PRACTICAL 2 – RECURSION

Thi Van Anh DUONG ID 90023112

Diploma of Information Technology, Curtin College

DSA1002: Data Structures and Algorithms

Coordinator: Khurram Hameed

13 July 2022

Student Declaration of Originality

☒	This assignment is my own original work, and no part has been copied from another student's work or source, except where it is clearly attributed,
☒	All facts/ideas/claims are from academic sources are paraphrased and cited/referenced correctly,
☒	I have not previously submitted this work in any form for THIS or for any other unit; or published it elsewhere before
☒	No part of this work has been written for me by another person,
☒	I recognise that should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with Curtin College policy.

Electronic Submission:

☒	I accept that it is my responsibility to check that the submitted file is the correct file, is readable and has not been corrupted,
☒	I acknowledge that a submitted file that is unable to be read cannot be marked and will be treated as a non-submission
☒	I hold a copy of this work if the original is damaged, and will retain a copy of the Turnitin receipt as evidence of submission

ACTIVITY 1: FACTORIAL AND FIBONACCI

Fibonacci implementation:

```python
import time

def fibIterative(n):
    fibVal = 0
    currVal = 1
    lastVal = 0

    if (n == 0 ):
        fibVal = 0
    elif (n == 1):
        fibVal = 1
    else:
        for i in range (n):
            fibVal = currVal + lastVal
            lastVal = currVal
            currVal = fibVal

    return fibVal

def fibRecursion(n):
    fibVal = 0

    if (n == 0):
        fibVal = 0
    elif ( n == 1):
        fibVal = 1
    else:
        fibVal = fibRecursion(n - 1) + fibRecursion(n - 2)
    return fibVal


start = time.time()
for i in range(38):
    #print("fibonacci(" + str(i) + ")",fibIterative(i))
    print("fibonacci(" + str(i) + ")",fibRecursion(i))

print("Time: ", time.time() - start)



~
```

Testing with fibIterative(n):

n = 38: (result has been showed immediately)

```
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$ python3 fibonacci.py
fibonacci(0) 0
fibonacci(1) 1
fibonacci(2) 2
fibonacci(3) 3
fibonacci(4) 5
fibonacci(5) 8
fibonacci(6) 13
fibonacci(7) 21
fibonacci(8) 34
fibonacci(9) 55
fibonacci(10) 89
fibonacci(11) 144
fibonacci(12) 233
fibonacci(13) 377
fibonacci(14) 610
fibonacci(15) 987
fibonacci(16) 1597
fibonacci(17) 2584
fibonacci(18) 4181
fibonacci(19) 6765
fibonacci(20) 10946
fibonacci(21) 17711
fibonacci(22) 28657
fibonacci(23) 46368
fibonacci(24) 75025
fibonacci(25) 121393
fibonacci(26) 196418
fibonacci(27) 317811
fibonacci(28) 514229
fibonacci(29) 832040
fibonacci(30) 1346269
fibonacci(31) 2178309
fibonacci(32) 3524578
fibonacci(33) 5702887
fibonacci(34) 9227465
fibonacci(35) 14930352
fibonacci(36) 24157817
fibonacci(37) 39088169
Time:  0.0022058486938476562
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$
```

n = 40 or even n = 50 or even higher (n=100) the result continuously is showed quickly

```
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$ python3 fibonacci.py
fibonacci(0) 0
fibonacci(1) 1
fibonacci(2) 2
fibonacci(3) 3
fibonacci(4) 5
fibonacci(5) 8
fibonacci(6) 13
fibonacci(7) 21
fibonacci(8) 34
fibonacci(9) 55
fibonacci(10) 89
fibonacci(11) 144
fibonacci(12) 233
fibonacci(13) 377
fibonacci(14) 610
fibonacci(15) 987
fibonacci(16) 1597
fibonacci(17) 2584
fibonacci(18) 4181
fibonacci(19) 6765
fibonacci(20) 10946
fibonacci(21) 17711
fibonacci(22) 28657
fibonacci(23) 46368
fibonacci(24) 75025
fibonacci(25) 121393
fibonacci(26) 196418
fibonacci(27) 317811
fibonacci(28) 514229
fibonacci(29) 832040
fibonacci(30) 1346269
fibonacci(31) 2178309
fibonacci(32) 3524578
fibonacci(33) 5702887
fibonacci(34) 9227465
fibonacci(35) 14930352
fibonacci(36) 24157817
fibonacci(37) 39088169
Time:  0.0010612010955810547
```

