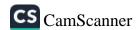
```
#(α) ερωτημα
      import time
      from time import perf counter
      def modpow(a,b,c):
        result=1
        while(b>0):
        if(b%2==1):
         result=result*a
         a=a*a
                                                                     ΑΣΚ. 6
         a=a\%c
         result%=c
 11
         b=b//2
12
13
        return result
      #με αυτο τον αλγοριθμο σε καθε επανάληψη ο μεγαλυτερος αριθμος είναι το πολύ α^2 σε μεγεθος
 15
      #αρα δεν χρειαζεται να υπολογιστουν οι τεραστιες δυνάμεις a^b για το τεστ ,αλλα μονο μεχρι α^2
      #απο space complexity ειναι διαχειρησιμο
17
      n=int(input("input number:"))
      ch=True
      for a in range (2,n-1):
 21
       if(modpow(a,n-1,n)!=1):
         ch=False
23
        break
25
      if(ch=True): print("is prime")
      else: print("not prime")
                                 TERMINAL
input number:1001219
is prime
PS C:\Users\vagga\ python -u "c:\Users\vagga\fermat.py"
input number:1001220
not prime
PS C:\Users\vagga>
```



```
C: > Users > vagga > 🕏 fermat.py > ...
      import random
      def modpow(a,b,c):
        result=1
        while(b>0):
         if(b%2==1):
          result=result*a
         a=a^*a
         a=a\%c
                                                                       Ασк. 6
         result%=c
         b=b//2
 11
 12
        return result
      #με αυτο τον αλγοριθμο σε καθε επανάληψη ο μεγαλυτερος αριθμος είναι το πολύ α^2 σε μεγεθος
      #αρα δεν χρειαζεται να υπολογιστουν οι τεραστιες δυνάμεις a^b για το τεστ ,αλλα μονο μεχρι α^2, αρα ειναι διαχειρισημο
 15
      n=int(input("input number:"))
      ch=True
      n=pow(2,2281)-1 # <-\pi po\sigma \omega pivn αλλαγή, για να δοκιμασω κ'άυτο, απλα δεν μπορω να το παρω απο keyboard
      for a in range (1,30):
       q=random.randint(2,n-1)
       if(modpow(q,n-1,n)!=1):
 21
         ch=False
         break
      if(ch=True): print("propably is prime")
      else: print("not prime")
                                 TERMINAL
PS C:\Users\vagga\fermat.py"
input number:67280421310721
propably is prime
                                                                                         δουλεύει για μεγάλους πολύ
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
                                                                                         αριθμούς
input number: 170141183460469231731687303715884105721
not prime
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
input number:00
propably is prime δώ είναι input=2^2281-1, απλά δεν το έβαλα από keyboard (βλ. κώδικα)
```



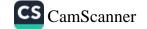
```
input number:443372888629441
not prime
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
input number:561
not prime
PS C:\Users\vagga> Oι δυο παραπανω αριθμοι ειναι carmichael numbers, και βλεπουμε πωσ το προγραμμα λειτουργει ορθα (δοκιμαζομενο και με αλλουσ carmichael αριθμους)
```

PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"



```
13
        return result
      #με αυτο τον αλγοριθμο σε καθε επανάληψη ο μεγαλυτερος αριθμος είναι το πολύ α^2 σε μεγεθος
14
      #αρα δεν χρειαζεται να υπολογιστουν οι τεραστιες δυνάμεις a^b για το τεστ ,αλλα μονο μεχρι α^2, αρα ειναι διαχειρισημο
15
      ch=True
      #το modpow το οριζω οπως και στο ερωτημα 1, απλα δε χωραει στο screenshot
17
      for b in range (2,201):
       for a in range (1,30):
        q=random.randint(1,b-1)
 20
                                                                                   Ασκ. 6, ευρεσή
        if(modpow(q,b-1,b)!=1):
21
         ch=False
22
         break
                                                                                   mersenne prime
       if(ch==True):
        for a in range(1,35):
25
         q=random.randint(1,pow(2,b)-2)
         if(modpow(q,pow(2,b)-2,pow(2,b)-1)!=1):
          ch=False
 28
          break
 29
        if(ch=True): print("2^",b,"-1 is marsenne prime")
       ch=True
PROBLEMS
                                  TERMINAL
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
2^ 2 -1 is marsenne prime
2^ 3 -1 is marsenne prime
2^ 5 -1 is marsenne prime
2^ 7 -1 is marsenne prime
2^ 13 -1 is marsenne prime
2^ 17 -1 is marsenne prime
2^ 19 -1 is marsenne prime
2^ 31 -1 is marsenne prime
2^ 61 -1 is marsenne prime
2^ 89 -1 is marsenne prime
2^ 107 -1 is marsenne prime
2^ 127 -1 is marsenne prime
PS C:\Users\vagga>
```

