EXERCICE 1.1 et 1.2

```python
from random import randint
import time

#DEBUT INIT TAB#

T = int(input("Rentrer taille tableau :"))
# s = T * 5
tab = [randint(0, T) for _ in range(T)]  # ou tab = [randint(0, s) for
_ in range(T)]

#FIN INIT TAB#

#TRI DU TAB#

# print(tab)
tab = sorted(tab)
# print(tab)

#FIN TRI DU TAB#


n = int(input("Entrez un entier :")) #RECHERCHE ENTIER#


#DEBUT DE RECHERCHE LINEAIRE#

i=0

start2 = time.time()

while i < len(tab) and tab[i] != n:

    i = i + 1

if i < len(tab):

    print("\nLa valeur", n, "est à l'indice", i)

else:

    print("\nLa valeur", n, "n'est pas présente dans ce tableau.")


end2 = time.time()

print("Temps de recherche linéaire", end2 - start2)
```

```
#FIN DE RECHERCHCHE LINEAIRE#


#DEBUT DE RECHERCHE DICHOTOMIQUE#

debut = 0
fin = T - 1
trouve = False

start1 = time.time()

while debut <= fin and not(trouve):

    milieu = int((debut+fin)/2)

    if tab[milieu] == n:

        trouve = True

    elif tab[milieu] > n:

        fin = milieu - 1

    else:

        debut = milieu + 1

if trouve:
    print("\nLa valeur", n , "est à l'indice", milieu)
else:
    print("\nLa valeur", n , "n'est pas dans ce tableau")

end1 = time.time()

print("Temps de recherche dichotomique", end1 - start1)

#FIN TRI DICHOTOMIQUE#


#DEBUT DE RECHERCHE INTERPOLATION (ITE)#

debut = 0
fin = T - 1
trouve = False

start3 = time.time()


while debut <= fin and n >= tab[debut] and n <= tab[fin]:
```

```
    probplace = debut + (((fin - debut) * (n - tab[debut])) /
((tab[fin]) - (tab[debut])))
    probplace = int(probplace)

    if debut == fin:

        if tab[debut] == n:
            trouve = True
            break

    if tab[probplace] == n:
        trouve = True
        break

    if  tab[probplace] < n:
        debut = probplace + 1
    else:
        fin  = probplace - 1




end3 = time.time()

if trouve == True:
    print("\nLa valeur", n, "est à l'indice", probplace, "dans le
tableau.")
else:
    print("\nLe nombre", n, "n'est pas dans le tableau.")

print("Temps de recherche par interpolation", end3 - start3)

#FIN DE RECHERCHCHE INTERPOLATION (ITE)#
```

Avec T = T

T = 100

```
La valeur 36 est à l'indice 33
Temps de recherche linéaire 0.0001361370086669922

La valeur 36 est à l'indice 33
Temps de recherche dichotomique 8.082389831542969e-05

La valeur 36 est à l'indice 33 dans le tableau.
Temps de recherche par interpolation 7.891654968261719e-05
```

T = 100000

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 0.013215065002441406

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 9.608268737792969e-05

Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 8.416175842285156e-05
```

T = 10^6

```
La valeur 5 est à l'indice 2
Temps de recherche linéaire 0.0001437664031982422

La valeur 5 est à l'indice 2
Temps de recherche dichotomique 9.608268737792969e-05

La valeur 5 est à l'indice 3 dans le tableau.
Temps de recherche par interpolation 8.20159912109375e-05
```

T = 10^8

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 25.734898805618286

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 0.00011396408081054688

Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 0.00016307830810546875
```

