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus grand que `T[Med]=T[1]=3`, on ne va garder que la partie du tableau entre les indices `Med+1=2` et `MAXI=3`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice			mini =2 Med = (2+3)//2 = 2	MAXI = 3					
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est plus grand que `T[Med]=T[2]=4`, on ne va garder que la partie du tableau entre les indices `Med+1=3` et `MAXI=3`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice				mini =3 Med = (3+3)//2 = 3 MAXI = 3					
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 6 est exactement le nombre `T[Med]=T[3]=6`, nous avons terminé notre travail. **Nous pouvons donc conclure que le nombre 6 est bien présent dans le tableau et qu'il se situe à la case indexé 3.**

Contrairement à ce que l'on pourrait penser, ceci est un très bon exemple. On a compris que globalement, dans l'algorithme de recherche dichotomique, à chaque étape on coupe le tableau en 2. Avec notre petite amélioration, à chaque étape, on coupe notre tableau en 2 et on enlève une valeur. Cette amélioration n'apparaît pas dans notre exemple car avec la méthode simple, on trouve notre valeur 6 en 3 coups. Avec la méthode améliorée, on trouve la valeur 6 en 4 coups [7].
Il faut toujours penser global. Il existe (et il existera) toujours des cas un peu particuliers pour lesquels un algorithme simple sera très efficace. Mais dans la plupart des cas (voire dans la majorité), cet algorithme sera très peu efficace.

7. Tu parles d'une amélioration !

d) Et si le nombre n'est pas dans le tableau ?

Reprenons notre exemple du tableau `T = [1, 3, 4, 6, 7, 8, 10, 13, 14]` et recherchons le nombre 12.

Indice	mini=0				Med = (0+8)//2 = 4				MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 12 est plus grand que `T[Med]=T[4]=7`, on ne va garder que la partie du tableau entre les indices `Med+1=5` et `MAXI=8`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice						mini = 5	Med = (5+8)//2 = 6		MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 12 est plus grand que `T[Med]=T[6]=10`, on ne va garder que la partie du tableau entre les indices `Med+1=7` et `MAXI=8`. On rejette la partie inférieure du tableau (en rouge dans la suite).

Indice								mini = 7 Med = (7+8)//2 = 7	MAXI = 8
Tableau `T`	1	3	4	6	7	8	10	13	14

Comme le nombre 12 est plus petit que `T[Med]=T[7]=13`, on ne va garder que la partie du tableau entre les indices `mini=7` et `Med-1=7-1=6`...
Euh... Il y a un problème, car 7 n'est pas plus petit que 6...
Oui, donc c'est la preuve que nous avons terminé notre recherche.

Le nombre 12 n'est pas dans le tableau !

II. Étude de l'algorithme de recherche dichotomique

a) Spécifications

Rôle :	Trouver une valeur dans un tableau.
Entrées :	Un tableau T de longueur n. Une valeur v.
Préconditions :	Les valeurs de T sont comparables. Le tableau est trié dans l'ordre croissant. La valeur v est comparable aux valeurs de T.
Sortie :	Un nombre entier *indice* entre -1 et $n-1$.
Postconditions :	Si la valeur v n'est pas dans le tableau T, alors *indice* vaut -1. Sinon, *indice* est un nombre entier tel que $0 \leq indice \leq n-1$ et $T[indice] = v$.

b) Algorithme de recherche dichotomique

L'algorithme de recherche :

Entrées : un tableau T de longueur n rangé dans l'ordre croissant
une valeur v
Sorties : un entier i entre -1 et $n-1$

```
1  min ← 0 ;
2  med ← 0 ;
3  max ← n − 1 ;
4  tant que min < max faire
5  |  med ← (min + max)//2 ;
6  |  si T[med] < v alors
7  |  |  min ← med + 1 ;
8  |  sinon
9  |  |  si T[med] > v alors
10 |  |  |  max ← med − 1 ;
11 |  |  sinon
12 |  |  |  min ← med ;
13 |  |  |  max ← med ;
14 |  |  fin si
15 |  fin si
16 fin tq
17 si T[min] == v alors
18 |  indice ← min ;
19 sinon
20 |  indice ← −1 ;
21 fin si
```

La version en langage Python :

```python
mini = 0
med = 0
maxi = n-1
while mini < maxi:
        med = (mini + maxi)//2
        if T[med]< v :
                #v est dans la partie supérieure du tableau
                mini = med +1
        elif T[med] > v :
                #v est dans la partie inférieure du tableau
                maxi = med -1
        else :
                maxi = med
                mini = med
if T[mini]==v :
        indice = mini
else:
        indice = -1
```

Remarque

Pourquoi ai-je préféré appeler mes variables `mini` et `maxi` et non pas `min` et `max` ?

