TP13 - Recherche dans une liste - Dichotomie

Exercice 1
Écrire une fonction d'en-tête `def dichotomie(L,x)` qui renvoie True si x appartient à la liste L et False sinon.

In [1]:
```
def dichotomie(L,x):
    d = 0 # indice de début
    f = len(L) -1 # indice de fin
    while f >= d:
        m = (d+f)//2
        if x == L[m]:
            return True
        elif x < L[m]: # on regarde la 1ere moitié
            f = m-1
        else : # on regarde la 2nd moitié
            d = m+1
    return False
```

Tester la fonction.

In [2]:
```
dichotomie([1,2,3,4], 3)
```

Out[2]:
```
True
```

In [3]:
```
dichotomie([1,2,3,4], 5)
```

Out[3]:
```
False
```

Exercice 2 - Temps d'exécution

1. On va mesurer le temps d'exécution de notre recherche dichotomique dans un cas peu favorable. Prenons $L = [1, 2, 3, . . , 10000]$ et $x = 0$.

In [4]:
```
import time

L = range(1,10001)
x = 0

debut = time.perf_counter() # on note l'heure précise
dichotomie(L,x) # on fait tourner la fonction
fin =  time.perf_counter() # on note l'heure précise
duree = fin-debut
print(duree)
```

```
4.305700167606119e-05
```

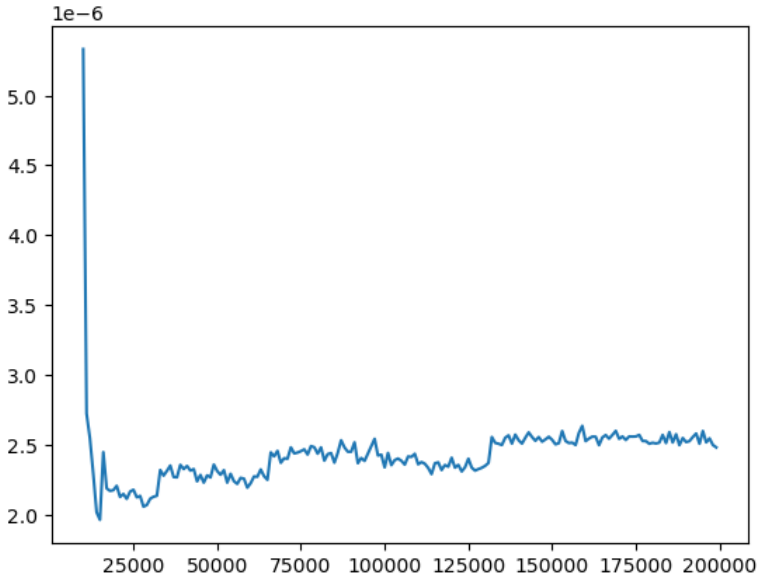
2. On souhaite tracer le graphique du temps d'exécution en fonction de $n \in \mathbb{N}^*$ quand on recherche 0 dans $L = [1, 2, \ldots, n]$. Completer le programme.

In [5]:
```
import matplotlib.pyplot as plt

t = []

for n in range(10000,200000,1000):
    L = range(1,n+1)
    x = 0
    debut = time.perf_counter() # on note l'heure précise
    dichotomie(L,x) # on fait tourner la fonction
    fin =  time.perf_counter() # on note l'heure précise
    duree = fin-debut
    t.append(duree)

N = range(10000,200000,1000) # abscisses n
plt.plot(N,t)
plt.show()
```



Exercice 2 - Comparaison

La mesure de la durée n'étant pas suffisamment précise, nous allons compter le nombre d'itérations réalisées dans la boucle. Nous comparerons la dichotomie à l'algorihtme naïf.

1. Écrire une fonction dichotomie_iterations qui renvoie le nombre d'itérations dans la boucle pour obtenir le résultat.