On observe que plus le tableau est grand , plus la recherche dichotomique est rapide et plus la recherche par interpolation est lente. Pour la recherche linéaire, on remarque que plus l'élément est situé en amont du tableau plus la recherche est rapide.

```python
from random import randint
import time

#DEBUT INIT TAB#

T = int(input("Rentrer taille tableau :"))
s = T * 5
tab = [randint(0, s) for _ in range(T)]  # ou tab = [randint(0, T) for _ in range(T)]

#FIN INIT TAB#

#TRI DU TAB#
```

```
# print(tab)
tab = sorted(tab)
# print(tab)

#FIN TRI DU TAB#


n = int(input("Entrez un entier :")) #RECHERCHE ENTIER#


#DEBUT DE RECHERCHE LINEAIRE#

i=0

start2 = time.time()

while i < len(tab) and tab[i] != n:

    i = i + 1

if i < len(tab):

    print("\nLa valeur", n, "est à l'indice", i)

else:

    print("\nLa valeur", n, "n'est pas présente dans ce tableau.")


end2 = time.time()

print("Temps de recherche linéaire", end2 - start2)

#FIN DE RECHERCHCHE LINEAIRE#


#DEBUT DE RECHERCHE DICHOTOMIQUE#

debut = 0
fin = T - 1
trouve = False

start1 = time.time()

while debut <= fin and not(trouve):

    milieu = int((debut+fin)/2)

    if tab[milieu] == n:

        trouve = True
```

```
    elif tab[milieu] > n:

        fin = milieu - 1

    else:

        debut = milieu + 1

if trouve:
    print("\nLa valeur", n , "est à l'indice", milieu)
else:
    print("\nLa valeur", n , "n'est pas dans ce tableau")

end1 = time.time()

print("Temps de recherche dichotomique", end1 - start1)

#FIN TRI DICHOTOMIQUE#


#DEBUT DE RECHERCHE INTERPOLATION (ITE)#

debut = 0
fin = T - 1
trouve = False

start3 = time.time()


while debut <= fin and n >= tab[debut] and n <= tab[fin]:


    probplace = debut + (((fin - debut) * (n - tab[debut])) /
((tab[fin]) - (tab[debut])))
    probplace = int(probplace)

    if debut == fin:

        if tab[debut] == n:
            trouve = True
            break

    if tab[probplace] == n:
        trouve = True
        break

    if  tab[probplace] < n:
        debut = probplace + 1
    else:
        fin  = probplace - 1
```

```
end3 = time.time()

if trouve == True:
    print("\nLa valeur", n, "est à l'indice", probplace, "dans le
tableau.")
else:
    print("\nLe nombre", n, "n'est pas dans le tableau.")

print("Temps de recherche par interpolation", end3 - start3)

#FIN DE RECHERCHCHE INTERPOLATION (ITE)#
```

avec T = T*5 ####

T = 100

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 0.00013709068298339844

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 7.987022399902344e-05

Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 8.082389831542969e-05
```

T = 100000

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 0.014156103134155273

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 9.584426879882812e-05

Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 8.511543273925781e-05
```

T = 10^6

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 0.12198114395141602

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 9.584426879882812e-05
```

```
Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 8.511543273925781e-05
```

T = 10^8

```
La valeur 67 n'est pas présente dans ce tableau.
Temps de recherche linéaire 25.53230381011963

La valeur 67 n'est pas dans ce tableau
Temps de recherche dichotomique 0.0001239776611328125

Le nombre 67 n'est pas dans le tableau.
Temps de recherche par interpolation 0.00014090538024902344
```

EXERCICE 2

2.1 Algorithme de recherche de motif

```
f = open("sequence_adn", "r")
sequence = f.read() #taille 14813
f.close()

#Exemple/Test : affichage d'un extrait de la chaîne
# print(sequence[:100])

f = open("sequence_adn", "r")
sequence = f.read()
f.close()

pattern = input("Votre Pattern ADN ici : ")

def exist_patterns(sequence, pattern):

    # pattern in sequence -> [count, address]
    # return the number of occurrences if there is a occurrence of the
pattern in this string and the coordinates of said pattern, otherwise
a boolean with the coordinates of said occurrences

    # code inspired from an str/str function in C language, mofidied
to retrun a count as well as the corresponding address

    idx_sequence = 0
    count = 0
    i_loc_start  = []
    i_loc_end = []
    address = []
```