.....
.....
.....

c) Terminaison

Nous avons une boucle **TANT ... QUE**, il faut déterminer le **variant**.
Pour mémoire, le variant est une expression :

- impliquée dans la répétitive ;
- à valeur entière ;
- qui évolue de façon strictement décroissante au cours des itérations ;
- qui est positive ou nulle du fait des initialisations et de la condition de la répétitive.

Nous allons ici choisir ... $maxi - mini$. En effet :

- $maxi$ et $mini$ sont impliquées dans la répétitive (ligne 4) ;
- à valeurs entières.
  En effet on n'effectue que des opérations sur des entiers (+1, -1, division entière, ...)
- qui évolue de façon strictement décroissante au cours des itérations ;
  Soit (ligne 7) $mins$ est remplacé par $med + 1 = (mins + maxs)//2 + 1$, autrement dit une valeur supérieure à la précédente. $maxs - mins$ décroît.
  Soit (ligne 10) $maxi$ est remplacé par $med - 1 = (mini + maxi)//2 - 1$, autrement dit une valeur inférieure à la précédente. $maxi - mini$ décroît.
  Soit (lignes 12 et 13) $maxi = mini = med$ et donc $maxi - mini$ décroît aussi car pour rentrer dans la boucle on doit avoir $mini < maxi$, c'est-à-dire $maxi - mini > 0$.
- qui est positive ou nulle du fait des initialisations et de la condition de la répétitive.
  Pour que l'on boucle, on doit avoir $mini < maxi$ autrement dit, $maxi - mini > 0$.

Donc cet algorithme termine.

d) Correction partielle

Pour déterminer si l'algorithme effectue bien son travail, on doit déterminer son **invariant**. A nouveau, pour mémoire, un invariant est :

- est un prédicat sur les variables impliquées dans la répétitive,
- est vrai juste avant la répétitive,
  (induite par la précondition et les instructions en début d'algorithme)
- est vrai après chaque itération de la répétitive
  (induite par l'invariant et condition+instructions de l'itération courante)
- sera vrai après la fin de la répétitive

Pour déterminer cet invariant on va s'inspirer fortement de la postcondition et imaginer les différentes étapes d'exécution de l'algorithme.
Pour mémoire, la postcondition est :

> **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
> **Sinon**, *indice* est un nombre entier tel que $0 \leq indice \leq n-1$ et $T[indice] = v$.

On peut s'en inspirer pour déterminer l'invariant à la i-ème étape. Comme alors nous n'avons déjà éliminé toutes les case du tableau inférieures à min et supérieures à max, on a :

> **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
> **Sinon**, *indice* est un nombre entier tel que $min \leq indice \leq max$ et $T[indice] = v$.

Allez, un peu de courage ! On effectue la correction partielle !

1. Initialisation de l'invariant : pour $i = 0$.
   $min = 0$, $max = n-1$. Soit la valeur est dans le tableau, soit elle ne l'est pas.

   > **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
   > **Sinon**, *indice* est un nombre entier tel que $mini = 0 \leq indice \leq maxi = n-1$ et $T[indice] = v$..

   L'invariant est vrai.

2. Regardons le passage de la $(i-1)$-ième étape à la i-ième étape, afin de vérifier l'invariant. Supposons donc que l'invariant est vrai à la $(i-1)$-ième étape. Autrement dit, on a :

   > **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
   > **Sinon**, *indice* est un nombre entier tel que $mini \leq indice \leq maxi$ et $T[indice] = v$.