In [6]:
```
def dichotomie_iterations(L:list, x: float or int) -> int:
    '''renvoie le nombre d'itérations dans la fonction dichotomie'''
    d = 0 # indice de début
    f = len(L) -1 # indice de fin
    n = 0 # nombre d'itérations
    while f >= d:
        n = n+1
        m = (d+f)//2
        if x == L[m]:
            break
        elif x < L[m]: # on regarde la 1ere moitié
            f = m-1
        else : # on regarde la 2nd moitié
            d = m+1
    return n

L=[3,6,7,12,13,20]
dichotomie_iterations(L,15)
```

Out[6]: 3

2. Écrire une fonction naif_iterations qui renvoie le nombre d'itérations réaliser avec une recherche naïve (où on parcourt la liste dans l'ordre des éléments).

In [7]:
```
def naif_iterations(L:list, x: float or int) -> int:
    '''renvoie le nombre d'itérations dans la recherche naïve'''
    n = 0 # nombre d'itérations
    for element in L:
        n = n+1
        if element == x:
            break
    return n

L=[3,6,7,12,13,20]
naif_iterations(L,15)
```

Out[7]: 6

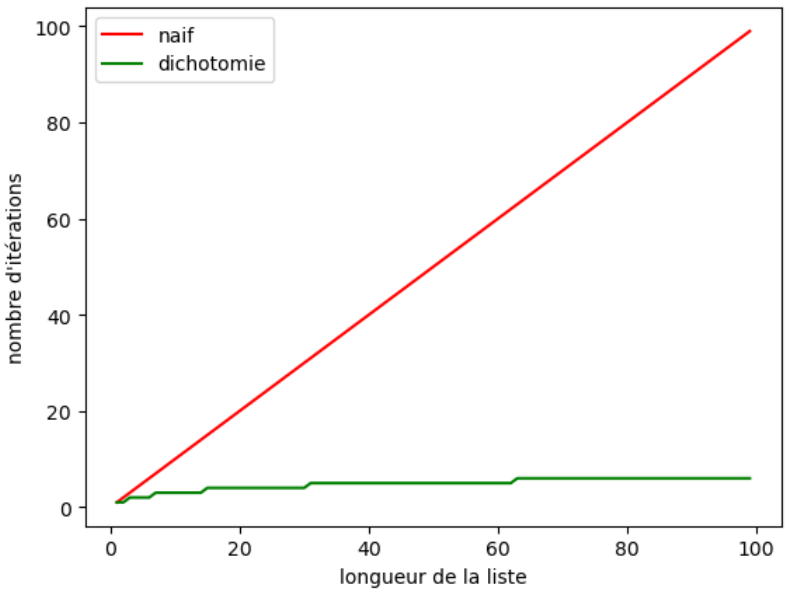
3. On va maintenant tracer un graphique comparatif. On s'aidera des fiches d'aides sur les modules matplotlib et random. Le but est de tracer le nombre d'itérations en fonction de la longueur de la liste n. Pour cela, on va créer des listes L aléatoires croissantes de longueur $n \in [1, 100[$:
   - le premier élément de L sera un entier aléatoire de $[1, 10[$;
   - si x_i est l'élément d'indice i dans L, le suivant x_{i+1} sera un entier aléatoire de $[x_i, x_i + 5[$.

   On cherchera ensuite le nombre $x = 0$ avec les deux méthodes.

In [8]:
```
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rd

X = []
Y_dicho = []
Y_naif = []
for n in range(1,100):
    X.append(n)
    # liste aléatoire croissante de taille n
    L = [rd.randint(1,10)]
    for k in range(n-1):
        L.append(rd.randint(L[-1], L[-1]+5))
    # recherches
    Y_naif.append(naif_iterations(L,0))
    Y_dicho.append(dichotomie_iterations(L,0))

plt.plot(X,Y_naif, color='red', label = 'naif')
plt.plot(X,Y_dicho, color='green', label = 'dichotomie')
plt.xlabel('longueur de la liste')
plt.ylabel("nombre d'itérations")
plt.legend()
plt.show()
```



4. Commenter.

On voit ici que l'algorithme dichotomique est beaucoup plus rapide (donc efficace) que l'algorithme naïf. Cet écart d'efficacité est important quand on manipule de grandes listes.

In [ ]: