Data Analysis and Algorithm

Practical 9

Write a program to implement multi threaded computation concepts in the generation of Fibonacci numbers.

Date.: 19-10-21

Name – Yash Vasudeo Prajapati

Rollno - 022

MSc. Computer Science

9) Write a program to implement multi-thread computation concepts in the generation of Fibonacci numbers.

Theory :-

What is a thread?

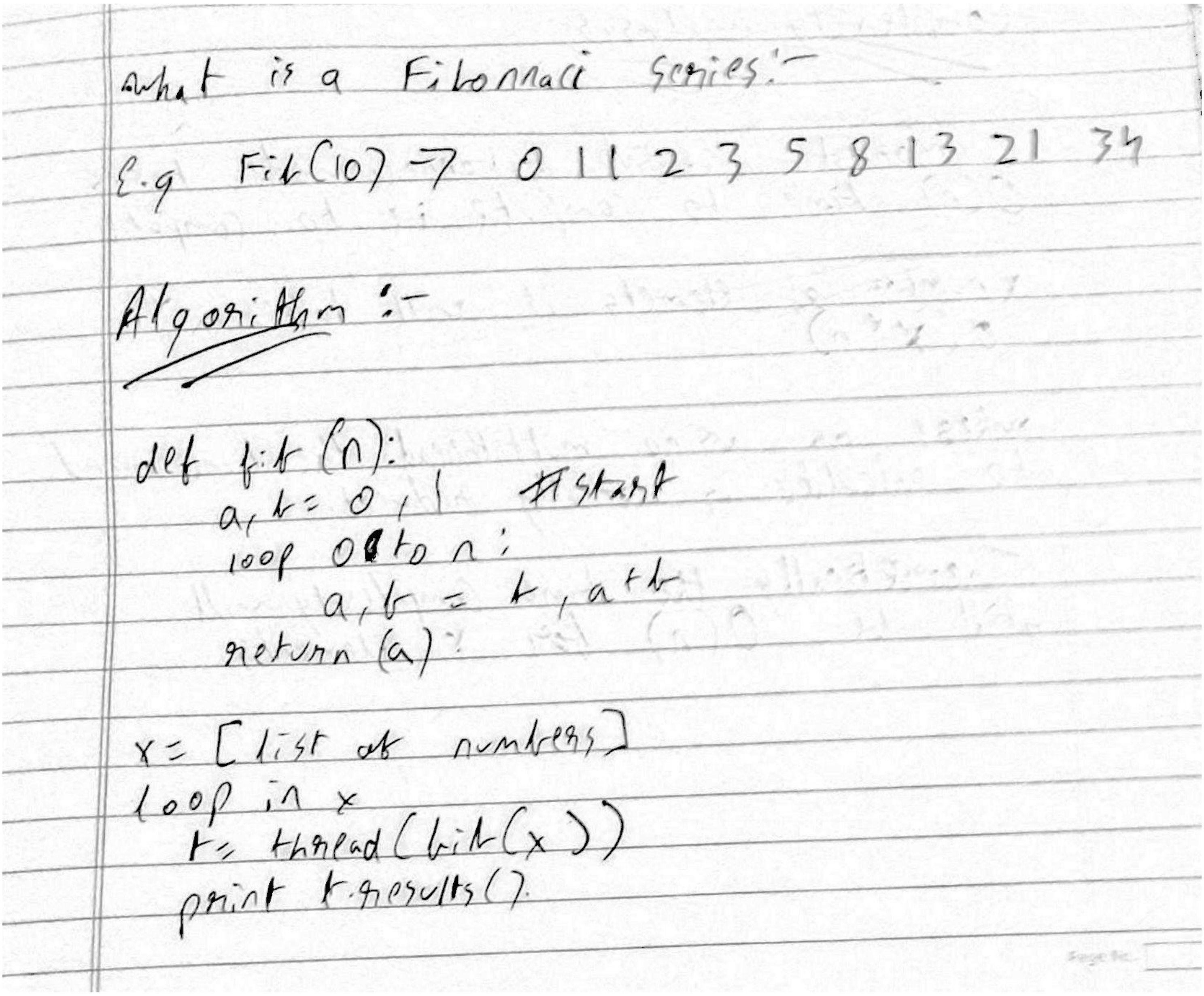
-> A thread is a path of execution within a process, where a process can contain a number of threads

Thus, the idea of running multiple threads in parallel to complete a process is called multi threading

Example :-

Consider a web browser, it has multiple process which need to be complete or been done. i.e provide UI, back-end processing, etc.

