RechercheSequentielleDichotomique

January 27, 2020

0.1 Recherche séquentielle

```
In [3]: def recherche_seq(L, e):
            trouve = False
            for element in L:
                if element == e:
                    trouve = True
            return trouve

        def recherche_seq2(L, e):
            for element in L:
                if element == e:
                    return True
            return False

        def recherche_seq3(L, e):
            i = 0
            n = len(L)
            while i < n and L[i] != e:
                i += 1
            return i == n
```

0.2 Recherche dichotomique

Pour commencer, traiter cet exercice du Concours Castor Informatique : exercice Tas de Graine du Castor Informatique 2017

0.2.1 Exemple 4 : juste prix

Avant chaque tour de boucle, trois cas se présentent :

- Si `proposition == prix` alors le tour de boucle n'est pas exécuté, le prix est trouvé
- Si `proposition > prix` alors le prix se trouve entre `prixInf` et `proposition - 1` donc on met à jour `prixInf` avec la valeur `proposition + 1` puis on affecte à `proposition` le milieu de l'intervalle `[prixInf ; prixSup]` c'est-à-dire `(prixInf + prixSup) // 2`
- Sinon `proposition < prix` le prix se trouve entre `proposition + 1` et `prixSup` donc on met à jour `prixInf` avec la valeur `proposition + 1`puis on affecte à `proposition` le milieu de l'intervalle `[prixInf ; prixSup]` c'est-à-dire `(prixInf + prixSup) // 2`

In [4]:
```python
from random import randint


def juste_prix(prix, prixInf, prixSup):
    proposition = (prixInf + prixSup) // 2
    essai = 1
    while proposition != prix:
        print("prixInf :", prixInf, "prixSup :", prixSup,
        "proposition :", proposition, )
        if proposition > prix:
            print("C'est moins.")
            prixSup = proposition - 1
        else:
            print("C'est plus.")
            prixInf = proposition + 1
        proposition = (prixInf + prixSup) // 2
        essai += 1
    print("Juste prix", prix, "trouvé en", essai, "essais.")
```

0.2.2 Exemple 6 : implémentation de la recherche dichotomique dans une liste triée de nombres

In [5]:
```python
def recherche_dicho(L, e):
    debut = 0
    fin = len(L) - 1
    while debut <= fin:
        milieu = (debut + fin) // 2
        if L[milieu] == e:
            return True
        elif L[milieu] > e:
            fin = milieu - 1
        else:
            debut = milieu + 1
    return False

def recherche_dicho2(L, e):
    x = 0
    n = len(L)
    pas = n // 2
    while pas >= 1:
        while x + pas < n and L[x + pas] <= e:
            x = x + pas
        pas = pas // 2
    return L[x] == e
```

0.3 Comparaison entre recherche séquentielle et recherche dichotomique

0.3.1 Exemple 8 :

On se donne une liste L de taille n dans laquelle on recherche un élément

1. Combien de comparaisons faut-il effectuer avec une recherche séquentielle si l'élément recherché est en dernière position dans la liste ou n'appartient pas à la liste ?

*Réponse : toute la liste doit être parcourue dans ces deux cas, ce qui nous donne n comparaisons (**Complexité temporelle linéaire**)*

2. Combien de comparaisons sont effectués avec une recherche dichotomique si l'élément recherché est trouvé au dernier tour de boucle ou n'appartient pas à la liste ?

*Réponse : Soit a_k le nombre d'éléments de la zone de recherche après k tours. On a $a_0 = n$ et $\forall k \in \mathbb{N}$, $a_{k+1} \leqslant \frac{a_k}{2}$. Par récurrence, on a $a_k \leqslant \frac{n}{2^k}$. Soit m le nombre maximal de tours de boucles effectués, le dernier tour est effectué lorsque $a_m = 1$ donc m est le plus petit entier tel que $a_m \leqslant 1 \iff \frac{n}{2^m} \leqslant 1 \iff \ln(n) \leqslant m\ln(2) \iff \frac{\ln(n)}{\ln(2)} \leqslant m$. Le nombre maximum de tours de boucles est donc le plus petit entier supérieur ou égal à $\frac{\ln(n)}{\ln(2)}$ (**Complexité temporelle logarithmique**)*

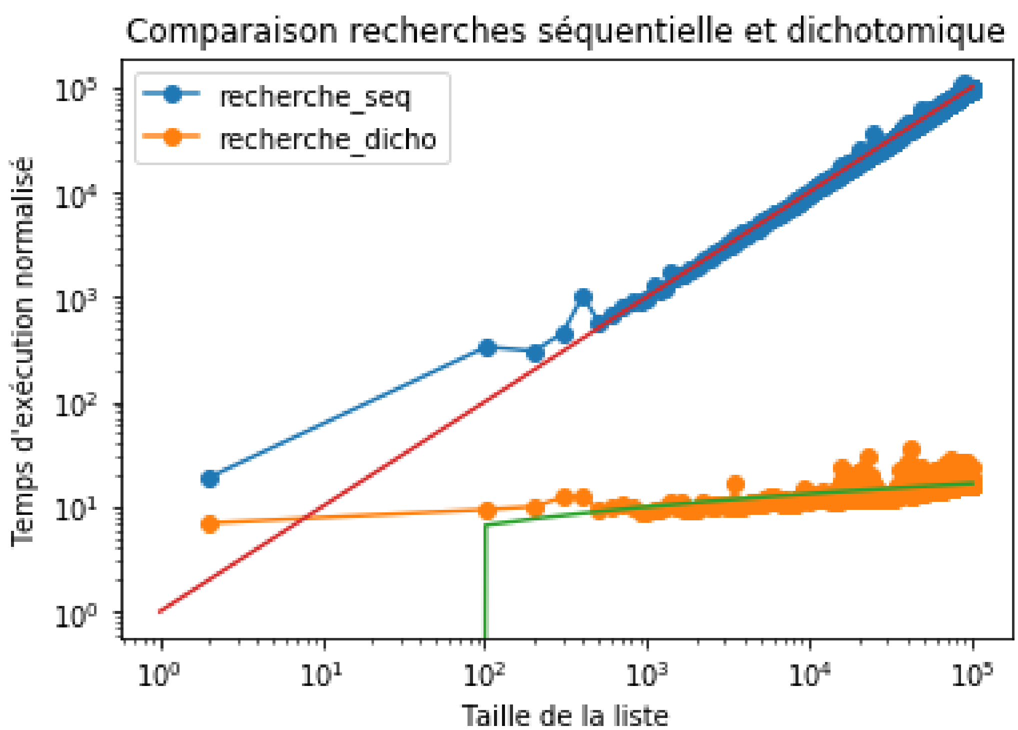
3. Le temps d'exécution d'une fonction de recherche séquentielle peut être modélisé par une fonction affine, tandis que le temps d'exécution d'une fonction de recherche dichotomique présente plutôt l'allure d'une courbe de fonction logarithme.