 \vee def modpow(a,b,c):



```
import time
     from time import perf counter ns
     def rec fib(n):
                                         recursive
       if(n=1 or n=0): return n
       return rec fib(n-1)+rec fib(n-2)
     def rep fib(n):
       prev1=1
                                                             Ασκ 7 (α)
       prev2=0
       res 0
11
       if(n=1): return 1
12
                                   for i in range (0,n-1);
           (variable) prev2: int
13
                                   res=prev1+prev2
14
         prev2=prev1
15
         prev1=res
                                    τι λέει πισώ απτο κουτακί ((variable prev2:int),
16
       return res
                                   αφησα το ποντικί σε λάθος σημείο
17
18
     def mat fib(n):
       a b c ar br cr dr 1
19
       d=0
20
21
       if(n<4):
22
        return n-int(n>1)
23
24
       n=n-3
25
       mat=[[a,b],[c,d]]
26
       result=[[ar,br],[cr,dr]]
27
28
       while(n>0):
        if(n%2==1):
29
30
          result[0][0]=a*ar+br*c
31
          result[0][1]=ar*b+br*d
32
          result[1][0]=cr*a+dr*c
33
          result[1][1]=cr*b+d*dr
34
          ar=result[0][0]
35
          br=result[0][1]
36
          cr=result[1][0]
37
          dr=result[1][1]
```



```
def mat fib(n):
  a b c ar br cr dr 1
  d-0
  if(n<4):
    return n-int(n>1)
                                          Ασκ. 7 (α)
  n=n-3
  mat=[[a,b],[c,d]]
  result=[[ar,br],[cr,dr]]
  while (n>0):
  if(n%2==1):
     result[0][0]=a*ar+br*c
     result[0][1]=ar*b+br*d
     result[1][0]=cr*a+dr*c
     result[1][1]=cr*b+d*dr
     ar=result[0][0]
     br=result[0][1]
     cr=result[1][0]
     dr=result[1][1]
   n=n//2
   mat[0][0]=a*a+b*c
   mat[0][1]=a*b+b*d
   mat[1][0]=a*c+d*c
   mat[1][1]=c*b+d*d
   a mat[0][0]
   b-mat[0][1]
   c=mat[1][0]
   d=mat[1][1]
  return result[0][0]+result[0][1]
n=int(input())
st=perf counter ns()
print(mat fib(n))
```

18

19

20

21

22 23

24

25

26 27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47 48

49

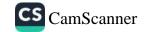
50 51

52

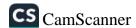
```
Lernin Leantrialialtaltaeantrialial
      n=int(input())
      st=perf counter ns()
      print(mat fib(n))
      t1-perf counter ns()
      print("time:",t1-st)
                                                                                    Ασκ. 7 (α)
      print(rep fib(n))
      t2=perf counter ns()
      print("time:",t2-t1)
      print(rec fib(n))
      t3=perf counter ns()
      print("time:",t3-t2)
 63
                                TERMINAL
KeyboardInterrupt
```

```
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
40
102334155
time: 358300
102334155
time: 591100
102334155
time: 22719425100
PS C:\Users\vagga> Παρατηρουμε (δοκιμαζοντας πολλους αριθμουσ) οτι σε καθε περιπτωσ
```