Thus we make two threads, consequently while providing UI, it may need multiple modules e.g menu, header, main, etc. here each module will be assigned to 1 thread.



what is a Fibonnaci series:-

E.g Fib(10) → 0 1 1 2 3 5 8 13 21 34

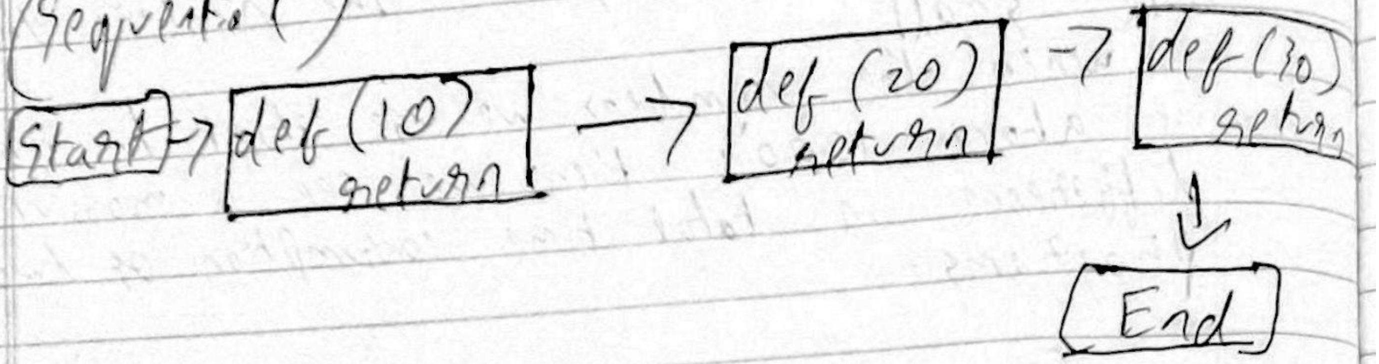
Algorithm :-

```
def fib (n):
    a, b = 0, 1    #start
    loop 0 to n:
        a, b = b, a+b
    return (a)

x = [list of numbers]
loop in x
  t = thread(fib(x))
  print t.results()
```