   Déroulons l'algorithme à la i-ème étape.
   Comme on ne modifie pas le tableau, si v n'y est pas, il n'y sera jamais. Intéressons-nous au cas où v est dans le tableau.
   Ligne 4, comme $mini < maxi$, on a ligne 5 $mini \leq med < maxi$.
   Comme le tableau est trié et qu'il existe un entier *indice* tel que $mini \leq indice \leq maxi$ avec $T[indice] = v$, on a 2 possibilités :
   - soit $T[mini] \leq T[med] < v \leq T[maxi]$
     La ligne 6 est vérifiée. On peut ne garder que $T[med] < v \leq T[maxi]$, mais comme il n'y a pas de nombre entre $T[med]$ et $T[med+1]$ (2 cases consécutives) et que l'on a supposé que v est dans le tableau, on a :
     $T[med+1] \leq v \leq T[maxi]$
     Ligne 7, $mini \leftarrow med+1$ et ainsi :
     $T[mini] \leq v \leq T[maxi]$ avec la nouvelle valeur de $mini$.
   - soit $T[mini] \leq v < T[med] \leq T[maxi]$
     La ligne 9 est vérifiée. On peut ne garder que $T[mini] \leq v < T[med]$, mais comme il n'y a pas de nombre entre $T[med-1]$ et $T[med]$ (2 cases consécutives) et que l'on a supposé que v est dans le tableau, on a :
     $T[mini] \leq v \leq T[med-1]$
     Ligne 10, $maxi \leftarrow med-1$ et ainsi :
     $T[mini] \leq v \leq T[maxi]$ avec la nouvelle valeur de $maxi$.

- soit $T[med] = v$
  La ligne 11 est vérifiée. Comme après les lignes 12 et 13, on a $mini = maxi = med$, on a bien :
  $T[med] = T[mini] \leq v \leq T[maxi] = T[med]$

Ainsi, après la i-ème étape, on a bien :

> **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
> **Sinon**, *indice* est un nombre entier tel que $mini \leq indice \leq maxi$ et $T[indice] = v$.

Lorsque l'on sort de la boucle, la condition $mini < maxi$ n'est plus respectée. On a alors $mini \geq maxi$ et comme le tableau est trié dans l'ordre croissant, $T[mini] \leq T[maxi]$.
L'invariant donne :

> **Si** la valeur v n'est pas dans le tableau T, alors *indice* vaut -1.
> **Sinon**, *indice* est un nombre entier tel que $mini \leq indice \leq maxi$ et $T[indice] = v$.

C'est-à-dire que $mini = maxi$ et $T[mini] = v = T[maxi]$ où v n'est pas dans le tableau.
La correction partielle est terminée. L'algorithme retrouve bien la valeur v si elle présente dans le tableau.

e) Complexité

Pour une fois, nous n'allons pas calculer le nombre d'opérations puisque nous avons bien compris maintenant que le but est de trouver la famille de complexité de notre algorithme et non pas le nombre exact d'opérations effectuées.

Si on a un tableau de taille n, à chaque étape, on va éliminer au moins la moitié des cases restantes. On partira de l'approximation que l'on divise la taille du tableau par 2 à chaque étape. D'où l'idée d'encadrer notre nombre n entre deux puissances de 2 :

$$2^{N-1} \leq n < 2^N$$

A la première étape, on divise le tableau en 2, on a alors :

$$\frac{2^{N-1}}{2} = 2^{N-2} \leq \frac{n}{2} < \frac{2^N}{2} = 2^{N-1}$$

A la deuxième étape, on divise le tableau restant en 2, on a alors :

$$\frac{2^{N-1}}{2^2} = 2^{N-3} \leq \frac{n}{2^2} < \frac{2^N}{2^2} = 2^{N-2}$$

...

A la i-ème étape, on divise le tableau restant en 2, on a alors :

$$\frac{2^{N-1}}{2^i} = 2^{N-i-1} \leq \frac{n}{2^i} < \frac{2^N}{2^i} = 2^{N-i}$$

...

A la dernière étape, ... mais au fait quel est la dernière étape ?
A la dernière étape, le reste du tableau ne doit contenir plus qu'une seule case, celle contenant le nombre recherché. Nous sommes donc à la $N-1$-ième étape :

$$\frac{2^{N-1}}{2^{N-1}} = 2^0 = 1 \leq \frac{n}{2^{N-1}} < \frac{2^N}{2^{N-1}} = 2^{N-N+1} = 2^1 = 2$$

Donc il faut au plus $N-1$ itérations. Mais quel est le lien entre $N-1$ et n ?

