

# INTRODUCTION TO ALGORITHMS

EC351-V SEM

## ASSIGNMENT-1

By – Jennifer yennam (18BEC018)

### Problem statement 1:

Consider the following Fibonacci series and solve the following conditions:

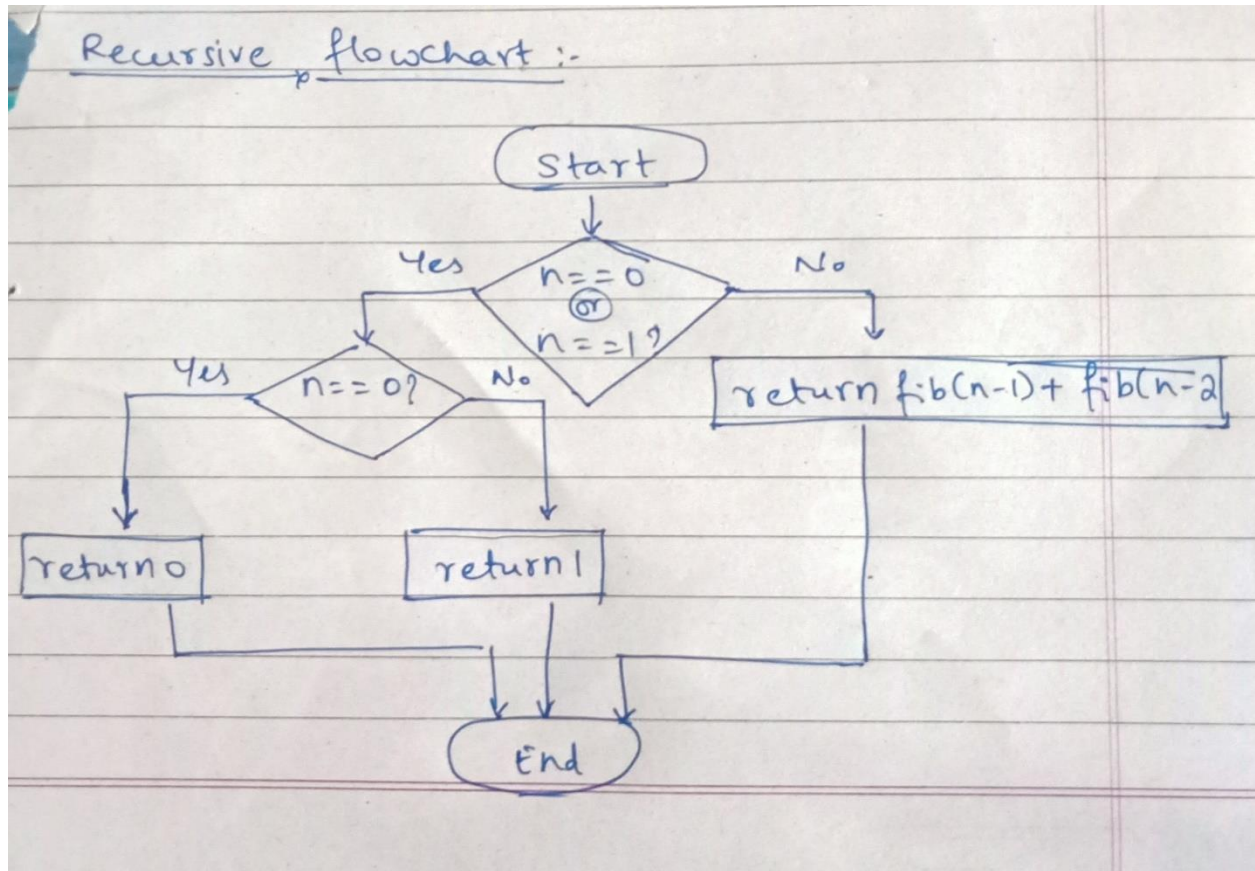
$\text{fib}(n) = \text{fib}(0), \text{fib}(1), \text{fib}(2), \dots, \text{fib}(n)$

where,  $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

1. Draw the flow chart, algorithms in pseudo code for solving.
2. Write two types of algorithm (recursive and non recursive) for  $\text{fib}(5)$  and  $\text{fib}(500)$  series.
3. Find out the total memory or space required to perform these Fibonacci series computational operations.
4. Find out the WORST CASE and BEST CASE scenario from the above identified approaches.
5. Write a program and compare the actual memory consumed by all the approaches.