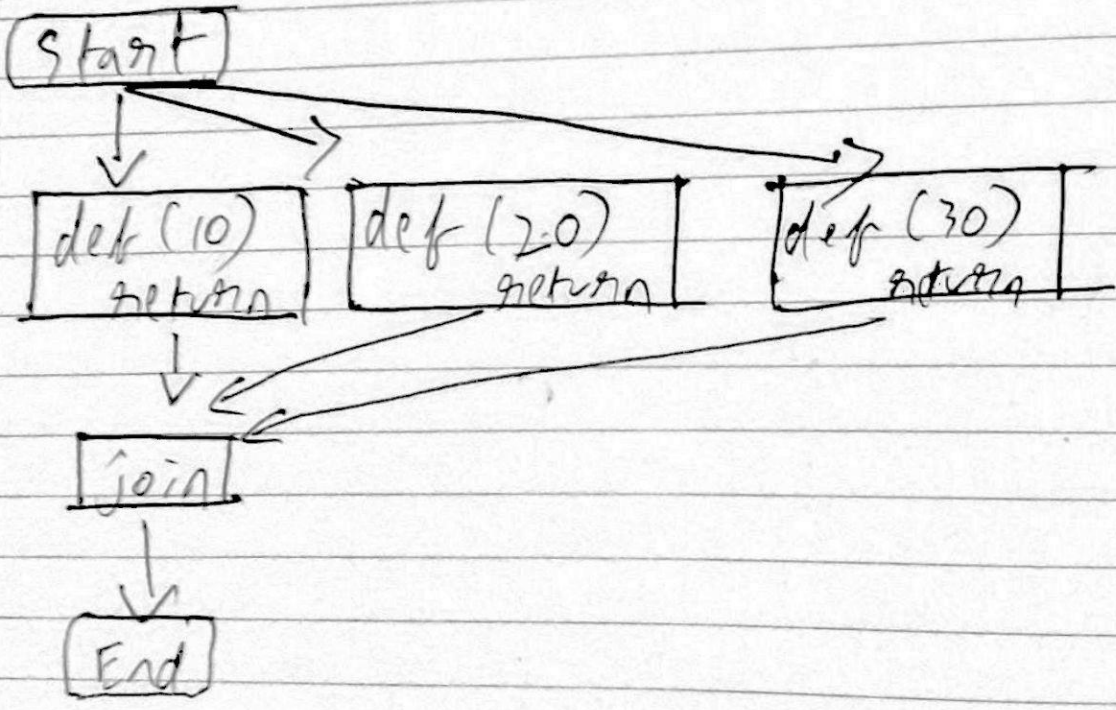
Eg:-

x = [10 ,20, 30]

(Sequential)



(Multithread)

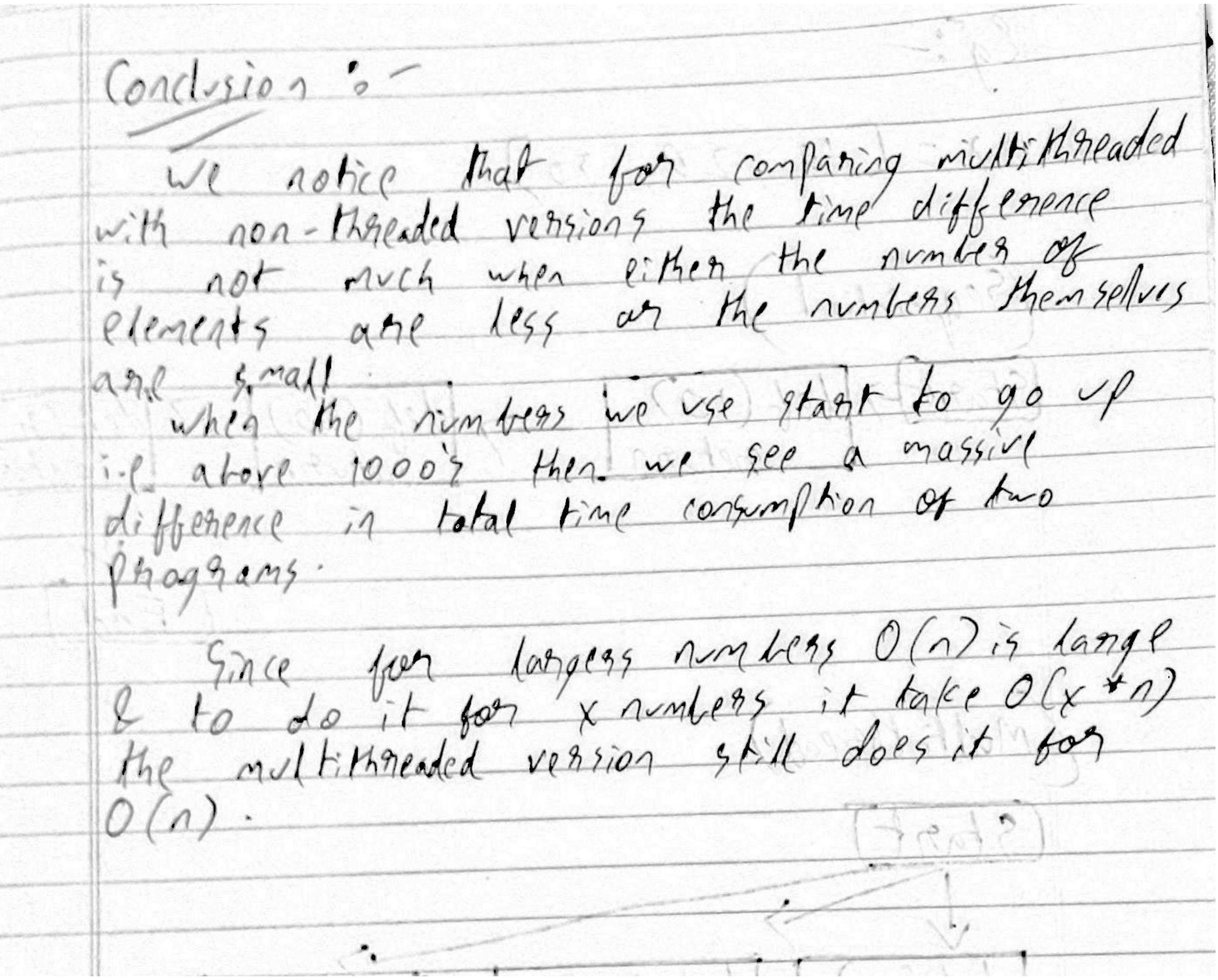


Complexity analysis

· Normal iterative fibonnaci code takes O(n) time to compute i.e to compute x number of elemets it will take x*n O(x*n)

where as using multithread the time need to calculate is greatly reduced.

