Recherche dichotomique dans une liste triée

On se donne une liste `L` de nombres de longueur `n`, triée dans l'ordre croissant, et un nombre `x0`.

Pour chercher `x0`, on va couper la liste en deux moitiés et chercher dans la moitié de tableau qui encadre `x0` et ainsi de suite...

On appelle `g` l'indice de l'élément du début de la sous-liste dans laquelle on travaille et `d` l'indice de l'élément de fin.

Au début, `g = 0` et `d = n-1`.

On utilise la méthode suivante :

- on compare `x0` à « l'élément du milieu » `L[m]` avec `m = (g + d) // 2;`
- si `x0 = L[m]`, on a trouvé `x0`, on peut alors s'arrêter ;
- si `x0` < `L[m]`, c'est qu'il faut chercher entre `L[g]` et `L[m-1]`;
- si `x0` > `L[m]`, c'est qu'il faut chercher entre `L[m+1]` et `L[d]`.

On poursuit jusqu'à ce qu'on a trouvé `x0` ou lorsque l'on a épuisé la liste `L`.

Question 1 Illustrer la méthode avec les deux exemples suivants :

- `x0 = 5` et `L` =

−3	5	7	10	11	14	17	21	30

- `x0 = 11` et `L` =

−2	1	2	7	8	10	13	16	17

.

Question 2 Si `x0` n'est pas dans la liste `L`, donner un test d'arrêt du processus de dichotomie portant sur `g` et `d`.

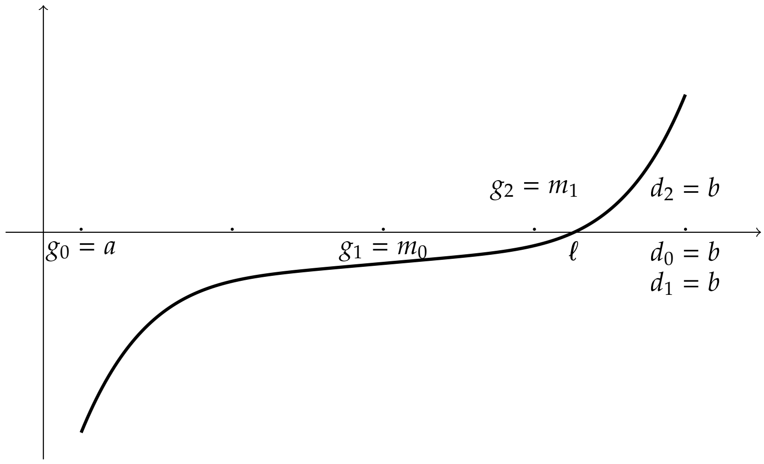
Question 3 Écrire une fonction `dichotomie(x0,L)` qui renvoie `True` ou `False` selon que `x0` figure ou non dans `L` par cette méthode. On utilisera une boucle **`while`** que l'on interrompra soit lorsque l'on a trouvé `x0`, soit lorsque l'on a fini de parcourir la liste.

Question 4 Combien vaut `g-d` au i^{e} tour de boucle ? Si `x0` ne figure pas dans `L`, montrer que le nombre de tours de boucles nécessaires pour sortir de la fonction est de l'ordre de $\ln n$ où `n=len(L)` (cela rend la fonction beaucoup plus efficace qu'une simple recherche séquentielle pour laquelle le nombre de comparaisons pour sortir de la boucle serait de l'ordre de n).

Recherche d'un zéro d'une fonction

Soit une fonction $f : [a,b] \longrightarrow \mathbb{R}$ $(a < b)$ vérifiant : f continue sur $[a,b]$ et $f(a) \cdot f(b) \leqslant 0$ ie $f(a)$ et $f(b)$ de signes opposés.

Le théorème des valeurs intermédiaires s'applique et assure que f possède au moins un zéro ℓ entre a et b.



L'idée consiste à créer une suite d'intervalles $[g_n, d_n]$ tels que pour tout entier naturel n, $g_n \leqslant \ell \leqslant d_n$ et $0 \leqslant g_n - d_n = \dfrac{g_{n-1} - d_{n-1}}{2}$.