Définition :
Le **logarithme entier (en base 2)** d'un entier n strictement positif est le nombre des bits de son écriture binaire diminué d'une unité.
On le note

$$\log_2(n)$$

Exemple
Le nombre 666 en binaire s'écrit : 1010011010_2 donc $\log_2(666) = 10 - 1 = 9$

Propriété :
Tout nombre entier positif n peut être encadré par

$$2^{N-1} \leq n < 2^N$$

où

$$N - 1 = \log_2(n)$$

Exemple
Le nombre 666 en binaire s'écrit : 1010011010_2 donc $N-1 = \log_2(666) = 10-1 = 9$ soit $N = 10$. Donc 666 est encadré par $2^{N-1} = 2^9 = 512$ et $2^N = 2^{10} = 1024$. Le nombre 666 est bien entre 512 et 1024.

L'algorithme de recherche dichotomique est un algorithme de **complexité logarithmique**. On le note

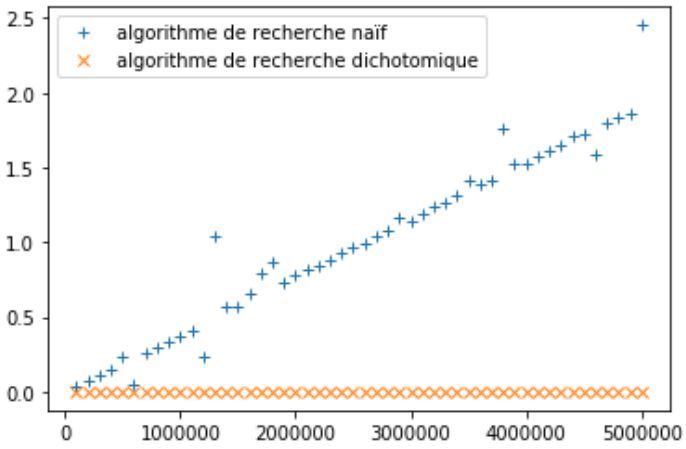
$$O(\log(n))$$

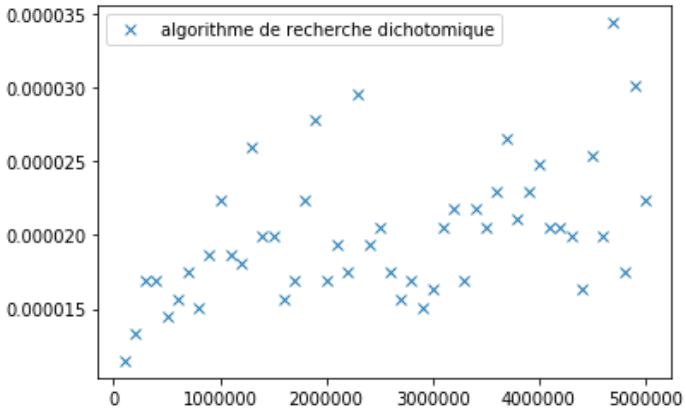
.

Exemple
Comme $\log_2(666) = 10-1 = 9$, si on a un tableau contenant 666 cases rangées dans l'ordre croissant, au pire, on trouvera un nombre dans ce tableau en 9 coups.

III. Un petit bilan ?

Avec un petit programme, on peut comparer l'efficacité de l'algorithme de recherche naïve (on prend toutes les cases les unes après les autres tant que l'on n'a pas trouvé la valeur) à l'algorithme de recherche dichotomique.

