```
from math import *



def affiche_ip(n):
    n1=n%1000 ; n=n//1000 ; n2=n%1000 ; n=n//1000
    n3=n%1000 ; n4=n//1000
    ip = "{0:}.{1:03}.{2:03}.{3:03}".format(n4, n3, n2, n1)
    return ip


#Q1

fichier=open('unsorted_IP_adresses_list.txt','r')
liste=fichier.readlines()
liste_triee=[]
for element in liste:
    liste_triee.append(int(element))
print(liste_triee[200]+liste_triee[760])
liste_triee.sort()
print(liste_triee[200]+liste_triee[760])
#Q2
print(affiche_ip(liste_triee[234567]))

#Q3

def rech_dicho(x,L):
    g = 0
    d = len(L)-1
    if x < L[g] or L[d] < x:  # pour la robustesse
        return None
    if L[d] == x:    # on elimine ce cas particulier pour s'assurer de verifier l invariant
        return d     # quand on rentre dans la boucle while
    while d-g>1 :  # on sort de la boucle while quand il ne reste que 2 termes.
        m = (g+d)//2
        if L[m] > x:  # on garantit l invariant de boucle :
            d = m       #    L[g] <= x < L[d]
        else:          # donc L[m]<=x
            g = m
    if L[g] == x: # si on arrive la, g=d-1, donc la sous-liste etudiee ne comporte que deux termes
                                                                    ,
    # dont le second est forcement > x. Il faut tester L[g]
        return g
    else:
        return None

#print(rech_dicho(78192004194,liste_triee))
#print(rech_dicho(127198224147,liste_triee))

#Q4

def rech_seq(x,L):
    n=len(L)
    for i in range(n):
        if L[i]==x:
            return i
    return None ##si l element n est pas dans la liste

#Q5

# #temps sequentiel
# from time import *
# t1=perf_counter()
# rech_seq(78192004194,liste_triee)
# t2=perf_counter()
# print(t1,t2)
# tseq=t2-t1
# print(tseq)
#
```

```
# #temps dichotomique
# t3=perf_counter()
# rech_dicho(78192004194,liste_triee)
# t4=perf_counter()
# print(t3,t4)
# tdicho=(t4-t3)
# print(tdicho)
# n=len(liste_triee)
#
# print(n/log(n) , tseq/tdicho)


#Q6

def recherche_somme(liste,somme):
    n=len(liste)
    for k in range(n-1):#on ne teste pas le dernier terme
        complement=somme-liste[k]
        for j in range(k,n):
            if liste[n-(j-k)-1]==complement:#on demarre du dernier terme de la liste
                return k,n-(j-k)-1
    return None

liste_test1 = [1,3,5,7,9,10,13,19,27,50,96,110,115] #pour tester 120 ou deux cas possibles

print(recherche_somme(liste_test1,120))

#Q9
def recherche_somme_dicho(liste,somme):
    n=len(liste)
    for k in range(n-1):#on ne teste pas le dernier terme
        complement=somme-liste[k]
        rech=rech_dicho(complement,liste[k+1:n])
        if rech != None:
            return k,(rech+k+1)
    return None


liste_triee2=liste_triee[0:1000]

def somme_dicho(liste,somme):
    n=len(liste)
#     somme2=somme//2 #on garde un nombre entier
#     i=0
#     while i<=n-1 and liste[i]<somme2:
#         i+=1
#     if i==n:#cela veut dire qu on est sorti de la boucle sans trouver un nombre inferieur a N/2
#         return None
    g,d = 0,n-1
    print(g,d)
    while d-g>1: #au moins un ecart entre deux termes
        test=liste[g]+liste[d]
        if test==somme:
            return g,d
        if test<somme:
            g+=1#on va grandir la somme
        else:#test>somme, donc on prend un nombre plus petit
            d-=1
    return None

valeurtest=84130176#84130175
#temps sequentiel
from time import *
# t1=perf_counter()
# a=recherche_somme(liste_triee2,valeurtest)
# t2=perf_counter()
# print(a)
# tseq=t2-t1
# print(tseq)

#temps semi-dichotomique
```

```
# t3=perf_counter()
# b=recherche_somme_dicho(liste_triee2,valeurtest)
# t4=perf_counter()
# print(b)
# tsemidicho=(t4-t3)
# print(tsemidicho)

#temps dichotomique
t5=perf_counter()
c=somme_dicho(liste_triee2,valeurtest)
t6=perf_counter()
print(c)
tdicho=(t6-t5)
print(tdicho)



def rech_indice(x,L):
    #permet de renvoyer l'indice de l element immediatement superieur a x
    g = 0
    d = len(L)-1
    if x < L[g] or L[d] < x:  # pour la robustesse
        return None
    if L[d] == x:    # on elimine ce cas particulier pour s'assurer de verifier l'invariant
        return d     # quand on rentre dans la boucle while
    while d-g>1 :  # on sort de la boucle while quand il ne reste que 2 termes.
        m = (g+d)//2
        if L[m] > x:  # on garantit l invariant de boucle :
            d = m       #    L[g] <= x < L[d]
        else:          # donc L[m]<=x
            g = m
    return d

def somme_dicho2(liste,somme):
    n=len(liste)
#     somme2=somme//2 #on garde un nombre entier
#     i=0
#     while i<=n-1 and liste[i]<somme2:
#         i+=1
#     if i==n:#cela veut dire qu on est sorti de la boucle sans trouver un nombre inferieur a N/2
#         return None
    g,d = 0,rech_indice(somme,liste)
    print(g,d)
    while d-g>1: #au moins un ecart entre deux termes
        test=liste[g]+liste[d]
        if test==somme:
            return g,d
        if test<somme:
            g+=1#on va grandir la somme
        else:#test>somme, donc on prend un nombre plus petit
            d-=1
    return None

#temps dichotomique
t7=perf_counter()
d=somme_dicho2(liste_triee2,valeurtest)
t8=perf_counter()
print(d)
tdicho=(t8-t7)
print(tdicho)
```