On considère $m_0 = \dfrac{g_0 + d_0}{2}$ et on évalue $f(m_0)$:

- Si $f(m_0) \times f(d_0) \geqslant 0$, on va poursuivre la recherche d'un zéro dans l'intervalle $[g_1, d_1] = [g_0, m_0]$.
- Sinon, on poursuit la recherche dans l'intervalle $[g_1, d_1] = [m_0, d_0]$.
- On recommence alors en considérant $m_1 = \dfrac{g_1 + d_1}{2}$ ...

Question 5 Si l'on souhaite que g_n et d_n soient des solutions approchées de ℓ à une précision ε, quelle est la condition d'arrêt de l'algorithme ? Préciser alors la valeur approchée de ℓ qui sera renvoyée par la fonction.

Question 6 Écrire une fonction `recherche_zero(f,a,b,epsilon)` qui renvoie une valeur approchée du zéro de `f` sur `[a,b]` a epsilon près.

Question 7 Tester la fonction avec $f : x \mapsto x^2 - 2$ sur $[0, 2]$ et $\varepsilon = 0,001$.

Question 8 Avec une erreur de $\varepsilon = \dfrac{1}{2^p}$, combien y a-t-il de comparaisons au final en fonction de p ?

Valeur d'un polynôme par plusieurs méthodes

Question 9 Écrire une fonction `exponaif(x,n)` d'arguments un réel x et un entier naturel n, qui renvoie la valeur de x^n par la méthode naïve $x^n = x \times x \times \ldots \times x$ (n termes). Compter le nombre d'opérations dans exponaïf.

Une autre méthode, celle de l'exponentiation rapide consiste à remarquer que $x^n = \begin{cases} (x^2)^{n/2} & \text{si} \quad n \text{ pair} \\ x \times (x^2)^{(n-1)/2} & \text{si} \quad n \text{ impair} \end{cases}$

Le code itératif correspondant est le suivant :

```python
def expo_rapide(x,n) :
    p,res,y = n,1,x
    while p>0 :
        if p%2==1 :
            res=res*y
        p=p//2
        y=y*y
    return(res)
```

Question 10 Quel est le nom de la variable locale dont le contenu est retourné par la fonction ?

Question 11 Faire tourner « à la main » la fonction pour $x = 2$ et $n = 10$ en complétant le tableau suivant puis encadrer le nombre d'opérations dans exporapide en fonction de $\ln(n)/\ln(2)$.

	`p`	`res`	`y`
sortie du 1er tour de boucle			
sortie du 2e tour de boucle			
...			
...			

On considère un polynôme $P(x) = \sum_{k=0}^{n} a_k.x^k$ que l'on modélisera en Python par la liste $P = [a_0, a_1, ..., a_n]$. Dans la suite, on prendra pour tout $k \in \mathbb{N}$, $a_k = k$.

Question 12 Ecrire une fonction `Pnaif(x,P)` d'arguments un réel `x` et `P` la liste des coefficients du polynôme, qui renvoie $P(x)$ à l'aide de la fonction `exponaif`. Compter le nombre d'opérations.

Question 13 Faire de même pour une fonction `Prapide(x,n)` qui renvoie $P(x)$ à l'aide de la fonction `exporapide`. On admettra que la complexité est en O(n ln(n)).

Une dernière méthode consiste à utiliser le schéma de Hörner : $P(x) = ((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a_0$

Question 14 Écrire une fonction `horner(x,L)` de paramètres un réel x et une liste L représentant un polynôme P, renvoie la valeur de $P(x)$ par la méthode de Hörner. Compter le nombre d'opérations.

On désire maintenant visualiser les temps d'exécution des trois fonctions précédentes pour des grandes valeurs de n.

Question 15 Définir la liste N des entiers naturels compris entre 0 et 100.

Question 16 Grâce à la fonction `perf_counter` de la bibliothèque `time`, écrire une fonction `Temps_calcul(x)` qui :

- définit 3 listes `Tn`, `Tr` et `Th` contenant les temps de calcul de $P(x)$ pour $P = \sum_{k=0}^{n} k.x^k$ lorsque n décrit N avec respectivement la méthode naïve, la méthode rapide puis la méthode de Hörner.
- trace les trois courbes `Tn`, `Tr` et `Th` en fonction de N (on prendra $x = 2$). Interpréter le résultat (on pourrait démontrer que les temps d'exécution des trois programmes sont de l'ordre de $n**2$ pour la méthode naïve (on parle de complexité quadratique), de l'ordre de $n\ln(n)$ pour l'exporapide, et de l'ordre de n pour la méthode de Hörner (complexité linéaire)).