Theoretically the time complexity will still be O(n) for x elements



Conclusion :-

We notice that for comparing multithreaded with non-threaded versions the time difference is not much when either the number of elements are less or the numbers themselves are small.

When the numbers we use start to go up i.e above 1000's then we see a massive difference in total time consumption of two programs.

Since for largers numbers $O(n)$ is large & to do it for x numbers it take $O(x*n)$ the multithreaded version still does it for $O(n)$.

Program:-

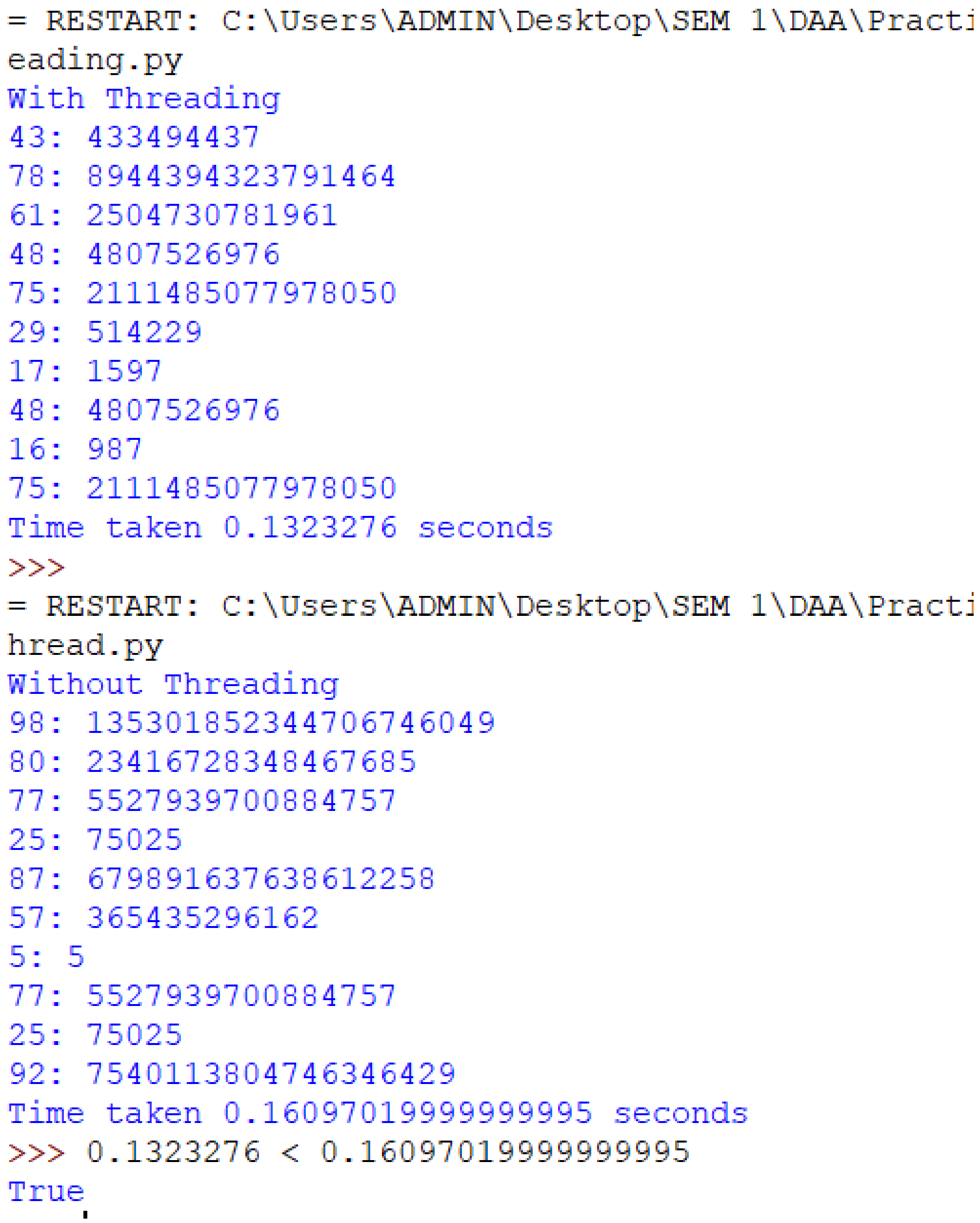
Multi-Threaded

```python
import queue, threading, random
from timeit import default_timer as timer

#setting up random list
x=[]
n=200
for i in range(n):
    x.append(random.randint(1000,10000))

print("With Threading in range (1000, 10000)")

#timer start
start = timer()
q = queue.Queue()

def fib(n):
    a, b = 0, 1
    for i in range(0, n):
        a, b = b, a + b
    q.put((n, a))
    return

for i in x:
    t = threading.Thread(target=fib, args = (i,))
    t.daemon = True
    t.start()

while not q.empty():
    n, f = q.get()
    #print ("{0}: {1}".format(n, f))
print("Time taken %s seconds" % (timer() - start))
```