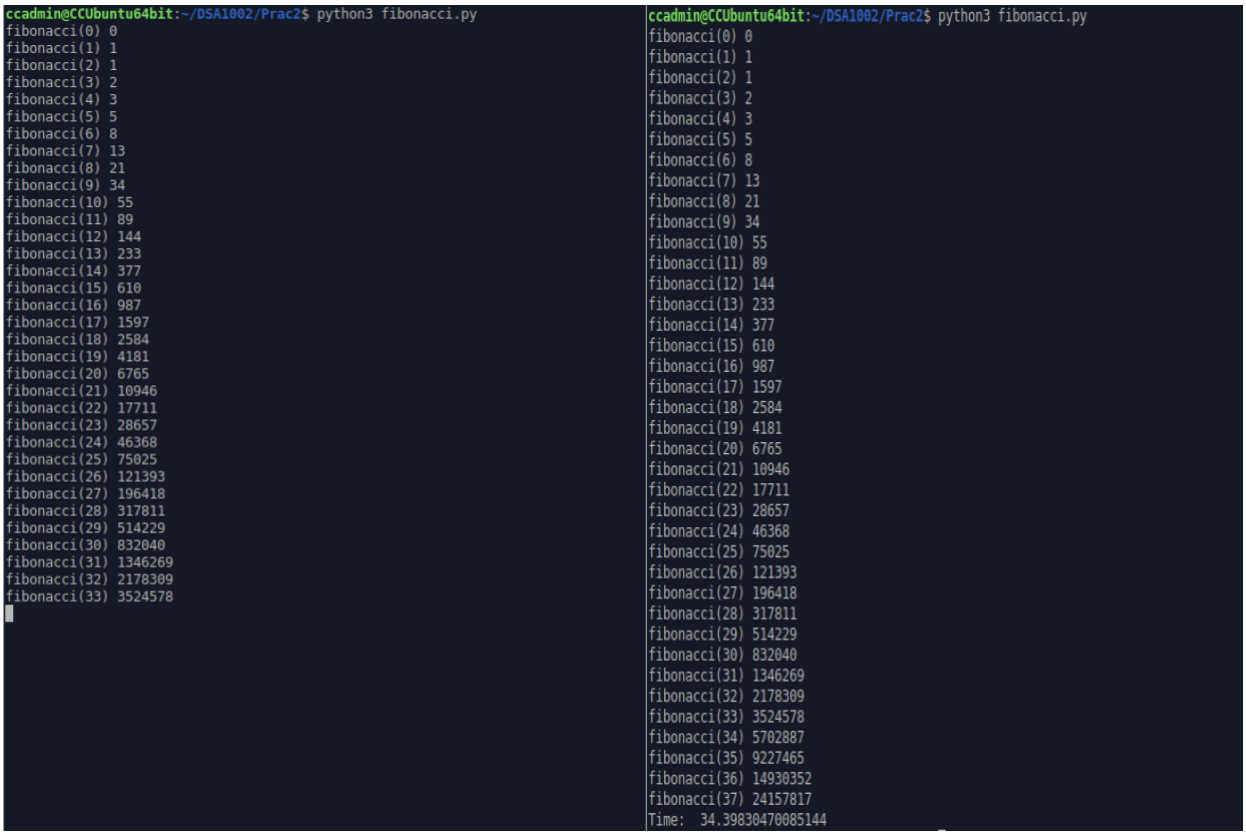
```
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$ python3 fibonacci.py
fibonacci(0) 0
fibonacci(1) 1
fibonacci(2) 2
fibonacci(3) 3
fibonacci(4) 5
fibonacci(5) 8
fibonacci(6) 13
fibonacci(7) 21
fibonacci(8) 34
fibonacci(9) 55
fibonacci(10) 89
fibonacci(11) 144
fibonacci(12) 233
fibonacci(13) 377
fibonacci(14) 610
fibonacci(15) 987
fibonacci(16) 1597
fibonacci(17) 2584
fibonacci(18) 4181
fibonacci(19) 6765
fibonacci(20) 10946
fibonacci(21) 17711
fibonacci(22) 28657
fibonacci(23) 46368
fibonacci(24) 75025
fibonacci(25) 121393
fibonacci(26) 196418
fibonacci(27) 317811
fibonacci(28) 514229
fibonacci(29) 832040
fibonacci(30) 1346269
fibonacci(31) 2178309
fibonacci(32) 3524578
fibonacci(33) 5702887
fibonacci(34) 9227465
fibonacci(35) 14930352
fibonacci(36) 24157817
fibonacci(37) 39088169
fibonacci(38) 63245986
fibonacci(39) 102334155
fibonacci(40) 165580141
fibonacci(41) 267914296
fibonacci(42) 433494437
fibonacci(43) 701408733
fibonacci(44) 1134903170
fibonacci(45) 1836311903
fibonacci(46) 2971215073
fibonacci(47) 4807526976
fibonacci(48) 7778742049
fibonacci(49) 12586269025
Time:  0.0010936260223388672
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$
```

```
fibonacci(48) 7778742049
fibonacci(49) 12586269025
fibonacci(50) 20365011074
fibonacci(51) 32951280099
fibonacci(52) 53316291173
fibonacci(53) 86267571272
fibonacci(54) 139583862445
fibonacci(55) 225851433717