```
    if pattern == '':
        return count

    while idx_sequence < len(sequence):
        index_pattern = 0

        while index_pattern < len(pattern) and idx_sequence +
index_pattern < len(sequence) and sequence[idx_sequence +
index_pattern] == pattern[index_pattern]:
            index_pattern += 1
            if index_pattern == len(pattern):
                count += 1
                address.append([idx_sequence, idx_sequence +
len(pattern)- 1])
        idx_sequence += 1

    if count > 0:
        print("Votre pattern '",pattern,"' est présent", count, "fois
commençant à/aux (l')indice(s)", address)
    else:
        print("Votre pattern n'est pas présent")

    return [count,address]

res = exist_patterns(sequence,pattern)
# print(res) vérifie mon return

#OUTPUT

# Votre pattern ' ATTGC ' est présent 2 fois commençant à/aux
(l')indice(s) [[4984, 4989], [5321, 5326]]
```

2.2 Algorithme de compression naïf

```
f = open("sequence_adn", "r")
sequence = f.read()
f.close()

cpt = 1
sequence2 = []

for letter_i in range(len(sequence) - 1):
    if sequence[letter_i] == sequence[letter_i + 1]:
        # on vérifie si  la lettre courante est différente de la
suivante
        cpt += 1
    else:
        if cpt == 1:
            sequence2.append(sequence[letter_i])
            # on réimprime dans une autre séquence le premier lettre
```

```
de la suite trouvé si  elle est composée d'une seule lettre
        else:
            sequence2.append(sequence[letter_i] + str(cpt))
            # sinon on  ajoute le nombre de lettres consécutives à la
suite du dernier élément 'appended' réduisant la taille la séquence
par le nombre imprimé + 1
            cpt = 1
            # réinitialise notre compteur

print("longueur séquence original <= ", len(sequence),
"≠",len(sequence2)," => longueur séquence modifiée")
#en faisant cette comparaison, on peut voir que la modification a bien
été faite et que

# print(sequence2)
# pour afficher la séquence modifiée


# OUTPUT

# longueur séquence original <=  14813 ≠ 10886  => longueur séquence
modifiée
```

2.3 Le nombre de suites de caractères uniques

```
f = open("sequence_adn", "r")
sequence = f.read()
f.close()

cpt = 1
occurences = []
totnb1 = 0
totnb2 = 0
totnb3 = 0
totnb4 = 0
totnb5 = 0
totnb6 = 0
sum = 0

# même principe que le précédant algorithme sauf qu'on implémente un
compteur pour chaque valeur trouvée du premier compteur.
# On compte ainsi le nombre d'occurences nécessaires à la réalisation
d'une moyenne

for letter_i in range(len(sequence) - 1):
    if sequence[letter_i] == sequence[letter_i + 1]:
        cpt += 1
    else:
        if cpt == 1:
            occurences.append(cpt)
```

```
            totnb1 += 1
            # ici  on peut compter les occurrences succéssives d'une
lettre égal à 1
            sum += cpt
        else:
            occurences.append(cpt)
            if cpt == 2:
                totnb2 += 1

            if cpt == 3:
                totnb3 += 1

            if cpt == 4:
                totnb4 += 1

            if cpt == 5:
                totnb5 += 1

            if cpt == 6:
                totnb5 += 1

            sum += cpt
            cpt = 1

# print(occurences) pour vérifier la bon fonctionnement de cet algo
print("nombre d'occurences => ", len(occurences), "| somme des
occurrences => ", sum, "\n")

moyenne = sum / len(occurences)

print("La moyenne est de : ", moyenne)

print("Nombre d'occurence d'une lettre égal à 1 : ",totnb1)

# OUTPUT

# nombre d'occurences =>  10886 | somme des occurrences =>  14811

# La moyenne est de :  1.3605548410802866
# Nombre d'occurence d'une lettre égal à 1 :  7964
```

2.4 Nouvel algorithme de compression

```


# I decided to write the commments and variable name in English in
accordance with international standards.
# Realised after the Python course at HETIC with Corto Dufour

def find_freq_sec(sequence, lg_pattern):
```

```
    # lg_pattern in séquence ->  rec_pattern
    # returns the frequent elements of patterns that are present in
sequence.

