

Introduction to Algorithms

EC301

Assignment-1

By- Harshil Goolla (18BEC014)

Yashwanth K (18BEC020)

Bharath (18BEC023)

Srinivas Nayaka (18BEC046)

Pramod Betha (18BEC007)

Aditi M (18BEC002)

1. Fibonacci Series (Recursive) :

Algorithm/Pseudocode :

- procedure fibonacci(n):
- if $n = 0$ then return 0
- if $n = 1$ then return 1
- else
- return $\text{fibonacci}(n-1) + \text{fibonacci}(n-2)$
- end procedure

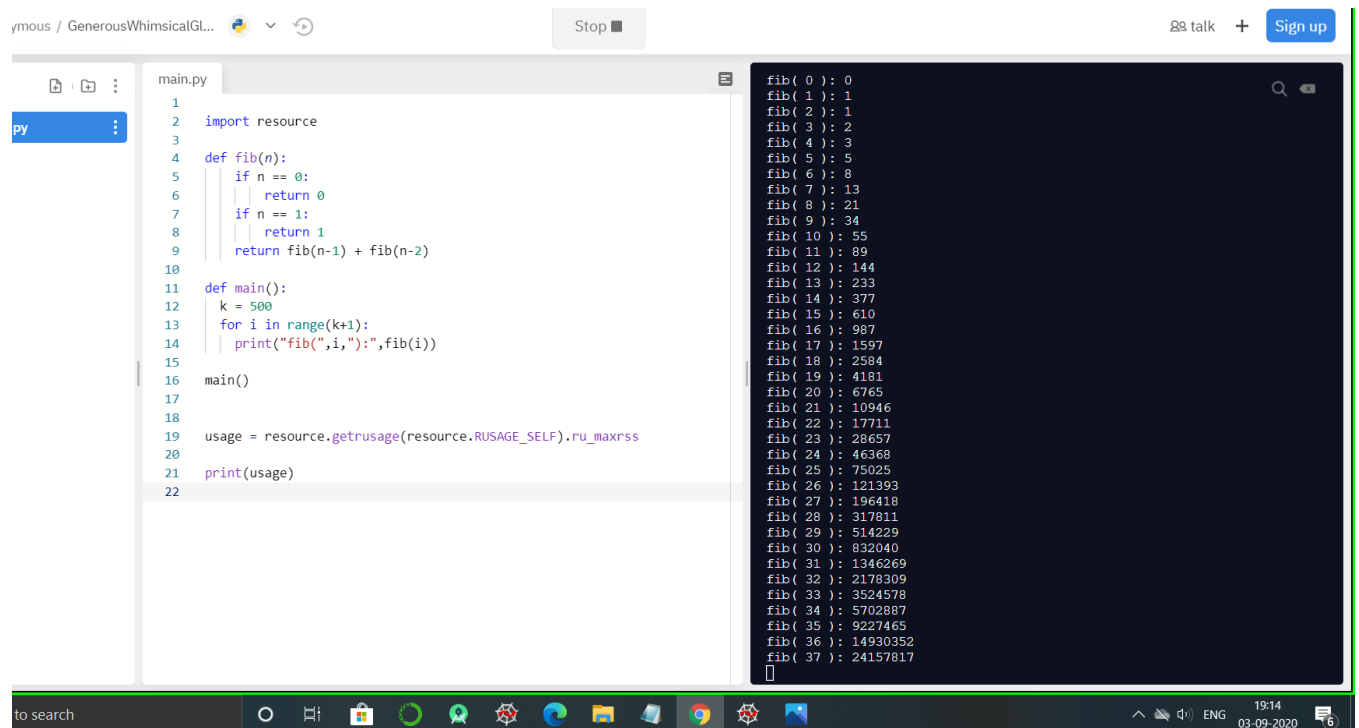
(Output is nth fibonacci number)

Time Complexity: Exponential

Execution Time :

Fibonacci(5) : 0.00066304 seconds

Fibonacci(500) :



The screenshot shows a Jupyter Notebook interface. The left pane displays the code for a recursive Fibonacci function and a main function that prints the first 500 Fibonacci numbers. The right pane shows the output of the function for values of n from 0 to 37. The output is a list of Fibonacci numbers, with the last value being 24157817. The notebook is titled 'ymous / GenerousWhimsicalGL...' and has a 'Stop' button. The bottom status bar shows the time as 19:14 on 03-09-2020.

```
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
```

Observation:

The recursion algorithm gives satisfactory results only for generating Fibonacci series till 38-41. After that it takes a lot of time, nearly some hours as the algo has exponential time complexity.