PS C:\Users\vagga> Παρατηρουμε (δοκιμαζοντας πολλους αριθμουσ) οτι σε καθε περιπτωση time(rec_fib)>time(rep_fib)>time(mat_fib) και για αριθμους μεγαλ -υτερους του 35 ειναι time(rec)>>time(rep)>time(mat), και για αρκετα μεγαλους αριθμους το rec_fib δεν τερματιζει, αρχικα λόγου χρόνου μετά λογω max depth της αναδρομης, απο ενα σημειο και μετα δε τερματιζει κανένας αλγοριθμος γιατι digits(result)>4.300, αλλα απο θεμα χρόνου οι αποδοτικοί αλγοριθμοι τερματιζουν σε καθε περιπτωση, πριν απο αυτο το σημειο



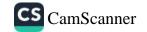
```
def gold(n): #(B)
49
        phi=(1+5**0.5)/2
       result=1
        while(n>0):
         if(n%2==1): result=result*phi
                                                                                                       Ασκ. 7 (β)
        n=n//2
        phi phi phi
       result/=(5**0.5)
       return int(result + int(result-int(result)>0.5))
     n=int(input())
     st=perf counter ns()
     print(mat fib(n))
     t1=perf counter ns()
     print("time:",t1-st)
     print(rep fib(n))
     t2=perf counter ns()
     print("time:",t2-t1)
     print(gold(n))
70
     t3=perf counter ns()
71
     print("time:",t3-t2)
72
                                 TERMINAL
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
100
354224848179261915075
time: 620500
354224848179261915075
time: 573300
354224848179261800448
time: 415800
PS C:\Users\vagga> Δοκιμάζοντας διάφορες τιμές η, παρατηρούμε οτί για "μικρες" τιμες η είναι time(rep fib)>time(gold)>time(mat fib),όμως καθώς αυξάνο
νται οι τιμες του η σε αρκετά "μεγάλο" επίπεδο η gold(n) γίνεται πιο αποδοτική και απο τις δυο (πχ n=100,οπως φαινεται παραπάνω), και καταλήγουμε σε
σημείο όπου time(gold)<time(mat)<time(rep)
```



```
#(y)
17
     #χρησιμοποιώ το mat fib αντι του gold,γιατι στο gold (νοομίζω) είχε rounding error σε πράξεις float%k,έτσι ειχαμε λάθος αποτελεσμα
     def mat fib(n,k):
       k=pow(10,k)
       a b c ar br cr dr-1
      d=0
      if(n<4):
                                                                                   Ασκ. 7 (γ)
        return (n-int(n>1))%k
      n=n-3
      mat=[[a,b],[c,d]]
      result=[[ar,br],[cr,dr]]
       while(n>0):
       if(n%2==1):
         result[0][0]=(a*ar+br*c)%k
         result[0][1]=(ar*b+br*d)%k
         result[1][0]=(cr*a+dr*c)%k
         result[1][1]=(cr*b+d*dr)%k
         ar=result[0][0]
         br=result[0][1]
         cr=result[1][0]
         dr=result[1][1]
        n=n//2
        mat[0][0]=(a*a+b*c)%k
        mat[0][1]=(a*b+b*d)%k
        mat[1][0]=(a*c+d*c)%k
        mat[1][1]=(c*b+d*d)%k
        a=mat[0][0]
        b-mat[0][1]
        c=mat[1][0]
        d=mat[1][1]
       return (result[0][0]+result[0][1])%k
```