    rec_pattern = set()
    #initialise a list in the form of a dictionary

    # will begin to count in an index over the sequence minus the
length of the pattern +1 in order not to go beyond the boundaries of
squence when finding a pettern
    for index_sequence in range(len(sequence) - lg_pattern + 1):
        pattern = sequence[index_sequence : index_sequence +
lg_pattern]
        if  pattern not in rec_pattern:
            rec_pattern.add(pattern)
            # each time we encounter a new pattern, we add it to our
set if it is not already present

    # print(rec_pattern)
    # to check  if everything works well
    return rec_pattern



def replace_freq_in_sec(sequence, lg_pattern, alias_letters):

    # alias_letters and lg_pattern in sequence -> new_sequence
    # return a new string of caracters with alias_letters instead of
rec_pattern
    rec_pattern = find_freq_sec(sequence, lg_pattern)

    # create a dictionary to link each element of rec_pattern to an
alias letter
    alias_generator = (letter for letter in alias_letters)
    alias_mapping = {}

    # look for a pattern in the list of rec_pattern, assign a letter
at the adresse of the alias map matching a pattern
    for pattern in rec_pattern:
        alias_mapping[pattern] = next(alias_generator)

    new_sequence = sequence

    # replace the pattern by a new_letter in a new sequence, the
letter was previously assigned to a pattern.
    for pattern, new_letter in alias_mapping.items():
        new_sequence = new_sequence.replace(pattern, new_letter)

    return new_sequence
```

```
f = open("sequence_adn", "r")
sequence = f.read()
f.close()

lg_pattern = 3
# max length of patterns with my set of alias_letters  / ... may need
a more advanced function to set a comparaison key ?...
# if lg_pattern = 4 it crashes !!!

alias_letters =
"1234567890ZERYUIOPQSDFHJKLMWXVBNazertyuiopqsdfghjklmwxcvbnæê®†Úºîœπô€
ⓣë¶«¡Çø–‡Ò∂ƒfiÌÏÈ¬µÙ@≈©◊ß~∞…÷≠≤„”[å»ÁØ]™ªŒ∏Ô¥#‰±•¿ı∫√¢⁄›"
# do not contain any of the original letter for clarity 'A'T'G'C'
# alias_letters as 'key'

compressed_sequence = replace_freq_in_sec(sequence, lg_pattern,
alias_letters)


# print("\nSéquence compressée : ", compressed_sequence)
print("\nTaille nouvelle sequence compressée : ",
len(compressed_sequence))

# In a way, this function is like a compression algorithm but not
really because we don't lose any information. It ressembles a cipher
where it is necessary to have the alias letter in order to decipher
the new sequence.


# OUTPUT

# {'ATG', 'CCG', 'TTT', 'ACA', 'CAC', 'CGC', 'GAC', 'TCC', 'TCG',
'TAT', 'GCT', 'AGG', 'CAA', 'TCT', 'TGA', 'TGT', 'CGA', 'GAA', 'GGT',
'CAT', 'CTT', 'GCC', 'CGG', 'GTT', 'TAC', 'AGC', 'ACC', 'TTG', 'TGG',
'GCA', 'AAA', 'TGC', 'AGA', 'ATA', 'CGT', 'AAG', 'CTA', 'CCT', 'ATT',
'CTC', 'GTA', 'TTC', 'ACT', 'TCA', 'GTC', 'CCA', 'TAG', 'CTG', 'GGG',
'GTG', 'GGA', 'ACG', 'GGC', 'AAC', 'ATC', 'GAT', 'GCG', 'AAT', 'GAG',
'TAA', 'TTA', 'CCC', 'AGT', 'CAG'}