(I) Fibonacci Series (recursive):

Flowchart :



Algorithm(Pseudo code):

Procedure fib(n):

If  $n=0$  then return 0

If  $n=1$  then return 1

Else

Return  $\text{fib}(n-1) + \text{fib}(n-2)$

End procedure

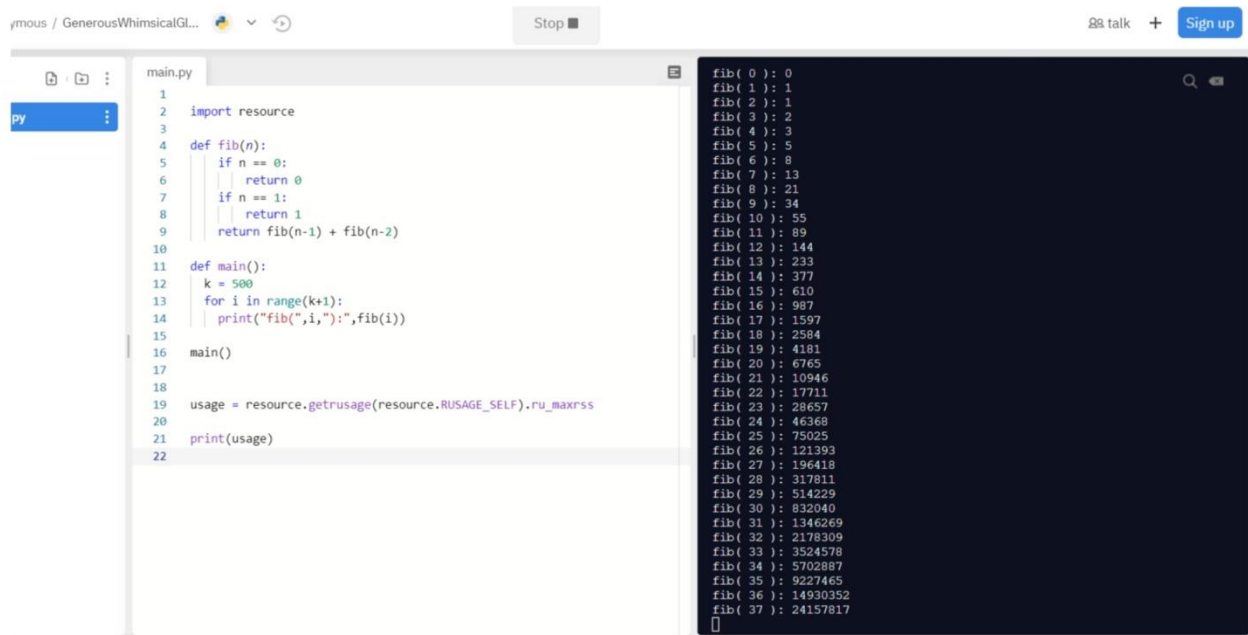
(Output will be in its nth Fibonacci number)

Here, the time complexity is exponential.

The execution time for,

fib(5)=0.00066304 seconds

fib(500)=



The screenshot shows a Jupyter Notebook interface. On the left, a file explorer shows a folder named 'py'. The main area displays a Python script in a file named 'main.py'. The script defines a recursive function 'fib(n)' and a 'main()' function that calculates Fibonacci numbers from 0 to 500. The output of the script is displayed on the right, showing the first 38 Fibonacci numbers (fib(0) to fib(37)).

```
1 import resource
2
3 def fib(n):
4     if n == 0:
5         return 0
6     if n == 1:
7         return 1
8     return fib(n-1) + fib(n-2)
9
10
11 def main():
12     k = 500
13     for i in range(k+1):
14         print("fib(",i,"):",fib(i))
15
16 main()
17
18 usage = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
19
20 print(usage)
```

