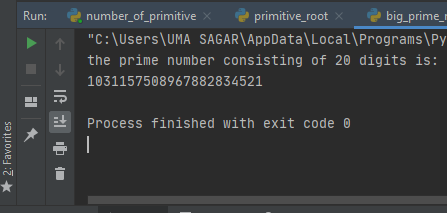
**Program1**

**Code for Finding a large prime number (min 20-digit)**

from random import randrange, getrandbits  
def is\_prime(n, k=128):  
 *""" Test if a number is prime  
 Args:  
 n -- int -- the number to test  
 k -- int -- the number of tests to do  
 return True if n is prime  
 """* # Test if n is not even.  
 # But care, 2 is prime !  
 if n == 2 or n == 3:  
 return True  
 if n <= 1 or n % 2 == 0:  
 return False  
 # find r and s  
 s = 0  
 r = n - 1  
 while r & 1 == 0:  
 s += 1  
 r //= 2  
 # do k tests  
 for \_ in range(k):  
 a = randrange(2, n - 1)  
 x = pow(a, r, n)  
 if x != 1 and x != n - 1:  
 j = 1  
 while j < s and x != n - 1:  
 x = pow(x, 2, n)  
 if x == 1:  
 return False  
 j += 1  
 if x != n - 1:  
 return False  
 return True  
def generate\_prime\_candidate(length):  
 *""" Generate an odd integer randomly  
 Args:  
 length -- int -- the length of the number to generate, in bits  
 return a integer  
 """* # generate random bits  
 p = getrandbits(length)  
 # apply a mask to set MSB and LSB to 1  
 p |= (1 << length - 1) | 1  
 return p  
def generate\_prime\_number(length=70):  
 *""" Generate a prime  
 Args:  
 length -- int -- length of the prime to generate, in bits  
 return a prime  
 """* p = 4  
 # keep generating while the primality test fail  
 while not is\_prime(p, 128):  
 p = generate\_prime\_candidate(length)  
 return p  
print(generate\_prime\_number())

output:

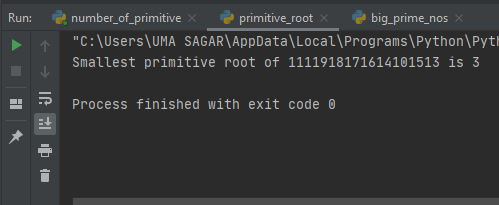


**Program2:**

**Code for finding Primitive root (min 10 digits**

# Python3 program to find primitive root  
# of a given number n  
from math import sqrt  
  
  
# Returns True if n is prime  
def isPrime(n):  
 # Corner cases  
 if (n <= 1):  
 return False  
 if (n <= 3):  
 return True  
  
 # This is checked so that we can skip  
 # middle five numbers in below loop  
 if (n % 2 == 0 or n % 3 == 0):  
 return False  
 i = 5  
 while (i \* i <= n):  
 if (n % i == 0 or n % (i + 2) == 0):  
 return False  
 i = i + 6  
  
 return True  
  
  
""" Iterative Function to calculate (x^n)%p  
 in O(logy) \*/"""  
  
  
def power(x, y, p):  
 res = 1 # Initialize result  
  
 x = x % p # Update x if it is more  
 # than or equal to p  
  
 while (y > 0):  
  
 # If y is odd, multiply x with result  
 if (y & 1):  
 res = (res \* x) % p  
  
 # y must be even now  
 y = y >> 1 # y = y/2  
 x = (x \* x) % p  
  
 return res  
  
  
# Utility function to store prime  
# factors of a number  
def findPrimefactors(s, n):  
 # Print the number of 2s that divide n  
 while (n % 2 == 0):  
 s.add(2)  
 n = n // 2  
  
 # n must be odd at this po. So we can  
 # skip one element (Note i = i +2)  
 for i in range(3, int(sqrt(n)), 2):  
  
 # While i divides n, print i and divide n  
 while (n % i == 0):  
 s.add(i)  
 n = n // i  
  
 # This condition is to handle the case  
 # when n is a prime number greater than 2  
 if (n > 2):  
 s.add(n)  
  
 # Function to find smallest primitive  
  
  
# root of n  
def findPrimitive(n):  
 s = set()  
  
 # Check if n is prime or not  
 if (isPrime(n) == False):  
 return -1  
  
 # Find value of Euler Totient function  
 # of n. Since n is a prime number, the  
 # value of Euler Totient function is n-1  
 # as there are n-1 relatively prime numbers.  
 phi = n - 1  
  
 # Find prime factors of phi and store in a set  
 findPrimefactors(s, phi)  
  
 # Check for every number from 2 to phi  
 for r in range(2, phi + 1):  
  
 # Iterate through all prime factors of phi.  
 # and check if we found a power with value 1  
 flag = False  
 for it in s:  
  
 # Check if r^((phi)/primefactors)  
 # mod n is 1 or not  
 if (power(r, phi // it, n) == 1):  
 flag = True  
 '''break'''  
  