# # Séquence compressée :
117oLL1TAHRC52555SC51f1CCK7GZ4G55CA2CG9GHCR6CLAZAC3Z2CLA4fFC5CTVZGCLLL
A4GSCLA3GCLAFCCLAG5CT5CT41ZAOGLL8CCERCK02CLLE6AZG4f2CC5CLL5CETQGCLAG5C
LA4GCLE6LLA4ZC02ST6AYC3AG8CAkHHÚ1GS55æK75GGPAT2GGRG2C8A8gTL5Y2b78KTAOr
T79k59KTA6TA22CAkF4JZC5fGRTAZ5C0TA22f1RGN5CLA4C444GCLA4GVGRTSCAZ2APHG1
1T4CRyG7N1TA85GFºZºZ2J5lT12Fa9EAPAKG228TG5FAZ152GA1GTZGGEcEA8A11CLEC5V
GT6AG1X55CLA4YwECAZ1CCtG1CC17T6TEA15F51S5GFTFC5GG4CCXGRDGFA4AT2SCLA4F4
AG8LLA4GCLAG25ºR2CA4T2CAPCR8fGHC4CCLAG55CAPCT2I6CRCQl1n6CLAG6jlGUTC82Q
AZC5T88TC5CT2CC9A1GL48TFAG5GRTRÚ6RATS8LLA8C5CT4AGX1RCTRGEA1T8BFTRG5CTY
```

KCT8LA2BCKCTFAkPT6CL157Kº1A1G4ZG71TG2CQGGEEHjCL7G9GGLQC2XGCJjGGRHA8C2l
9GECG832TVPC8CUTA1GO4kHHHHPGTFCA8RHC1wGXGYGGVGHAT27UVZVFTC8LA4CC1CLAG5
CAYGCLLLA4GCLLLLELLLAYGCELLLAYVGRhS5C1CG555CTFsFTC84C5CLS5C4G6V8C5CNL
LLA4GCLAGRC1º9T6C8T5CC9GN8LLLA4VGTRC85C4STC5F5GC1CQT1KGEA8T2CCIGC5W92N
LLA4CC82CT2CLLLAGRC8A4GCLLA4GCLA44ZwG442GX2G6yCAd2CCR4CC5TG7D2Cf44V4YC
4T6CC55ºRh2CLA4AM2A4GIFAk689Ac8LLLLA4VGA1CTVJGC8TGFa6a9WCG2AC5GGOk3C8C
8AGIgTYCaEr4TRT18A8AuAG65TV0AC2CS2GG2GT2lT9A7GG4CC9GHCQRGHC4HC4CXFT4AC
7GEGuRsGSKGZ2QAtO2RG51TT6t8TKfPA6AG32JQºKXGFCRt2CC2As75CYG2ºHHCGFK1AmE
CG7w5GUb226G4GT29EGC4TC7G8AZAOG2CC5CK8T6CECGFTG6GZA8ºOPCK8AZCEAFYQlT02
XGLAGXG8W5FRTC27TG5æXA2dEI2T2PBG6r2K8rT8A7CCXAZGGaEAG3C3s1e13CRGRALJGG
PWFAPA3VZGA2LG4QRv1GT8AP9G61Aa4APPi4G6CECRTCLa4444AT9LAGRCC4AC5G6EC7CC
LAG5C4GHC55CUACDAFUAC1Q5C12CRG15ºU1m1Ú4A4qGOGL1M9TAD5RC9K1jb71SC1GG5K2
GGRHQCA2MXA6CCLAG155T4FAP6PT5CEGRC1†K263C5CtO2C1†KT4mFEGG17CLLLAG5CEYA
i2C2G4FCC8AFEXGHQC1†CE6A7RCQCAJjI8CCLLA4GCLLLA4FCT6APCCLLA4GCE6LE†C4GR
CNLLAZG2CEU6A4QG4S2CLLA44GCL29YA6CAZGRJGRGH2TTQG7TGQ1JC2EGHAT6CT2CTQlT
R1T2GT551AA6fG1A6TFCS51GG4T2CE6AJjC0m7JlJC2AeG4GE8AsPCSC4AC1w1KCLLAG9l
J22kQFC5iEGC115CEFEG884G15CEG48TGFA45F8T8Rx2Aq6AG89j12YC5IA8CX1OAGsG3Q
rIRPGRA6lEº4T4C2GKCA7YaZaGAEAG35TTRT6TAD729GOEA9aT6C5FCIFT5lG6CaTBGA3Q
8AP9G61rTGPGAEa4A7GAEGOG2lT2GEOEC7GaTA4C5f4GT2CAoG1AZC3YC1Q02C15C4TC18
aG8C18XV1ZTG7CRAG322GX21AC0S5T41G4C091A2CAYmQS1ÚQG29GCQQGCEIXGYG9GETQG