fib( 0 ): 0  
fib( 1 ): 1  
fib( 2 ): 1  
fib( 3 ): 2  
fib( 4 ): 3  
fib( 5 ): 5  
fib( 6 ): 8  
fib( 7 ): 13  
fib( 8 ): 21  
fib( 9 ): 34  
fib( 10 ): 55  
fib( 11 ): 89  
fib( 12 ): 144  
fib( 13 ): 233  
fib( 14 ): 377  
fib( 15 ): 610  
fib( 16 ): 987  
fib( 17 ): 1597  
fib( 18 ): 2584  
fib( 19 ): 4181  
fib( 20 ): 6765  
fib( 21 ): 10946  
fib( 22 ): 17711  
fib( 23 ): 28657  
fib( 24 ): 46368  
fib( 25 ): 75025  
fib( 26 ): 121393  
fib( 27 ): 196418  
fib( 28 ): 317811  
fib( 29 ): 514229  
fib( 30 ): 832040  
fib( 31 ): 1346269  
fib( 32 ): 2178309  
fib( 33 ): 3524578  
fib( 34 ): 5702887  
fib( 35 ): 9227465  
fib( 36 ): 14930352  
fib( 37 ): 24157817

Observations:

The recursion algorithm gives adequate results, but only for generating Fibonacci series till 38-41. After 41 it takes a lot of time (nearly some hours) for the algorithm to execute and give out output, as the algorithm has exponential time complexity.

Therefore,

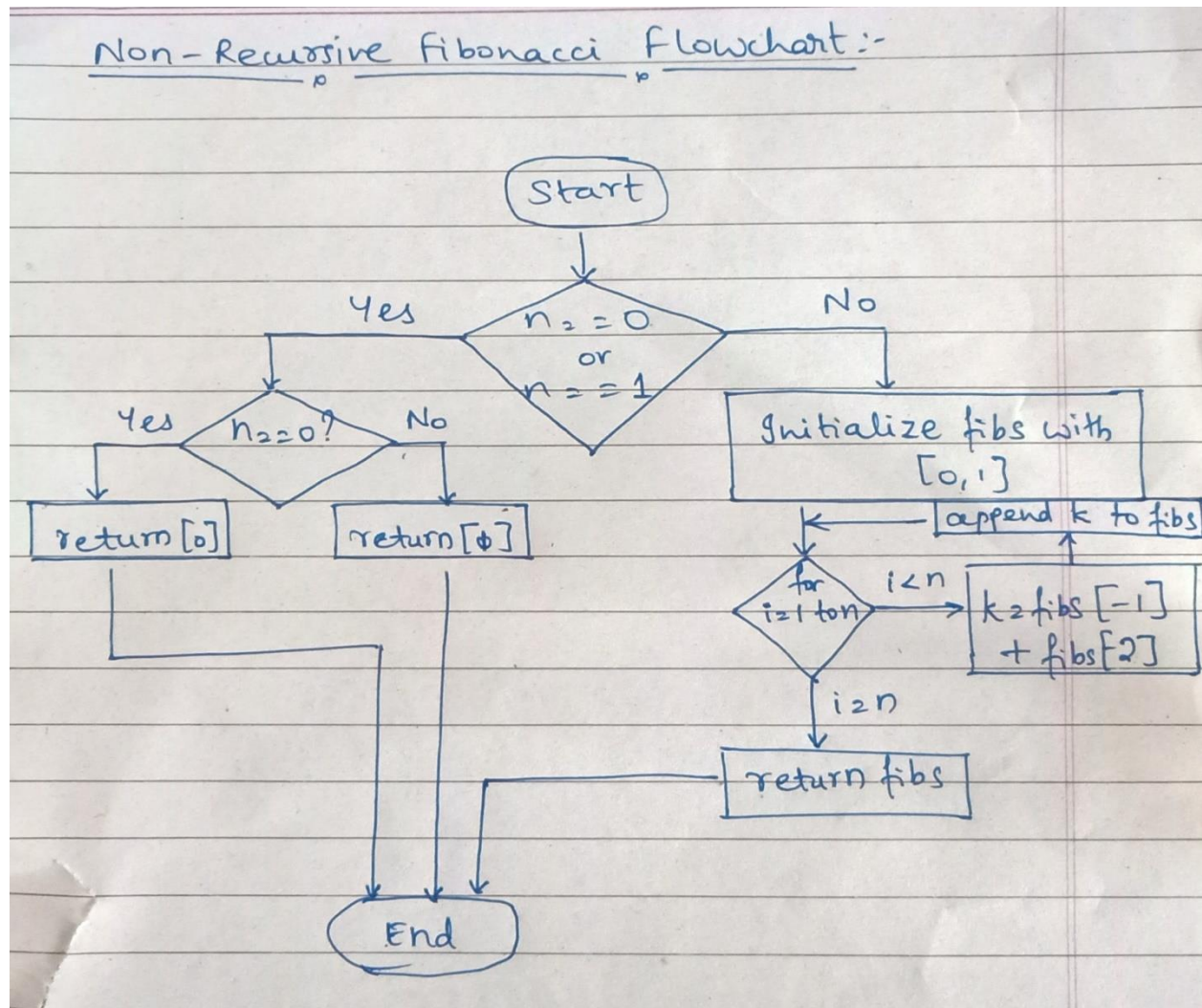
Best case – when n is less in fin(n) {n<42}

