Recherche par dichotomie

Lorsque les données sont triées, plusieurs traitements peuvent leur être appliquées. Les opérations de recherche sont essentielles, par exemple pour les moteurs de recherche. Là encore les mathématiques ont permis de mettre au point des algorithmes performants.

Parcourir une liste de données de manière linéaire peut être longue et laborieuse. Sur des données de petite taille cette méthode peut s'avérer être la plus efficace. Elle fonctionne aussi pour sur des structures non ordonnées. La méthode par dichotomie permet d'obtenir la place d'un élément de manière plus rapide sur des listes ordonnées de grande taille.

Le principe de la dichotomie est assez simple. La liste de donnée est divisée en deux parties. La partie contenant l'élément recherché est identifiée. Elle est à son tour divisée en deux parties. Le processus se poursuit jusqu'à obtenir une liste contenant un seul élément. C'est l'élément recherché. Si la liste finalement obtenue est vide, cela signifie que l'élément recherché n'est pas présent.

Un algorithme possible est le suivant.

```
debut_sous_liste ← 0
fin_sous_liste ← longueur de liste_nombres - 1
trouve ← Faux

BOUCLE tant que trouve est égale à Faux et debut_sous_liste <= fin_s
    i_milieu ← (debut_sous_liste + fin_sous_liste) // 2
    SI element recherche est strictment inférieur à élément i_milieu
        fin_sous_liste ← i_milieu - 1
    SINON SI element recherche est strictment supérieur à élément i_
        debut_sous_liste ← i_milieu + 1
    SINON
        trouve ← Vrai
```

1. Coder l'algorithme en Python et le tester sur une liste de 30 nombres aléatoires entre 0 et 100. Cette liste sera générée par compréhension avec `liste_nombres = [random.randint(0, 100) for i in range(30)]` grâce au module `random`. La liste est ensuite triée avec `liste_nombres.sort()`.
2. Montrer que l'algorithme se termine.
3. Montrer que l'algorithme est correct.

La complexité de l'algorithme est logarithmique. La fonction logarithme népérien sera vue en terminale. Elle est notée $\ln()$. Elle est représentée sur le graphique ci-dessous. On remarque que plus x augmente, moins le fonction croît rapidement.

Une des propriétés de la fonction est de transformer les multiplications en additions :

- $\ln(a.b) = \ln(a) + \ln(b)$
- $\ln(a^k) = k.\ln(a)$

En informatique on utilise aussi très souvent la fonction logarithme base 2 notée $\log_2()$. C'est la fonction inverse de la puissance de 2.

- Si $2^a = b$ alors $a = \log_2(b)$

4. Soit n le nombre d'éléments de la liste. Combien d'éléments restera-t-il dans la sous-liste après une division par 2, puis deux divisions par 2, puis trois divisions par 2.
5. Combien d'éléments restera-t-il dans la dernière sous-liste après la dernière division valable, dans le pire des cas ? C'est à dire lorsque l'on cherche un nombre non présent dans la liste.
6. Soit k le nombre de fois où la boucle est exécutée dans le pire des cas. Expliquer pourquoi n et k sont liés par la relation $\frac{n}{2^k} = 1$.
7. En déduire que k est proportionnel à $\ln(n)$.
8. Montrer que la complexité en temps dans le pire des cas de cet algorithme est logarithmique.
9. En utilisant la courbe, expliquer pourquoi l'algorithme est de plus en plus performant avec l'augmentation de la taille des données.
10. Exécuter l'algorithme pour différentes valeurs de n. On prendra `n = 10000`, `n = 20000`, `n = 40000`, ..., jusque `n = 10240000` (valeur de n doublée à chaque fois). Noter les valeurs obtenues pour le temps d'exécution Δt. On travaillera dans le pire des cas, c'est à dire lorsque l'on cherche un nombre non présent dans la liste.
11. Tracer le nuage de points $\Delta t = f(n)$. Quel type de courbe obtient-on ? Est-ce attendu ? Expliquer.