fibonacci(56) 365435296162
fibonacci(57) 591286729879
fibonacci(58) 956722026041
fibonacci(59) 1548008755920
fibonacci(60) 2504730781961
fibonacci(61) 4052739537881
fibonacci(62) 6557470319842
fibonacci(63) 10610209857723
fibonacci(64) 17167680177565
fibonacci(65) 27777890035288
fibonacci(66) 44945570212853
fibonacci(67) 72723460248141
fibonacci(68) 117669030460994
fibonacci(69) 190392490709135
fibonacci(70) 308061521170129
fibonacci(71) 498454011879264
fibonacci(72) 806515533049393
fibonacci(73) 1304969544928657
fibonacci(74) 2111485077978050
fibonacci(75) 3416454622906707
fibonacci(76) 5527939700884757
fibonacci(77) 8944394323791464
fibonacci(78) 14472334024676221
fibonacci(79) 23416728348467685
fibonacci(80) 37889062373143906
fibonacci(81) 61305790721611591
fibonacci(82) 99194853094755497
fibonacci(83) 160500643816367088
fibonacci(84) 259695496911122585
fibonacci(85) 420196140727489673
fibonacci(86) 679891637638612258
fibonacci(87) 1100087778366101931
fibonacci(88) 1779979416004714189
fibonacci(89) 2880067194370816120
fibonacci(90) 4660046610375530309
fibonacci(91) 7540113804746346429
fibonacci(92) 12200160415121876738
fibonacci(93) 19740274219868223167
fibonacci(94) 31940434634990099905
fibonacci(95) 51680708854858323072
fibonacci(96) 83621143489848422977
fibonacci(97) 135301852344706746049
fibonacci(98) 218922995834555169026
fibonacci(99) 354224848179261915075
Time:  0.0161218643188476556
```