GLEXG1qQS2S8152b71GsGT701G6AT5C4AT2C4GC58ST64CC5T5TA255SLA4kR0CX4C5CES
22GCQT4GT8Au1f1C5T2CyC4T25T6A846CS25CaF9GC0Kd55FC2AuCA02CELQG5T2A85PHz
HA2GHRAe8BTAOV1GCQQGGHErX6AGN9GEX6AkQl9GR1GA0TAHCGQACQjQT1l6PCGPGIGGVG
5C58CAE4C9258º11GAYFCDCÚVZC5GPf2AO4H6X7NEAuEyBAtPB6zPPZGKh2G44ACUaZEG1
OWHIG1AT1lT2BAC7v1sGEEA9ZC1CXA9c8A7GAZjC8Am4AC777C9GG7sG4b1EAYG1GHA7GA
21GOAHAsYMCCX1AAPGA81XHGAJGRAHGYKº2l1GLRAA6R6A2y45GCSCa8TGPhPAEAJC5T4A
2CKCU6G9GH6A9aTTFSACJÚYC9æHCaTRGAYQ8a4GG61rTGPGAE44AZFCA4CRPeG6AAP7Ql6
RHRCf42oGOAPM5RGHG2CC8BALaGRG6LLLLÚ6LLLa6La6LA7T2k1A1AkS9TT24G3GGao9GT
7dL98CXGRT2†0z2GH5CU77CAZCT02lYCLAG5iAGRTA4441VGHRTA4z44444444GRyRTA4z
4z4ATR1QC4OAYGC1GG5DCA4V4AT2AYC1ACRYsZ4SRVG7TAOY7GFCIAG1ZC4Ed4GXÚBsZGQ
wG3TIXEAECC3FapA4HRBAW9xXC4TYGGL9F1UTA3C9nYGRG0R1G1TGUGRAuC9GGE0T1AA4G
PGSAC75GTZA9GEA8AL5GIGF4G2r9TgRAA36Ú4AA6GAk8gSA3ZXPAT9lTa®DCEAPCG3GKêO
ê0ACRGIG8TRTB2TG0UPADGZG6BELAGRAG5jAT2GGL8L4T22GRCTRsG6BAkYdLA9GXRCGUV
1XRGHA9AGE5GTUZCT8A2AYT1º4GC58YTxY9PLQGG6AAOq2jEoGU4GGEO3AOPZ9ZJBKsG5Z
VG7º75XAFTFAGLAG6CDQETRT5CRC5CTEx2A8L5CCfZQG5QxSCE5T4VGSCA4VGAZGG2CC2T
50T2GAYTQMC5CYDAGOAAKfYCA9GC8R06AT2®He1Sº7D5TTZ5T5Ud5CRCG4TRAAFTGFAo9G
T5CaPqF2WRHE4T43BBAPDG1GTSCÚ21BHZ2GUEUBC8A1GX6TTRA0EAA4hE1ATRGC9ZSqBAA
PAJGR6AA9A4TC35PC7Cag4GYUBaG3PO45TA8AG13WC2Om3GZGPaWQTFAGERPpE3QYRA29A
GRA1Ge6GEAPZGOAMAGOa4CRAT81ZC4Ed4GXÚBsZGQwG3TIXEAECC3FapA4HRBAW9xXC4TY
GGL9F1UTA3C9nYGRG0R1G1TGUGRAuC9GGE0T1AA4GPGSAC75GTZA9GEA8AL5GIGF4G2r9T
gRAA36Ú4AA6GAk8gSA3ZXPAT9lTa®DCEAPCG3GKêOê0ACRGIG8TRTB2TG0UPADGZG6BELA
GRAG5jAT2GGL8L4T22GRCTRsG6BAkYdLA9GXRCGUV1XRGHA9AGE5GTUZCT8A2AYT1º4GC5
8YTxY9PLQGG6AAOq2jEoGU4GGEO3AOPZ9ZJBKsG5ZVG7º75XAFTFAGLAG6CDQETRT5CRC5
CTEx2A8L5CCfZQG5QxSCE5T4VGSCA4VGAZGG2CC2T50T2GAYTQMC5CYDAGOAAKfYCA9GC8
R06AT2®He1Sº7D5TTZ5T5Ud5CRCG4TRAAFTGFAo9GT5CaPqF2WRHE4T43BBAPDG1GTSCÚ2
1BHZ2GUEUBC8A1GX6TTRA0EAA4hE1ATRGC9ZSqBAAPAJGR6AA9A4TC35PC7Cag4GYUBaG3
PO45TA8AG13WC2Om3GZGPaWQTFAGERPpE3QYRA29AGRA1Ge6GEAPZGOAMAGOa4CRAT81Ca
4d4GHAGUAsZTQGGIC3GTRHGHC5TG27TC52VPGA9TaZAZC4GHw2G2NG43C2nYGCETA8xQIG
UGTBATYjCE0TAO7XC8As7CCEGR9GEGAoLRæXC78APGTIrRAzFº0BHæTFCT8c3ZXPAT2AG4

