ΑΣΚ. 6

```python
#(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
   a=a%c
   result%=c
   b=b//2

  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
#απο space complexity ειναι διαχειρησιμο

n=int(input("input number:"))
ch=True
for a in range (2,n-1):
 if(modpow(a,n-1,n)!=1):
   ch=False
   break

if(ch==True): print("is prime")
else: print("not prime")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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 > ...

Ασκ. 6

```python
#(α) ερωτημα
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
     result%=c
     b=b//2

    return result
#με αυτο τον αλγοριθμο σε καθε επανάληψη ο μεγαλυτερος αριθμος ειναι 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
n=int(input("input number:"))
ch=True
n=pow(2,2281)-1 # <-προσωρινη 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):
     ch=False
     break

if(ch==True): print("propably is prime")
else: print("not prime")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
input number:67280421310721
propably is prime
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
input number:170141183460469231731687303715884105721
not prime
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
input number:00
propably is prime
```

εδώ είναι input=2^2281-1, απλά δεν το έβαλα από keyboard (βλ. κώδικα)

δουλεύει για μεγάλους πολύ αριθμούς

Ασκ. 6

```
PS C:\Users\vagga> python -u "c:\Users\vagga\fermat.py"
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> Οι δυο παραπανω αριθμοι ειναι carmichael numbers, και βλεπουμε πωσ το προγραμμα λειτουργει ορθα (δοκιμαζομενο και με αλλουσ carmic
hael αριθμους)
```

Ασκ. 6, ευρεσή mersenne prime

```python
def modpow(a,b,c):

  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
ch=True
#το modpow το οριζω οπως και στο ερωτημα 1, απλα δε χωραει στο screenshot
for b in range (2,201):
 for a in range (1,30):
  q=random.randint(1,b-1)
  if(modpow(q,b-1,b)!=1):
   ch=False
   break
 if(ch==True):
  for a in range(1,35):
   q=random.randint(1,pow(2,b)-2)
   if(modpow(q,pow(2,b)-2,pow(2,b)-1)!=1):
    ch=False
    break
  if(ch==True): print("2^",b,"-1 is marsenne prime")
 ch=True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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>
```

```python
import time
from time import perf_counter_ns
def rec_fib(n):
    if(n==1 or n==0): return n
    return rec_fib(n-1)+rec_fib(n-2)

def rep_fib(n):
    prev1=1
    prev2=0
    res=0
    if(n==1): return 1
    fo
        (variable) prev2: int
        prev2=prev1
        prev1=res
    return res

def mat_fib(n):
    a=b=c=ar=br=cr=dr=1
    d=0
    if(n<4):
        return n-int(n>1)

    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]
```

recursive

repetion

Ασκ 7 (α)

for i in range (0,n-1):
res=prev1+prev2

τι λέει πισώ απτο κουτακί ((variable prev2:int),
αφησα το ποντικί σε λάθος σημείο

matrix

(continues next pic)

Ασκ. 7 (α)

```python
def mat_fib(n):
    a=b=c=ar=br=cr=dr=1
    d=0
    if(n<4):
        return n-int(n>1)

    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))
```

```python
    return result[0][0]+result[0][1]

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(rec_fib(n))
t3=perf_counter_ns()
print("time:",t3-t2)
```

Ασκ. 7 (α)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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 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 αλγοριθμ
οι τερματιζουν σε καθε περιπτωση, πριν απο αυτο το σημειο
```

Ασκ. 7 (β)

σχέση με το φ (golden ratio)

```python
def gold(n):    #(B)
    phi=(1+5**0.5)/2
    result=1
    while(n>0):
        if(n%2==1): result=result*phi
        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))
t3=perf_counter_ns()
print("time:",t3-t2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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 n, παρατηρούμε οτί για "μικρες" τιμες n είναι time(rep_fib)>time(gold)>time(mat_fib),όμως καθώς αυξάνο
νται οι τιμες του n 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)
```

Ασκ. 7 (γ)

```python
#(γ)
#χρησιμοποιώ το mat_fib αντι του gold,γιατι στο gold (νοομίζω) είχε rounding error σε πράξεις float%k,έτσι ειχαμε λάθος αποτελεσμα
#ο τροποποιημένος mat_fib ωστε να δινε mod(10^k) αποτελεσμα:
def mat_fib(n,k):
    k=pow(10,k)
    a=b=c=ar=br=cr=dr=1
    d=0
    if(n<4):
        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
```

Ασκ. 7 (γ)

```python
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
    n+=1     #εντελη ειναι αργος τροπος, αλλα επειδη ενα τεστ ειναι να γινει τ΄άφησα ,ας γινεται και πιο γρηγορα
             #τελικα ειδα οτι για 13500 ηταν τ>1 sec, αρα τάφησα απο 13100 να τρέχει
    q=pow(10,n)
    st=perf_counter()
    mat_fib(q,17)
    t1=perf_counter()


q=pow(10,n-1)
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)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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)
```

```python
import time
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=[]
    length=len(bin(n))
    while(n>0):
     a.insert(0,n%2)
     n//=2
    for i in a:
      c=(f[1]*f[1]+f[0]*f[0])%k
      b=(f[0]*(2*f[1]-f[0]))%k
      if(i==1):
        f[0]=c
        f[1]=(b+c)%k
      else:
        f[0]=b
        f[1]=c

    return f[0]
#(y)
```

Ασκ. 7 (δ)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
time 1: 734600
time 2: 979200
PS C:\Users\vagga> python -u "c:\Users\vagga\fibonacci.py"
Θεωρητικα:το fast double κανει πράξεις χρονου O(1) σε loop μήκους len(n,binary) αρα θέλει χρόνο O(logn) οπου n η εισοδος (n-οστος αριθμος fib),ομοια
το 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 αλγοριθμων ειναι η ιδια
```

Ασκ. 7 (δ)

```python
n=int(input())
k=int(input())
n=pow(10,n)
st = perf_counter_ns()
print(fast_double(n,k))
t1= perf_counter_ns()
print(mat_fib(n,k))
t2=perf_counter_ns()
print("time 1:",t1-st)
print("time 2:",t2-t1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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>
T nlogn_major(T a[],int n){
if(n==0) return -1;
if(n==1) return a[0];

T t1=nlogn_major(a,n/2);
T t2=nlogn_major(a+n/2,n-n/2);  //2T(n/2)+O(1)

if(t1==t2) return t1;
int n1=0,n2=0;


for(int i=0;i<n;i++){
n1+=(a[i]==t1);
n2+=(a[i]==t2);
  }//O(n)
if (n1 > n / 2) return t1;
    if (n2 > n / 2) return t2;
return -1;
}// Αρα συνολικά εχουμε O(n)  logn φορεσ, αρα πολυπλοκοτητα O(nlogn)

int main(){
char jk[]={'a','b','c','d','c','b','c','b','b','l','b','a','b','a','b','a','b','b','b'};
```

Ασκ. 8
O(nlogn)
αλγόριθμος

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
bb
PS C:\Users\vagga>
```

output of n_major(jk) και nlogn_major(jk)

b , b majority element=b

```cpp
#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++){
      q[a[i]]++;
    }
  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)
```

Ασκ. 8

O(n) αλγοριθμος