Testing with fibRecursion(n):

Limit: n = 37 (I think it can be larger, but I am afraid that my laptop can not bear of it and crash so I stop at that point).

From n = 25 and especially from 37 onwards : (really slow and it take surplus amount of time to show the final result. Sometimes, I also saw the whole program stuck in a long just for figuring out the next fib number).



And the time it takes apparently longer than Iterative function!

Factorial implementation:

factorial.py fibonacci.py

```
 1 import time
 2
 3
 4 def facIterative(n):
 5     factorial = 1
 6     for i in range(n,1,-1):
 7         factorial = factorial * i
 8     return factorial
 9
10 def facRecursion(n):
11     factorial = 1
12     if n == 0:
13         return 1
14     else:
15         factorial = n * facRecursion(n-1)
16         return factorial
17
18 start = time.time()
19 for i in range(10000):
20     print("factorial(" +str(i) +")",facIterative(i))
21     #print("factorial(" +str(i) +")",facRecursion(i))
22
23 print("Time: ", time.time() - start)
24
25
26
~
~
```

Testing with facIterative(n):

The program continue running over 1000 but it takes a bit longer.

```
62642327153054308892163846110691356624323910437259988058816630549130919816338420
06354699525518784828195856033032645477338126512662942408363494651203239333321502
11425281141171314884337059480114577757503563031288598977986388832075922488212714
15443662515039749101007216506738103035770746401541128333930472760257998112245715
34249672518380758145683914398263952929391318702517417558325636082722982882372594
81658248682672861463319972621127307277513132522224010014095284257249080182299422
40699716135346034878749968524986235843831060145338306500224110536685081655478389
62087111297947300444414551980512439088964301520461155436870989509667681805149977
99304444413842858206514278735645552868111439268095081541820807239353261612233943
44370344242878421193160588811298874742399923365567643379685380368619499188470097
63612475872782742568849805927378373244946190707168428807837146267156243185213724
36454670110055771452046233508408217643117334692933039407147607181359875958881895
43123942343313277002244550158717754761003716150319409450987888948288126484263657
76746774528000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
factorial(999) 40238726007709377354370243392300398571937486421071463254379991042
99385123986290205920442084869694048004799886101971960586316668729948085589013238
29669944590997424504087073759918823627727188732519779505950995276120874975462497
04360141827809464649629105639388743788648733711918104582578364784997701247663288
98359557354325131853239584630755574091142624174743493475534286465766116677973966
68820291207379143853719588249808126867838374559731746136085379534524221586593201
92809087829730843139284440328123155861103697680135730421616874760967587134831202
54785893207671691324484262361314125087802080002616831510273418279777047846358681
70164365024153691398281264810213092761244896359928705114964975419909342221566832
57208082133318611681155361583654698404670897560290095053761647584772842188967964
62449451607653534081989013854424879849599533191017233555566021394503997362807501
37837615307127761926849034352625200015888535147331611702103968175921510907788019
39317811419454525722386554146106289218796022383897147608850627686296714667469756
29112340824392081601537808898939645182632436716167621791689097799119037540312746
22289988005195444414282012187361745992642956581746628302955570299024324153181617
21046583203678690611726015878352075151628422554026517048330422614397428693306169
08979684825901254583271682264580665267699586526822728070757813918581788896522081
64348344825993266043367660176999612831860788386150279465955131156552036093988180
61213855860030143569452722420634463179746059468257310379008402443243846565724501
44028218852524709351906209290231364932734975655139587205596542287497740114133469
62715422845862377387538230483865688976461927383814900140767310446640259899490222
22176590433990188601856652648506179970235619389701786004081188972991831102117122
98459016419210688843871218556461249607987229085192968193723886426148396573822911
23125024186649353143970137428531926649875337218940694281434118520158014123344828
01505139969429015348307764456909907315243327828826986460278986432113908350621709
50025973898635542771967428222487575867657523442202075736305694988250879689281627
53848863396909959826280956121450994871701244516461260379029309120889086942028510
64018215439945715680594187274899809425474217358240106367740459574178516082923013
53580818400969963725242305608559037006242712434169090041536901059339838357779394
10970027753472000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000
Time:  0.47996020317077637
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$
```