```
v002EjYGKROTA4KfGIGTSRê4æTYT6AMTGZAGOAALA4G1GG6A7w1T1ATBtE1TUAHVGA2PZj
GC46yQX0C2G7pAGKCEGRCC5GRG1ZFLA4lEEjaGA4CRC8AsGTYAAHAP79G1AAZº1C7pAZPb
PZGRAPATa4TC3KCRGLaTUGZCXAPaG3GaG7CSTA8AG1TG3GC2GPA7TLLRnaYMUAGRPT4Q1G
3YAA9AEGCR1GAPGAPGAPqGEAMaPG4GFAGRC1M1L93TUTBC8Aq49AFRGT8C83CGEAJ6C25Q
w2wQj7wG2XGHGQC2ZAC2CGFCCaG6C25h2F2x25GHC2fFCC2DAt6G2HGHOFCAZM2jFIM262
22yG8t6h2HGFAF2w72LAG82FC2G6G5†L2F2CAFCC2TFCDj5TG6TC2FC55C2ZGT67jGGVG6
CC2CGN2CTZZwG7CEGC2GHGC2AEVEAG2GXIMCCX1AAPGc5GUb2TRC8Ú0AAHEOMTr6TC9AA2
CC0AC2VGRGYTGPCXaEAuC53AA2CKCT72GHCG2CA9aT6CC46YZTG9æ6CETRGAYGG3CAP2Ga
1rTGPPA7C4Aq8CA4CRG1OY8A4CTF9FyG2GC2CCXPL4AAi4FC4†f8C5yC8CTQZ84CCS8C88
A0AA170nE1Ú83TTaGKC53oA2GG7fG6CT4G6CCRyC38CCyZCAZ2G2W4ZA6EGGaG2CT1GXEZ
FTAZº68A4BC9GQA8TG8LAG22s8CXGQG83CfF2C8C9GQA6CfG2yGRCCfZ1GFTYº00YkQº9G
AEAG1PZA3TS29AZAHAFCAZEGGQN24Z1GLGQlG9ZCEGCSA8T22CAFCCSmPCUVGA2RCC3GSC
AFK7C8YhE4En6DGC9sPCfG5Y5827DC7X1AETRGAPM5CTr4GOEI2AZGG7XG6TC2GHY2C4AA
D55G6CAZ7lZGGaEAA3CA3CA4AAKCUG2N2GEGXa92C5CCXAZCA1B6EUaYQ8AP2G61ALAPPA
6æOGEjGCE1CCNE††ETTFREr2CXAG1CCY7CÚ2eG5C8fPFT6fGT8AT5CYU1Z1Ú83TUTRCTRq
1TC7CCRGT8CT2NGEAG0TC4æOTKCKCCL75T62QACKG2bFAGEOæLGZ8AG2GECGF3jCG2wG2T
G5IVG6TA6GFTG2A7C2GFC2C5CT22†22†2f2C2GZG8CC9w8ÚVFA4G2GGFC8TC2GL2GVG6LG
ECQCLLAFCC22CC8h6fP†yV8CL285T2eGRAGL29YMCCX16BJ5GUG5QAAMCRTKG2GHlFRG6T
C97GFT4CT6CLE8XAJGGaEAG35T4A2CKCU6G22GEGXaT6C5G6YFI2AG6CaTRGAYQ8AP9G61