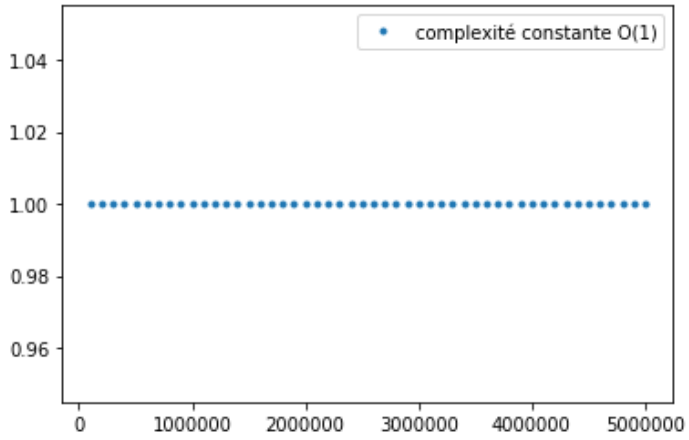




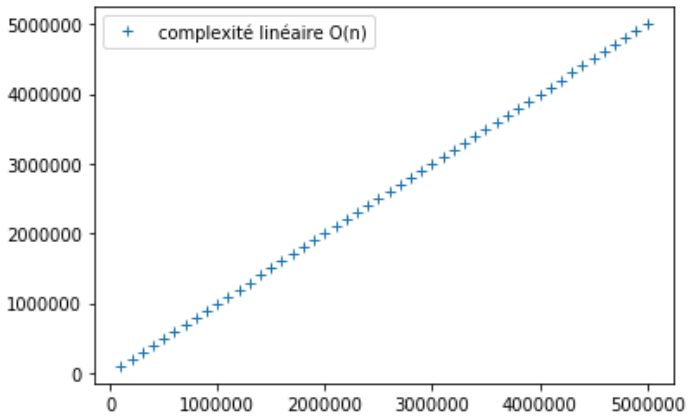
Attention, la complexité logarithmique est difficilement comparable avec la complexité affine. Il ne faut pas croire sur la graphique de gauche que la complexité logarithmique est constante. On peut le constater sur le graphique de droite dans lequel on n'affiche que le temps d'exécution pour l'algorithme de recherche dichotomique. Pour comparer les 2 méthodes, nous avons un problème d'échelle.
A ce propos, dans le graphique de gauche, nous avons quelques irrégularités. Pourquoi ?
.....
.....
.....
dans le graphique de droite, nous avons des résultats très désordonnés. Pourquoi ?
.....
.....
.....

IV. Bilan sur les complexités

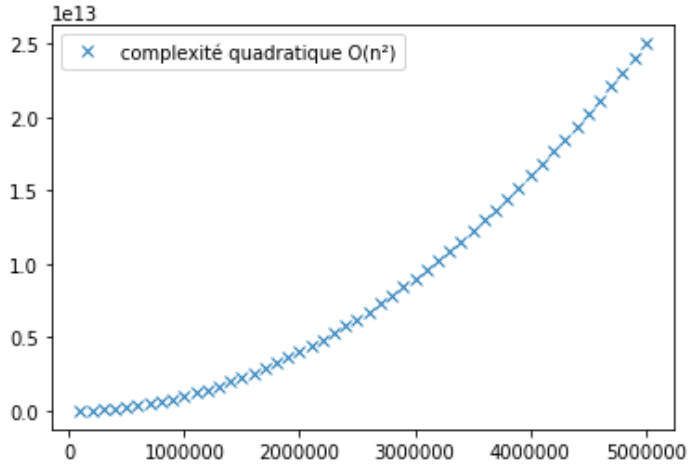
Nous avons vu les complexités suivantes :



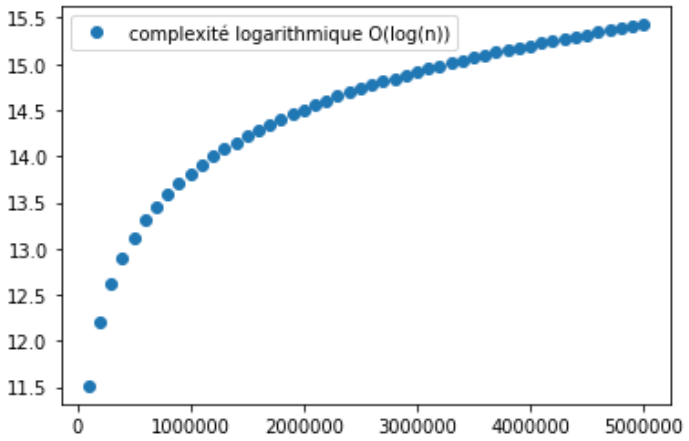
Complexité constante.
Quelque soit la taille du tableau, le nombre d'opérations est le même.



Complexité linéaire.
Le nombre d'opérations est du même ordre que la taille du tableau.
Si le tableau a une taille de 10 cases, il faut des dizaines d'opérations
Si le tableau a une taille de 100 cases, il faut des centaines d'opérations
...