Testing with facRecursion(n):

n = 999 is the limit

```
86386691007675071877807576966750324364824122374305711531224079319276711080348987
20665674488451096226601883313676719178695741224608735431894488255708886136246309
50088685237388796652728275164891221521280993431516402364395680979097740600824173
69088000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000
factorial(997) 40359724461645390234292652765290740211090335246139389143030797373
51966319010684237263758833853587102137006631984661977193944113181265519616824478
08198924968051643330792790545975658652366984953410102994729193397927862707860663
31221142813965733919921028883982924596589308458677218884794980135443761645075224
50666655988980094175577967376951675213432494794136314145342021847264214793926157
30781173164526393982880263279118925406206180689438683308644696334133955184235540
07724245116590381101827719832180031595827989994138156615149091737998105454985248
32232927524389811980802708882543991975745364605704734303715958724031694867571661
54294941258045311241382930836862005052391967478429035362983199050663230586866257
61240280494240344233166394434168335073220412356534986944621623211159899567872446
21825685011317463838577067904001075072667390026316129311241122279096729357421049
68533278074796000335855930432060517447195226436187301231195091058916141500005034
48656884759964900494067769318523221837865944485464570390882493401514455003570460
53179773786203118550953567694888922171302000112504911516415310901200837651592219
69755314437880209281708574493693840125338722070514029362985801732618715060934298
23657909616709585950405331060872571119845720022654435044594115773486340742853243
11264856866787884661486819750191740104532976390040068195207044638407735286912244
55265229985489764356909675383800245969276679872407757924211918488179598530382266
64754790722616547980297654789693965688881325682653906791569527887851625739692098
35113890295631011123253723954647397831433613628798725785501475711681360833913542
42735142803988735616917749898060073075542403509536490539404444972668319521415425
66791832347367596656633239099325959195904942407038086186468220698646372928155733
87474665466278592062875719964916067979790641428194695892008126790265612881240871
36359830959867034513441434850212864818601504529520195828528045600869420646442863
72048541496836531269052383502650854577265971210516113769359526291937135884001947
33838020283445311816794177165630135012424772911390424228141663696011522232935969
57527530934652046662174154235850073391729650007182794396630407081318880947107940
24503677464985742937922077663735689021159654000934909225598804790941759477837570
57238419181676630262770090339396547856717150451221853157302493936160447379021701
16980736000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000
Traceback (most recent call last):
  File "factorial.py", line 21, in <module>
    print("factorial(" +str(i) +")",facRecursion(i))
  File "factorial.py", line 15, in facRecursion
    factorial = n * facRecursion(n-1)
  File "factorial.py", line 15, in facRecursion
    factorial = n * facRecursion(n-1)
  File "factorial.py", line 15, in facRecursion
    factorial = n * facRecursion(n-1)
  [Previous line repeated 995 more times]
  File "factorial.py", line 12, in facRecursion
    if n == 0:
RecursionError: maximum recursion depth exceeded in comparison
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$
```

SUMMARY:

	Recursion	**Loop**
Memory	Each recursion step requires *more memory*. Assuming our function is complex and have a lot of variables. Each time when we try to call recursion, the entire function will be store in stack.	Each loop does not require much memory. Simply because, when I try to call the function and finish it, all things related to function's memory will be release (that is the reason why my laptop did not lag why I try to call a iterative loop ☺)

ACTIVITY 2: GREATEST COMMON DEMONINATOR

```
 1 def gcdIterative(a,b):
 2     while b != 0:
 3         if a == 1 or b == 1:
 4             return 1
 5         else:
 6             a,b = b, a%b
 7     return a
 8
 9
10 def gcdRecursion(a,b):
11     if b == 0:
12         return a
13     if a == 1 or b == 1:
14         return 1
15     return gcdRecursion(b, a%b)
16
17 print(gcdIterative(12,3))
18 print(gcdIterative(9,27))
19 print(gcdIterative(1,23))
20 print()
21 print(gcdRecursion(12,3))
22 print(gcdRecursion(9,27))
23 print(gcdRecursion(1,23))
```

I did try with the large number with gcdRecursion function but it still work well. So I automatically set my limit at [-1000,1000] in order to save time and computer memory.