rTGPBA7C4AJFCA4CRv2C9QGV2HG6AG2GE††X2FRHE†CyG6YAG1Ú8TA3CGUAi5CT82T5YGF
AGHGCEAG8T8XFAZG2CYAC9YGZ1CG72hH5KC26D1GG2G72FE4EGC3GHCG880KC5º2GHCGZA
H2GGH6yGUTT1GG2G6CTFTYA2GGaEA6GFTG6YCT2b2AOG2†2p5C2AG26rT2qG64GG4AGLGI
9GaGA2ELAPGTZC5GC2YAC2X1º61n9TV4GT8TC3XFºH9bE2f7CT45ZMA7CCXAZGGaEAA350
n6KCT722GH6ATav6G5FCIFU2AGHCaTRGA3GG3CAP2G61rTGPAAEaGRqZºNGYCAZCAkHEEE
EBAFAG2AGU1AJSTG0T1O1C3A3YBTA8AG2RGT9G3T68EAFT82A4AqYTG6C3Z8AMCfG62GZ1
GLET2GG683CT2CC9GNEGYGKT2GHjGF1GHFaG6GZwGY1H2ZAHGC9AA2Ú8TTRC15TG6FC3GL
APCCY2jIGT2w7G2fE2GX6CAELAGE797WGH2aoAD2GAYMCCX1OAZCEFU2AALGE2f7CT46ZM
A7CCXAZlPAP35T4AT6KCT79GHE6A9aT6GC46YFs2BG4vRGAYGG3CAP2G61rTGPAAEa4AAM
TCEA2GPf2FE4EHAECGEEEEAaGU18A3CXGA6TTLL0T1TC79ATSrT5æLGEA6F8LRG3y9GFAo
6PC2Hº2w5G2AG5h7T8CC9TVZ8AFRLG2GI9G65CXb2CUGC25F9jAFTG2YGYAC5†KCTVFCCE
G6G2GH2AALEKCT25hZaFLG26G2E2CC2jCEFEH2UEHAG2CTZj6G2jGAZZR4j5GC2Ú4G3GPq
9GHGCEæZG8TGYnRaZGZAHGG4ATBA3GOEALGE4AaGE2QRACR†CYGCRhZG5GC2CYTC2PBG6T
G8CT4CRAKCa6XAZaGAEAG3CITRT1KCT5CEGA2CEC1rTGFA78TRYæUæ4ÚRAAYGG3CAP2GH1
B1AAPG1AzELEGRaoA1c1L9TA3G17YAq49AT2B3CC81OAP0T6BzZ48CU4CÚ2P0TAH27CEP9
lT8AGS55CEAZGKC5CA2y26CtZMCUGeº0AZG5CAd8EGFXRTT6GAF5j2FCEHGC555CCUPAT6
AG2CYGOG2G6CYiE5CYF828fF2fFTVFAFAFy7CA8C8LG2fGRGRCCSAGYMDGG1BBu5JAG5GG
UMCRT0AAHElFRG69E4FT42GCLE8TGPTTaPaG3ST4A2CKCT72AEaEA9aT6C5T6IFYCUGECa
TRAAYQ8BA29T1AA1PPG7CA2AC8f4CRAkEsG6CC2G2G6TG6fG5yC9GCL5CC2G5TYP758CaG
65G22GLAGRHVGE4T5CET0AA

# Taille nouvelle séquence compressée :  6673
```