Worst case – when n is relatively large. {n>100}

Memory usage – 17668 bytes (calculated using 'resource' package of python)

## (II) Fibonacci Series (non recursive):

Flowchart:



Algorithm(pseudo code):

procedure fib(n):

if  $n=0$  then return(0)

if  $n=1$  then return (1)

initialize list fibs with (0,1)

For i=1to n

k=fibs[-1] + fibs[-2]

append k to fibs

return fibs

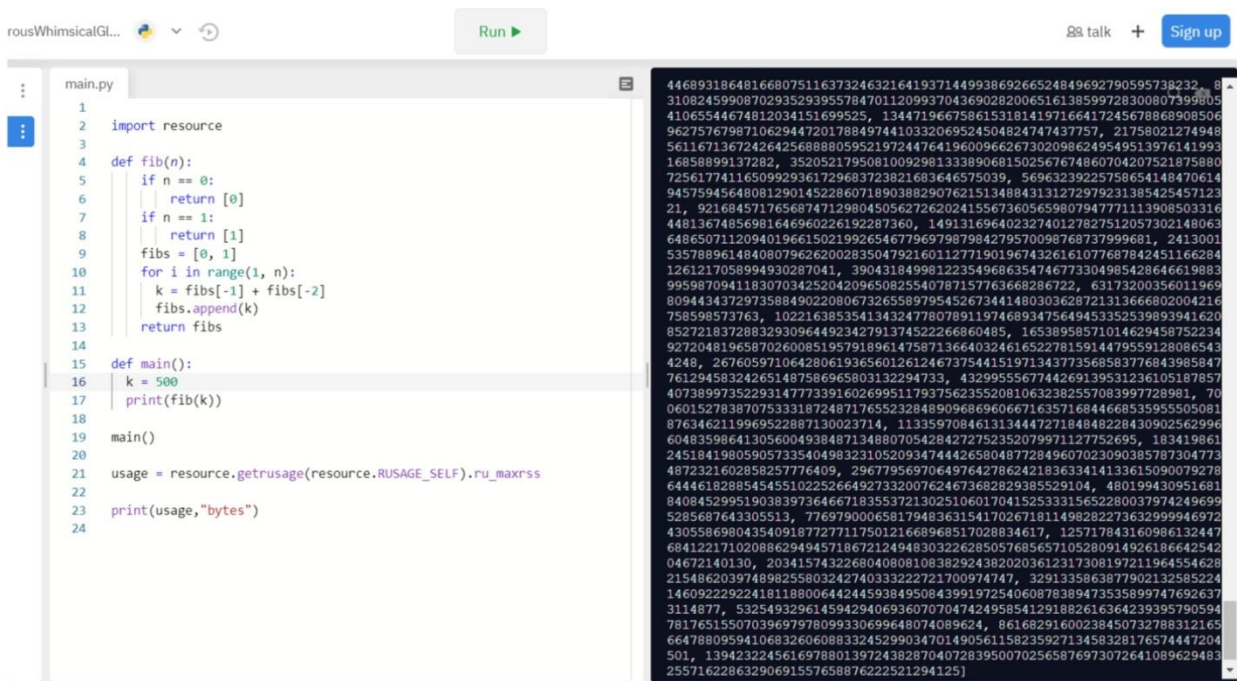
end procedure

(output is Fibonacci series upto n)

Execution time :

Fib(5) – 5.3644e<sup>-05</sup> secs

Fib(500) – 0.0023839 secs



The screenshot shows a code editor with a Python script in a file named 'main.py'. The script defines a function 'fib(n)' that calculates the nth Fibonacci number using a list 'fibs'. It then defines a 'main()' function that sets 'k = 500' and prints 'fib(k)'. The script also includes a resource usage check using 'resource.getrusage(resource.RUSAGE\_SELF).ru\_maxrss' and prints the usage in bytes. The output of the script is displayed in a terminal window on the right, showing the first 500 Fibonacci numbers and the memory usage of 20068 bytes.

```
1 import resource
2
3
4 def fib(n):
5     if n == 0:
6         return [0]
7     if n == 1:
8         return [1]
9     fibs = [0, 1]
10    for i in range(1, n):
11        k = fibs[-1] + fibs[-2]
12        fibs.append(k)
13    return fibs
14
15 def main():
16     k = 500
17     print(fib(k))
18
19 main()
20
21 usage = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
22
23 print(usage, "bytes")
24
```

446893186481668075116373246321641937144993869266524849692790595738232... 8  
310824599087029352939557847011209937043690282006516138599728300807399805  
41065544674812034151699525, 1344719667586153181419716641724567886808506  
96275767987106294472017884974410332069524504824747437757, 21758021274948  
56116713672426425689805952197244764196009662673020986249549513976141993  
16858899137282, 35285217950810092961133890681502567674860704207521875800  
7256177411650929361729683723821683646575039, 56963239225758654148470614  
945759456480812901452286071890388290762151348843131272979231385425457123  
21, 9216845717656874712980450562726202415567360565980794771113908503316  
44813674856981646960226192287360, 14913169640232740127827512057302148063  