ACTIVITY 3: NUMBER CONVERSIONS

numConvert.py gcd.py

```
 1 def numConvert(n, base):
 2     if n == 0:
 3         return
 4     else:
 5         numConvert(n//base, base)
 6         return print(n % base, end="")
 7
 8
 9 print(numConvert(5,2))
10
11
```

Similarly, to gcd.py, I also tried with large numbers with gcdRecursion function but it still works well.

Because the function is designed to convert decimal number to any base up to 16 ➔ base limit <= 16.

ACTIVITY 4: WRAPPERS AND EXCEPTIONS

Update Fibonacci Recursion by exceptions:

factorial.py **fibonacci.py**

```
 1 import time
 2
 3 def fibIterative(n):
 4     fibVal = 0
 5     currVal = 1
 6     lastVal = 0
 7
 8     if (n == 0 ):
 9         fibVal = 0
10     elif (n == 1):
11         fibVal = 1
12     else:
13         for i in range (n):
14             fibVal = currVal + lastVal
15             lastVal = currVal
16             currVal = fibVal
17
18     return fibVal
19
20 def fibRecursion(n):
21     fibVal = 0
22
23     if (n == 0):
24         fibVal = 0
25     elif ( n == 1):
26         fibVal = 1
27     else:
28         fibVal = fibRecursion(n - 1) + fibRecursion(n - 2)
29     return fibVal
30
31
32 start = time.time()
33 for i in range(1,1000):
34     try:
35         print("fibonacci(" + str(i) + ")",fibRecursion(i))
36     except RecursionError:
37         print("Fibonacci Recursion limit at i = ", i)
38         break
39
40 print("Time: ", time.time() - start)
41
42
43
~
~
~
```

Update Factorial Recursion by exceptions:

factorial.py + fibonacci.py

```
 1 import time
 2
 3
 4 def facIterative(n):
 5     factorial = 1
 6     for i in range(n,1,-1):
 7         factorial = factorial * i
 8     return factorial
 9
10 def facRecursion(n):
11     factorial = 1
12     if n == 0:
13         return 1
14     else:
15         factorial = n * facRecursion(n-1)
16         return factorial
17
18 start = time.time()
19 for i in range(1,1000):
20     try:
21         print("factorial(" +str(i) +")",facRecursion(i))
22     except RecursionError:
23         print("Factorial Recursion limit at i = ", i)
24         break
25
26 print("Time: ", time.time() - start)
27
28
29
~
~
~
~
~
~
~
```

Update Greatest Common Demoninator Recursion by wrappers:

numConvert.py **gcd.py** fibonacci.py

```
 1
 2 def gcdIterative(a,b):
 3     while b != 0:
 4         if a == 1 or b == 1:
 5             return 1
 6         else:
 7             a,b = b, a%b
 8     return a
 9
10 def wrapper(a,b):
11      if ( a < -1000 or a > 1000) or ( b < -1000 or b > 1000):
12          print("STOP entering too big number, my computer will be crashed!")
13      else:
14          result = _gcdRecursion(a,b)
15          return result
16
17
18 def _gcdRecursion(a,b):
19     if b == 0:
20         return a
21     if a == 1 or b == 1:
22         return 1
23     return _gcdRecursion(b, a%b)
24
25 a = int(input("Enter a: "))
26 b = int(input("Enter b: "))
27
28 print(wrapper(a,b))
~
~
~
~
~
~
~
~
```