Without Multi-Thread

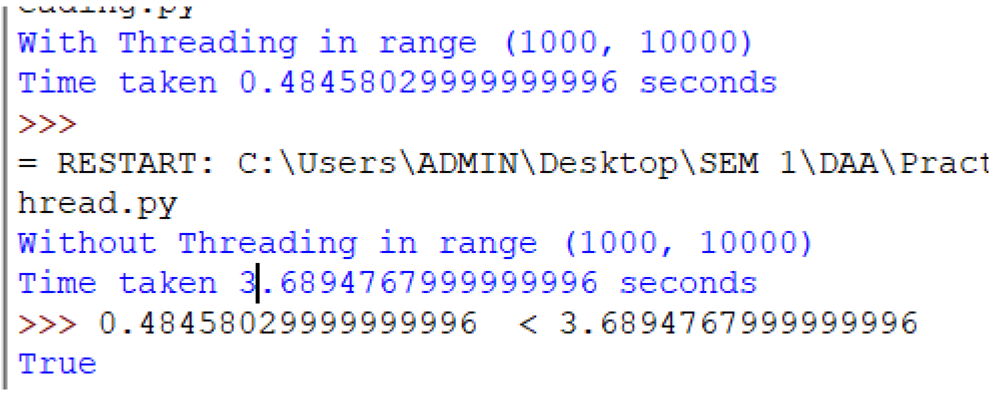
```python
import random
from timeit import default_timer as timer

#setting up random list
x=[]
n=2000
for i in range(n):
    x.append(random.randint(1000,10000))

print("Without Threading in range (1000, 10000)")
#timer start
start = timer()
arr = []

def fib(n):
    a, b = 0, 1
    for i in range(0, n):
        a, b = b, a + b
    return (a)

for i in range(0,len(x)):
    arr.append(fib(x[i]))
    #print("{0}: {1}".format(x[i], arr[i]))

print("Time taken %s seconds" % (timer() - start))
```

Output:-



```
= RESTART: C:\Users\ADMIN\Desktop\SEM 1\DAA\Practi
eading.py
With Threading
43: 433494437
78: 8944394323791464
61: 2504730781961
48: 4807526976
75: 2111485077978050
29: 514229
17: 1597
48: 4807526976
16: 987
75: 2111485077978050
Time taken 0.1323276 seconds
>>>
= RESTART: C:\Users\ADMIN\Desktop\SEM 1\DAA\Practi
hread.py
Without Threading
98: 135301852344706746049
80: 23416728348467685
77: 5527939700884757
25: 75025
87: 679891637638612258
57: 365435296162
5: 5
77: 5527939700884757
25: 75025
92: 7540113804746346429
Time taken 0.16097019999999995 seconds
>>> 0.1323276 < 0.16097019999999995
True
```

Increasing the size of numbers



```
eading.py
With Threading in range (1000, 10000)
Time taken 0.48458029999999996 seconds
>>>
= RESTART: C:\Users\ADMIN\Desktop\SEM 1\DAA\Pract
hread.py
Without Threading in range (1000, 10000)
Time taken 3.6894767999999996 seconds
>>> 0.48458029999999996  < 3.6894767999999996
True
```