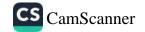


```
def mat fib(n,k):
        return (result[0][0]+result[0][1])%k
      n=13100
      t1=st=perf counter()
      while(t1-st<=1):
       print(n) #για να βλεπω το ρυθμο του προγραμματος (αν ειναι κοντα στο 1sec,αν τελειωσει σημερα),θα μποροσυα να βαζω τιμες απο keyboard
                #εντελη ειναι αργος τροπος, αλλα επειδη ενα τεστ ειναι να γινει τ΄άφησα ,ας γινεται και πιο γρηγορα
 57
       q=pow(10,n)
       st=perf counter()
       mat fib(q,17)
       t1=perf counter()
                                                                                           Ασκ. 7 (γ)
      q=pow(10,n-1)
 64
      print(mat fib(q,17), "biggest i is:",n-1)
      st=perf counter()
      mat fib(q,17)
                     #για να μην μετρησει το print, μικρο το κακο but still
      t1=perf counter()
      print("time:",t1-st)
 70
                                 TERMINAL
13284
13285
13286
83788299560546875 biggest i is: 13286
time: 0.9630888999672607
83788299560546875 biggest i is: 13286
time: 0.9630888999672607
PS C:\Users\vagga> σημείωση *ειναι περιπου 13286, καθε φορα που το ετρεχα αλλαζε στο περιπου ο χρονος, αλλα ειναι σε αυτη την περιοχη τιμων του 13286
 (το μεγαλυτερο i που ζητάει)
```



```
import math
     from time import perf counter ns
     def fast double(n,k):
       k=pow(10,k)
       b=0
       c=1
       f=[b,c]
       a=[]
11
                                                                                           Ασκ. 7 (δ)
       length=len(bin(n))
12
       while(n>0):
13
        a.insert(0,n%2)
14
15
       for i in a:
         c=(f[1]*f[1]+f[0]*f[0])%k
17
         b=(f[0]*(2*f[1]-f[0]))%k
         if(i==1):
           f[0]=c
           f[1]=(b+c)%k
         else:
22
           f 0 b
           f[1]=c
25
       return f[0]
                                 TERMINAL
ime 1: 734600
ime 2: 979200
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
Θεωρητικα:το fast double κανει πράξεις χρονου Ο(1) σε loop μήκους len(n,binary) αρα θέλει χρόνο Ο(logn) οπου n η εισοδος (n-οστος αριθμος fib),ομοια
to mat fib θέλει χρόνο O(logn) γι'αύτο βλέπουμε και παρομοιους χρόνους εκτέλεσης που αυξομειώνονται ανάλογα την περίπτωση, αλλα γενικα η πολυπλοκοτητ
```

α των δυο αλνοριθμων ειναι η ιδια

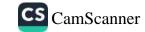


```
61
 62
      n=int(input())
      k=int(input())
 63
      n=pow(10,n)
 64
      st = perf counter ns()
 65
      print(fast double(n,k))
      t1= perf counter ns()
 67
                                                 Ασκ. 7 (δ)
      print(mat fib(n,k))
 68
      t2=perf counter ns()
 69
      print("time 1:",t1-st)
 70
      print("time 2:",t2-t1)
 71
PROBLEMS
                   DEBUG CONSOLE
                                  TERMINAL
15000
4
6875
6875
time 1: 1761559900
time 2: 1100868000
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
19000
5
46875
46875
time 1: 2837245800
time 2: 1730727700
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
```

Οπως εξηγήσαμε θεωρητικα προηγουμενως φαινεται στα παραπανω παραδειγματα



```
template <typename T>
27
      T nlogn_major(T a[],int n){
28
      if(n=0) return -1;
29
      if(n=1) return a[\emptyset];
30
31
      T t1=nlogn major(a,n/2);
32
      T t2=nlogn major(a+n/2, n-n/2); //2T(n/2)+0(1)
33
34
      if(t1=t2) return t1;
35
      int n1=0, n2=0;
37
                                                        Аок. 8
      for(int i=0;i<n;i++){
39
                                                        O(nlogn)
      n1 + (a[i] - t1);
40
      n2 + (a[i] - t2);
41
      ]//o(n)
42
                                                        αλγόριθμος
      if (n1 > n / 2) return t1;
43
         if (n2 > n / 2) return t2;
44
45
      return -1;
      }// Αρα συνολικά εχουμε O(n) logn φορεσ, αρα πολυπλοκοτητα O(nlogn)
47
      int main(){
48
      char jk[]={'a','b','c','d','c','b','c','b','b','l','b','a','b','a','b','a','b','b','b'};
49
50
PROBLEMS
                  DEBUG CONSOLE
                                TERMINAL
      output of n_major(jk) και nlogn_major(jk) b, b majority
```



```
#include <iostream>
#include "map"
using namespace std;
template <typename T>
T n_major(T a[],int n){
   map<T, int> q;
   for(int i=0;i<n;i++){
                                                   Ασκ. 8
     q[a[i]]++;
                                                   O(n) αλγοριθμος
  T res;
  int mx=0;
    for(auto i: q){
     if(i.second>mx){
        res=i.first;
        mx=i.second;
      if(mx > n/2) {
       return res;}
      return -1;
}//we go through each element once through 2 indepents loops so time complexity: O(2n)=O(n)
```