 # If there was no power with value 1.  
 if (flag == False):  
 return r  
  
 # If no primitive root found  
 return -1  
  
  
# Driver Code  
n = 1111918171614101513  
print("Smallest primitive root of",  
 n, "is", findPrimitive(n))

Output:



**Program3:**

**Code for DH (Taking random private keys (Min 3 of different sizes with min 8 digits) (Paste code and output screenshot)**

/\* This program calculates the Key for two persons

using the Diffie-Hellman Key exchange algorithm \*/

#include<stdio.h>

#include<math.h>

// Power function to return value of a ^ b mod P

long long int power(long long int a, long long int b,long long int P)

{

if (b == 1)

return a;

else

return (((long long int)pow(a, b)) % P);

}

//Driver program

int main()

{

long long int P, G, x, a, y, b, ka, kb;

// Both the persons will be agreed upon the

// public keys G and P

P = 23; // A prime number P is taken

printf("The value of P : %lld\n", P);

G = 9; // A primitve root for P, G is taken

printf("The value of G : %lld\n\n", G);

// Alice will choose the private key a

a = 4; // a is the chosen private key

printf("The private key a for Alice : %lld\n", a);

x = power(G, a, P); // gets the generated key

// Bob will choose the private key b

b = 3; // b is the chosen private key

printf("The private key b for Bob : %lld\n\n", b);

y = power(G, b, P); // gets the generated key

// Generating the secret key after the exchange

// of keys

ka = power(y, a, P); // Secret key for Alice

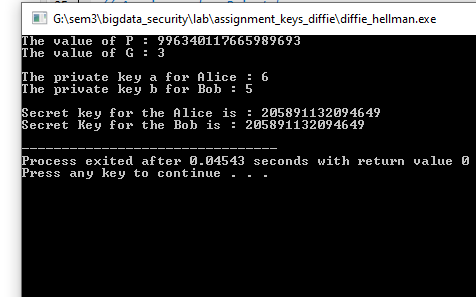
kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);

printf("Secret Key for the Bob is : %lld\n", kb);

return 0;

}



**Program 4:**

**Code to Find an integer k such that  where a and m are relatively prime.(Paste code and output)**

// C++ program to calculate discrete logarithm

#include<bits/stdc++.h>

using namespace std;

int discreteLogarithm(long long int a, long long int b, long long int m)

{

long long int n = (int) sqrt (m) + 1;

// Calculate a ^ n

int an = 1;

for (int i = 0; i<n; ++i)

an = (an \* a) % m;

unordered\_map<int, int> value;

// Store all values of a^(n\*i) of LHS

for (int i = 1, cur = an; i<= n; ++i)

{

if (! value[ cur ])

value[ cur ] = i;

cur = (cur \* an) % m;

}

for (int i = 0, cur = b; i<= n; ++i)

{

// Calculate (a ^ j) \* b and check

// for collision

if (value[cur])

{

int ans = value[cur] \* n - i;

if (ans < m)

return ans;

}

cur = (cur \* a) % m;

}

return -1;

}

// Driver code

int main()

{

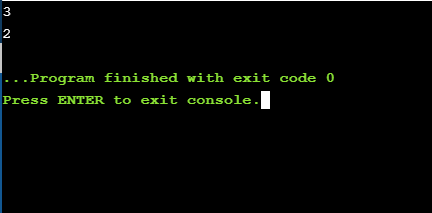
long long int a = 4566878, b = 6, m = 11;

cout << discreteLogarithm(a, b, m) << endl;

a = 3373663, b = 7, m = 6 ;

cout << discreteLogarithm(a, b, m);

**Output:**



**Program 5:**

**Apply this code to see how long it takes to break your chosen 3 keys. Record your observation as shown in the table(min 3 (p,g) values and for each (p,g) 3 private keys (weak, medium strong and strong)**

/\* This program calculates the Key for two persons

using the Diffie-Hellman Key exchange algorithm \*/

#include<stdio.h>

#include<math.h>

// Power function to return value of a ^ b mod P

long long int power(long long int a, long long int b,

long long int P)

{

if (b == 1)

return a;

else

return (((long long int)pow(a, b)) % P);

}

//Driver program

int main()

{

long long int P, G, x, a, y, b, ka, kb;

// Both the persons will be agreed upon the

// public keys G and P

P = 23;

// A prime number P is taken

printf("The value of P : %lld\n", P);

G = 9; // A primitve root for P, G is taken

printf("The value of G : %lld\n\n", G);

// Alice will choose the private key a

a = 14;//a is the chosen private key

printf("The private key a for Alice : %lld\n", a);

x = power(G, a, P); // gets the generated key

// Bob will choose the private key b

b = 19;// b is the chosen private key

printf("The private key b for Bob : %lld\n\n", b);

y = power(G, b, P); // gets the generated key

// Generating the secret key after the exchange

// of keys

ka = power(y, a, P); // Secret key for Alice

kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);

printf("Secret Key for the Bob is : %lld\n", kb);

return 0;

}

**Output:**