648650711209401966150219926546779697987984279570098768737999681, 2413001  
535788961484080796262002835047921601127719019674326161077687842451166284  
1261217058994930287041, 390431849981223549686354746773304985428646619893  
99598709411830703425204209650825540787157763668286722, 63173200356011969  
809443437297358849022080673265589795452673441480303628721313666802004216  
758598573763, 1022163853541343247780789119746893475649453352539893941620  
85272183728832930964492342791374522266860485, 16538958571014629458752234  
927204819658702600851957918961475871366403246165227815914479559128086543  
4248, 267605971064280619365601261246737544151971343773568583776843985847  
761294583242651487586965803132294733, 432995567744269139531236105187857  
40738997352293147773391602699511793756235520810632382557083997728981, 70  
06015278387075333187248717655232848909686960671635716844668535955505081  
8763462119969522887130023714, 113359708461313444727184848228430902562996  
6048359864130560049384871348807054284272752352079971127752695, 183419861  
245184198059057335404983231052093474442658048772849607023090385787304773  
4872321602858257776409, 2967795697064976427864242183633414133615090079278  
6444618288545455102252664927332007624673682829385529104, 480199430951681  
840845299519038397364667183553721302510601704152533315652280037974249699  
5285687643305513, 776979000658179483631541702671811498282273632999946972  
4305586980435409187727711750121668968517028834617, 125717843160986132447  
684122171020886294945718672124948303226285057685657105280914926186642542  
04672140130, 20341574322680408081083829243820203612317308197211964554628  
21548520387489825580324274033222721700974747, 32913386387790213258524  
146092229224181188006442445938495084399197254060878389473535899747692637  
3114877, 532549329614594294069360707047424958541291882616364239395790594  
7817651550703969797809330699648074089624, 86168291600238450722788312165  
66478809594106832606088324529903470149056115823592713458328176574447204  
501, 1394232245616978801397243828704072839500702565876973072641089629483  
25571622863290691557658876222521294125]

Best case – it is a good option for computing Fibonacci series for large n.