So for this algorithm, The Best Case is : when n is low in fib(n) {n<42}

The Worst Case is : when n is relatively large. { n > 100 }

Memory Usage : 17668 Bytes (Calculated using 'resource' package of python)

2. Fibonacci Series (non-recursive):

Algorithm/Pseudocode :

- procedure fibonacci(n)
- if n = 0 then return [0]
- if n = 1 then return [1]
- initialize list fibs with [0,1]
- for i = 1 to n

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

append k to fibs

- return fibs
- end procedure

(Output is fibonacci series upto n)

Execution Time :

Fibonacci(5) : 5.3644e^-05 secs

Fibonacci(500) : 0.0023839 secs

Best Case : Good choice for computing Fibonacci series for large n.

```
main.py
1
2 import resource
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...
31082459087029352939557847011209937043690282006516138599728300807399805
41065544674812034151699525, 13447196675861531814197166417245678868908506
96275767987106294472017884974410332069524504824747437757, 21758021274948
56116713672426425688805952197244764196009662673020986249549513976141993
16858899137282, 35205217950810092981333890681502567674860704207521875880
72561774116509929361729683723821683646575039, 56963239225758654148470614
945759456480812901452286071890388290762151348843131272979231385425457123
21, 92168457176568747129804505627262024155673605659807947771113908503316
44813674856981646960226192287360, 14913169640232740127827512057302148063
648650711209401966150219926546779697987984279570098768737999681, 2413001
535788961484080796262002835047921601127719019674326161077687842451166284
1261217058994930287041, 390431849981223549686354746773304985428646619883
995970941183070842520420965082540787157763668286722, 6317320056011969
80944343729735884902208067326558979545267344148030362872131366802004216
758598573763, 1022163853541343247780789119746893475649453352539893941620
85272183728832930964492342791374522266860485, 16538958571014629458752234
92720481965870260085195791896147587136640324616522781591447955912808543
4248, 267605971064280619365601261246737544151971343773568583776843985847
761294583242651487586965803132294733, 4329955567744269139531236105187857
40738997352293147773391602699511793756235520810632382557083997728981, 70
06015278387075331872487176552328489096869606671635716844668535955505081
8763462119969522887130023714, 113359708461313444727184848228430902562996
6048359864130560049384871348807054284272752352079971127752695, 183419861
245184198059057335404983231052093474442658048772849607023090385787304773
4872321602858257776409, 296779569706497642786242183633414133615090079278
6444618288545455102252664927332007624673682829385529104, 480199430951681
840845299519038397364667188553721302510601704152533315552280037974249695
5285687643305513, 7769790006581194893615417026718114982822736329994672
430558698043540918727711750121668968517028834617, 125717843160986132447
68412217102088629494571867212494830322685057685657105280914926186642542
04672140130, 20341574322680408081083829243820203612317308197211964554628
21548620397489825580324274033222721700974747, 3291335863877902132585224
146092229224181188006442445938495084399197254060878389473535899747692637
3114877, 532549329614594294069360707047424958541291882616364239395790594
78176515507039697978099330699648074089624, 86168291600238450732788312165
664788095941068326060883324529903470149056115823592713458328176574447204
501, 1394232245616978801397243828704072839500702565876973072641089629483
2557162286329069155765887622521294125]
```

Memory Usage :