Update Number Conversions Recursion by try and exceptions:

numConvert.py + gcd.py fibonacci.py

```
 1
 2 def numConvert(n, base):
 3     raise ValueError("Is your base = " + str(base) + "<= 16 ?")
 4     if n <= 1:
 5         return n
 6     else:
 7         numConvert(n//base, base)
 8         return print(n % base, end = "")
 9
10
11 n = int(input("Enter number you want to convert: "))
12
13 try:
14     base = int(input("Enter base: "))
15     result = numConvert(n,base)
16
17 except ValueError as err:
18     print(err)
19
20 else:
21     print(result)
22
23 finally:
24     print("Nice! Converting finished")
25
26
27
28
```

ACTIVITY 5: TOWERS OF HANOI IMPLEMENTATION

```python
def moveDisk(src,dest):
    print("Moving top disk from tower %s to tower %s" %(src,dest))


def towers(n, src, dest, space):

    tmp = 6 - src - dest
    print("%s tower(%s,%s,%s)" %(space, n, src, dest))
    print("%s    n = %s, src = %s, dest = %s, tmp = %s" %(space, n, src, dest, tmp))

    if n == 1:
        moveDisk(src, dest)
    else:
        towers(n-1, src, dest, space+"    ")
        moveDisk(src, dest)
        towers(n - 1, tmp, dest, space+"    ")


towers(5,1,3, "")
```

Output:

```
ccadmin@CCUbuntu64bit:~/DSA1002/Prac2$ python3 towerOfHanoi.py
 tower(5,1,3)
    n = 5, src = 1, dest = 3, tmp = 2
     tower(4,1,3)
        n = 4, src = 1, dest = 3, tmp = 2
         tower(3,1,3)
            n = 3, src = 1, dest = 3, tmp = 2
             tower(2,1,3)
                n = 2, src = 1, dest = 3, tmp = 2
                 tower(1,1,3)
                    n = 1, src = 1, dest = 3, tmp = 2
Moving top disk from tower 1 to tower 3
Moving top disk from tower 1 to tower 3
                 tower(1,2,3)
                    n = 1, src = 2, dest = 3, tmp = 1
Moving top disk from tower 2 to tower 3
Moving top disk from tower 1 to tower 3
             tower(2,2,3)
                n = 2, src = 2, dest = 3, tmp = 1
                 tower(1,2,3)
                    n = 1, src = 2, dest = 3, tmp = 1
Moving top disk from tower 2 to tower 3
Moving top disk from tower 2 to tower 3
                 tower(1,1,3)
                    n = 1, src = 1, dest = 3, tmp = 2
Moving top disk from tower 1 to tower 3
Moving top disk from tower 1 to tower 3
         tower(3,2,3)
            n = 3, src = 2, dest = 3, tmp = 1
             tower(2,2,3)
                n = 2, src = 2, dest = 3, tmp = 1
                 tower(1,2,3)
                    n = 1, src = 2, dest = 3, tmp = 1
Moving top disk from tower 2 to tower 3
Moving top disk from tower 2 to tower 3
                 tower(1,1,3)
                    n = 1, src = 1, dest = 3, tmp = 2
Moving top disk from tower 1 to tower 3
Moving top disk from tower 2 to tower 3
             tower(2,1,3)
                n = 2, src = 1, dest = 3, tmp = 2
                 tower(1,1,3)
                    n = 1, src = 1, dest = 3, tmp = 2
Moving top disk from tower 1 to tower 3
Moving top disk from tower 1 to tower 3
                 tower(1,2,3)
                    n = 1, src = 2, dest = 3, tmp = 1
Moving top disk from tower 2 to tower 3
Moving top disk from tower 1 to tower 3
     tower(4,2,3)
        n = 4, src = 2, dest = 3, tmp = 1
         tower(3,2,3)
            n = 3, src = 2, dest = 3, tmp = 1
             tower(2,2,3)
```