Memory usage – 20068 bytes (calculated using 'resource' package of python)



## Observation:

1. For generating Fibonacci series upto a small value of  $n$ , any of the other algorithms can be used.
2. But, for generating Fibonacci series for a larger value of  $n$  that is for  $n=100,300,500,1000$  etc the recursive algorithm takes exponential time and the non-recursive algorithm takes minimal time. So, the second algorithm would be better for large  $n$ .

## Example :

Fibonacci(10000):

1. Takes 127.09 seconds and consumes only 38152 bytes of memory using 2<sup>nd</sup> approach.
2. Can take hours (days) altogether using the recursive approach.

```
main.py
1
2 import time
3 start = time.time()
4 def fib(n):
5     if n == 0:
6         return [0]
7     if n == 1:
8         return [1]
9     fibs = [0, 1]
10    for i in range(1, n):
11        k = fibs[-1] + fibs[-2]
12        fibs.append(k)
13    return fibs
14
15 def main():
16     k = 10000
17     print(fib(k))
18
19 main()
20 end = time.time()
21
22
23 print(end-start, "seconds")
24
```

```
053383165360388595116980245927935225901537634925654872380877183009301074
569444002426436414756905094535072804764684492105680024739914490553904391
369218696387092918189246157103450387050229300603241611410707453960080170
928277951834763216705242485820801423866526633816082921442883095463259080
471819329201710147828025221385656340207489796317663278872207607791034431
700112753558813478888727503825389066823098683355695718137867882982111710
796422706778536913192342733364556727928018953989153106047379741280794091
639429908796650294603536651238230626, 3364476487643178326662161200510754
331030214846068006390656476997468008144216666236815559551363373402558206
533268083615937373479048386526826304089246305643188735454436955982749160
660209988418393386465273130008883026923567361313511757929743785441375213
052050434770160226475831890652789085515436615958298727968298751063120057
542878345321551510387081829896979161312785626503319548714021428753269818
796204693609787990035096230229102636813149319527563022783762844154036058
440257211433496118002309120828704608892396232883546150577658327125254609
359112820392528539343462090424524892940390170623388899108584106518317336
043747073790855263176432573399371287193758774689747992630583706574283016
163740896917842637862421283525811282051637029808933209990570792006436742
6202389783111470054074984592503606335609338838192338678305613643535189
21332797329081337326426526339897639227234078829281779538057099369104917
547080893184105614632233821746563732124822638309210329770164805472624384
237486241145309381220656491403275108664339451751216152654536133311131404
243685480510676584349352383695965342807176877532834823434555736671973139
274627362910821067928078471803532913117677892465908993863545932789452377
767440619224033763867400402133034329749690202832814593341882681768389307
200363479562311710310129195316979460763273758925353077255237594378843450
40677155577905645044301664011946258097221672975861502696844316495203461
493229110597067624326851599283470989128470674086200858713501626031207190
317208609408129832158107728207635318662461127824553720853236530577595643
007251774431505153960090516860322034916322264088524885243315805153484962
243484829938090507048348244932745373262456775587908918719080366205800959
474315005240253270974699531877072437682590741993963226598414749819360928
522394503970716544315642132815768890805878318340491743455627052022356484
649519611246026831397097506938264870661326450766507461151267752274862159
864253071129844118262266105716351506926002986170494542504749137811515413
994155067125627119713325276363193960690289565028826860836224108205056243
0701794976171121233066073310059947366875]
127.0921086921692 seconds
```