20068 Bytes (Calculated using 'resource' package of python)

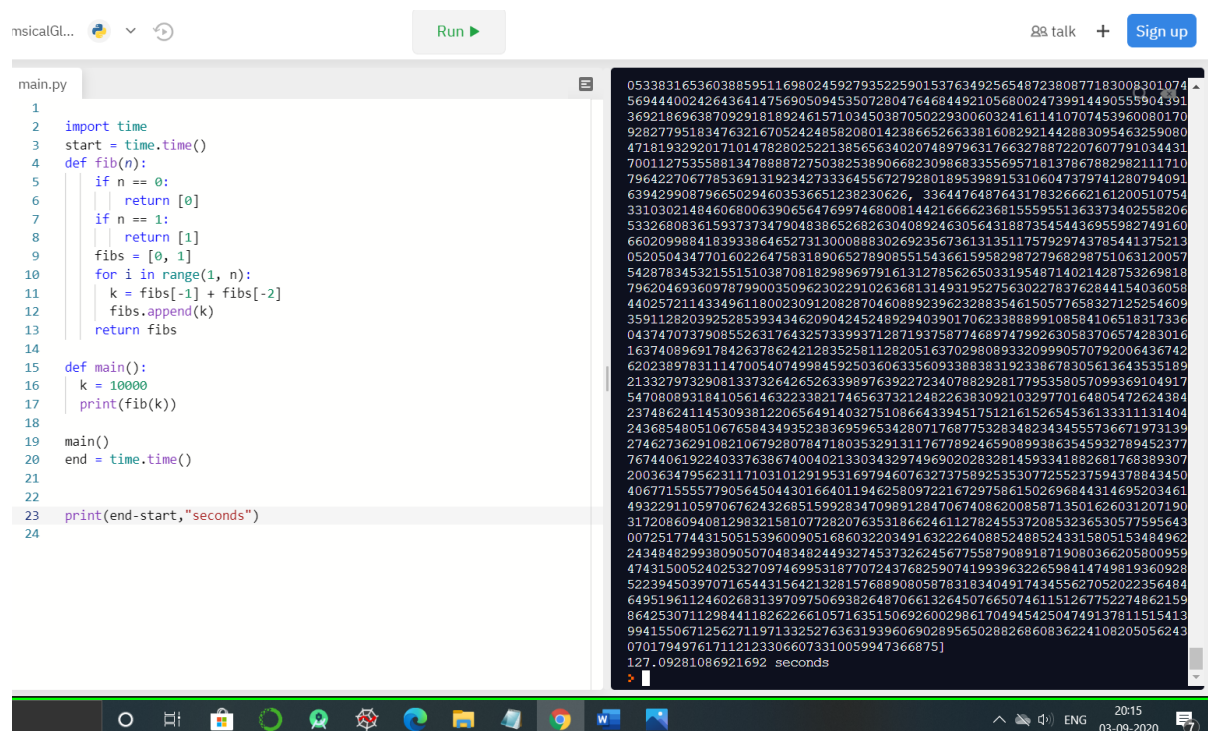
Observation :

- For generating Fibonacci Series upto a small value of n , any of the above algorithms can be used.
- But, for generating Fibonacci Series for a larger value of n .i.e, 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) :

- Takes 127.09 seconds and consumes only 38152 bytes of memory using 2nd approach.
- Can take hours or days altogether using the recursive approach.



```
msicalGL... Run
```

```
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
```

```
053383165360388595116980245927935225901537634925654872380877183008301074
569444002426436414756905094535072804764684492105680024739914490555904391
369218696387092918189246157103450387050229300603241611410707453960080170
928277951834763216705242485820801423866526633816082921442883095463259080
471819329201710147828025221385656340207489796317663278872207607791034431
700112753558813478888727503825389066823098683355695718137867882982111710
796422706778536913192342733364556727928018953989153106047379741280794091
639429908796650294603536651238230626, 3364476487643178326662161200510754
331030214846068006390656476997468008144216666236815559551363373402558206
53326808361593737347904838652682630408924630564318873545443695982749160
660209988418393386465273130008883026923567361313511757929743785441375213
052050434770160226475831890652789085515436615958298727968298751063120057
542878345321551510387081829896979161312785626503319548714021428753269818
796204693609787990035096230229102636813149319527563022783762844154036058
440257211433496118002309120828704608892396232883546150577658327125254609
359112820392528539343462090424524892940390170623388899108584106518317336
043747073790855263176432573399371287193758774689747992630583706574283016
163740896917842637862421283525811282051637029808933209990570792006436742
62023897831114700540749984592503606335609338838139233867830561364353189
21332797329081337326426526339897639227234078829817795358057099369104917
547080893184105614632233821746563732124822638309210322770164805472624384
237486641145309381220656491403275108664339451751216152654536133311131404
24368548051067658434835238369565342807176877532834823434555736671973138
274627362910821067928078471803532613117677882465908993863545932789452377
76744619224033763867400402133034329749690202832814593431882681768389307
200363479562311710310129195316979460763273758925353077255237594378843450
406771555577905645044301664011946258097221672875861502696844314695203461
493229110597067624326851599283470989128470674086200858713501626031207190
317208609408129832158107728207635318662461127824553720853236530577595643
007251774431505153960090516860322034916322264088524885243315805153484962
24348482993809050704834824493274537326245677558790891871908036620580959
4743150052402523270974699531877072437682590741993963226598414749819360928
522394503970716544315642132815768890805878318340491743455627052022356484
649519611246026831397097506938264870661326450766507461151267752274862159
864253071129844118262266105716351506926002986170494542504749173811515413
994155067125627119713325276363193960690289565028826860836224108205056243
070179497617121233066073310059947366875]
127.09281086921692 seconds
```