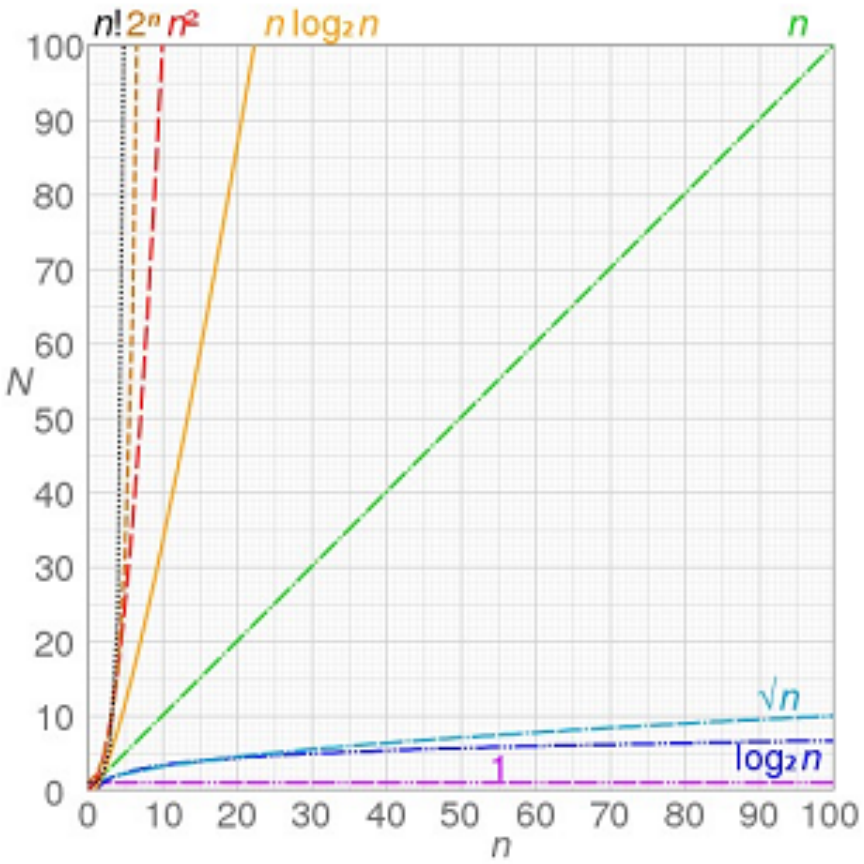


Complexité quadratique.
Le nombre d'opérations est de l'ordre du carré de la taille du tableau.
Si le tableau a une taille de 10 cases, comme $10^2 = 100$, il faut des centaines d'opérations
Si le tableau a une taille de 100 cases, comme $100^2 = 10\,000$, il faut des dix-milliers d' opérations
...



Complexité logarithmique.
Le nombre d'opérations est de l'ordre du nombre de bit nécessaire à l'écriture de la taille du tableau en binaire.
Si le tableau a une taille de 10 cases, comme $10 = 1010_2$, il faut $4 - 1 = 3$ opérations
Si le tableau a une taille de 100 cases, comme $100 = 1100100_2$, il faut $7 - 1 = 6$ opérations
...

Voici un tableau qui permet de comparer quelques-unes des complexités les plus communes en algorithmique.



V. Exercices

Exercice 1 (*)
Faire tourner « à la main » cet algorithme pour `T=[1, 5, 24, 28, 29, 31, 33, 34, 52]` et `v=34`.
Refaire tourner avec `v=35`.

Exercice 2 (*)
Mon petit Larousse comporte exactement 1769 pages de définitions.
Je recherche la définition d'un mot. En combien de coups maximum vais-je la retrouver en utilisant l'algorithme de recherche dichotomique ?

VI. Corrections

`T=[22, 31, 45, 69, 79, 94, 101, 111, 121, 517, 666, 2020, 9999, 10000]`
Comme `len(T)=14`$= 1110_2$, on trouvera un nombre dans ce tableau en un maximum de $\log_2(14) = 4$ coups.

`T=[22, 31, 45, 69, 79, 94, 101, 111, 121, 517, 666, 2020, 9999, 10000]`

`T=[22, 31, 45, 69, 79, 94, 101, 111, 121, 517, 666, 2020, 9999, 10000]`

`T=[22, 31, 45, 69, 79, 94, 101, 111, 121, 517, 666, 2020, 9999, 10000]`

`T=[22, 31, 45, 69, 79, 94, 101, 111, 121, 517, 666, 2020, 9999, 10000]`

Correction de l'exercice 1 (*)

Correction de l'exercice 2 (**)
Comme $1769 = 11011101001_2$, on a $\log_2(1769) = 11 - 1 = 10$. On trouvera la page contenant la définition du mot en maximum 10 coups.