In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt
import time
from random import randint

def timetest(fonction):
    """exécute la fonction et affiche son temps d'exécution """

    def fonction_modifiee(*args,**kargs):
        debut = time.perf_counter()
        fonction(*args,**kargs)
        return time.perf_counter() - debut

    return fonction_modifiee

def graphique_comparaison_recherches(liste_recherches):
    liste_taille = np.array(list(range(2, 100103, 100)))
    liste_temps = np.array([[0 for _ in range(len(liste_taille))] for _ in range(len(li
    for i, taille in enumerate(liste_taille):
        liste = [randint(0, taille) for _ in range(taille)]
        element = taille + 1
        for j, recherche in enumerate(liste_recherches):
            liste_temps[j][i] = timetest(recherche)(liste, element)
    plt.clf()
```

```
    fonction_modele = [lambda u : u, lambda t : np.log(t)/np.log(2)]
    for k, temps in enumerate(liste_temps):
        constante = np.mean([t/fonction_modele[k](taille) for (t, taille) in zip(temps
        plt.loglog(liste_taille, temps/constante, label=liste_recherches[k].__name__, 
    x = np.arange(1, max(liste_taille) + 1, 100)
    plt.loglog(x, np.log(x)/np.log(2))
    plt.loglog([1, max(liste_taille)], [1, max(liste_taille)])
    plt.title("Comparaison recherches séquentielle et dichotomique")
    plt.legend()
    plt.xlabel("Taille de la liste")
    plt.ylabel("Temps d'exécution normalisé")
    plt.savefig('comparaison-recherches.pdf')
```

```
In [7]: %matplotlib inline
```

```
In [8]: graphique_comparaison_recherches([recherche_seq, recherche_dicho])
```



0.4 Reprogrammation des fonctions bisect_left et bisect_right du module bisect

```
In [9]: def dicho_gauche(L, x):
            debut = 0
            fin = len(L) - 1
            if L[fin] < x:
                return fin + 1
```

```python
    #invariant de boucle L[fin] >= x
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] >= x:
            fin = milieu
        else:
            debut = milieu + 1
    return fin


def dicho_droite(L, x):
    debut = 0
    fin = len(L) - 1
    if L[fin] <= x:
        return fin + 1
    #invariant de boucle L[fin] > x
    while debut < fin:
        milieu = (debut + fin) // 2
        if L[milieu] > x:
            fin = milieu
        else:
            debut = milieu + 1
    return fin

def bisect_right(t, x):
    """
    Invariant conservé par la boucle : t[a] <= x < t[b]
    En sortie de boucle, on a 3 cas :
        invariant conservé : on retourne b
        t[b] <= x : on retourne b + 1
        t[a] > x : on retourne a
    """
    a, b = 0, len(t) - 1
    while b - a > 1:
        m = (a + b) // 2
        if t[m] > x:
            b = m
        else:
            a = m
    if t[b] <= x:
        return b + 1
    elif t[a] > x:
        return a
    else:
        return b

def bisect_left(t, x):
    """
```

```python
        Invariant : t[a] < x <= t[b]
        En sortie de boucle, on a 3 cas :
            invariant conservé : on retourne b
            t[b] < x : on retourne b + 1
            t[a] >=  x : on retourne a
        """
        a, b = 0, len(t) - 1
        while b - a > 1:
            m = (a + b) // 2
            if t[m] >= x:
                b = m
            else:
                a = m
        if t[b] < x:
            return b + 1
        elif t[a] >= x:
            return a
        else:
            return b


    def dicho_comptage(L, x):
        return dicho_droite(L, x) - dicho_gauche(L, x)

    # In [9]: dicho_gauche([1,2,2,2,4,5], 2)
    # Out[9]: 1
    #
    # In [10]: dicho_droite([1,2,2,2,4,5], 2)
    # Out[10]: 4
    #
    # In [11]: dicho_comptage([1,2,2,2,4,5], 2)
    # Out[11]: 3

    t = list(range(10)) * 2
    t.sort()
    print(t)
    print(bisect_left(t, 6.9))
    print(dicho_gauche(t, 6.9))
    print(bisect_left(t, 7))
    print(dicho_gauche(t, 7))
    print(bisect_right(t,7))
    print(dicho_droite(t, 7))
    print(bisect_right(t,7))
    print(dicho_droite(t, 7.1))
```

```
[0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9]
14
14
```

14
14
16
16
16
16