Barabasi with degree 2:

In [8]:
```
import random  #libraries for random number
import numpy as np #for numpy array
import pandas as pd
import matplotlib.pyplot as plt
import statistics as st
import numpy as np
from collections import Counter
import random
import operator
import math
import networkx as nx
import statistics as st
```

In [9]:
```
n_init=5
def init_graph():
  matrix=[]
  n=5
  e=10
  for i in range(20):
    temp=[]
    for j in range(20):
     temp.append(0)
    matrix.append(temp)
  # print(matrix)
  while e>=0:
    print(e)
    for i in range(n):
      r=random.randint(0,n-1)
      # print(r,"and",i)
      if r != i and matrix[i][r]!=1:
        matrix[r][i]=1
        matrix[i][r]=1
        e=e-1
  return matrix
```

In [ ]:
```
iter=100
n=5
avg_cl=[]
avg_pl=[]
for itr in range(iter):
  matrix=init_graph() #create inital graph randomly
  degree={}#degree of each node take out
  sum_of_all_degree=0
  for i in range(n):#loop for degree of each node
    degree[i]=sum(matrix[i])
    sum_of_all_degree+=(degree[i]*degree[i])
  prob=[]#create a probailty list ki/sum degree
  for i in degree.keys():
    prob.append((degree[i]*degree[i])/sum_of_all_degree) #ki/sum of all degree
for prob    #que3: degree[i]2/sum_all_degree2
  cummulative=[]#cumulative list
  cumsum=0.0
  for i in range(len(prob)):
    cumsum+=prob[i]
    cummulative.append(cumsum)#cumulative list
  for j in range(n,20):
     row_col=[0]*len(matrix[0])
     column_to_be_added =row_col
     matrix = np.column_stack((matrix, column_to_be_added))
     row_to_be_added =row_col
     matrix = np.vstack((matrix,row_to_be_added))
     e1=3
     while e1>0:
       rnd=random.uniform(0.0,1.0)
       index=0
       if rnd<=cummulative[0]:
         index=0
       for c in range(0,len(cummulative)-1):
         if cummulative[c]<rnd and cummulative[c+1]>rnd:
           index=c+1
       if index!=j:
        matrix[index][j]=1
        matrix[j][index]=1
        e1-=1
  overall_100_graphs_degree_distribution={} #it is grphwise degree for all 1o0
graphs and vales are the unique degree counts of each node of the random 100 g
raphs
  for i in range(1,101):
    overall_100_graphs_degree_distribution[i]={}
  unique_degrees={} #here it is dictionary for the unqiue degree
  ##intitialise:graph
  # adjancency_matrix= [[0 for i in range(len(n))] for j in range(len(n))]
  # edge list intialise
  edge_list={}    #edge list for each new individual  random  graph generate
  for i in range(len(n)):   #initialization of edge list
    edge_list[i]=[]
  for c in range(len(n)):  #here we make 2 loops for creating edges if possibl
e between 2 node
    for r in range(c+1,len(n)):
        x=random.uniform(0,1)  #random unform probabilty is generated and chec
k with p if p>x then edge is possbile else not
```

```
        if p>x:
          # adjancency_matrix[r][c]=1
          # adjancency_matrix[c][r]=1
          edge_list[r].append(c)   #make undirected graph
          edge_list[c].append(r)

  degree_of_nodes={}    #now here we find out all possible degree of nodes [pr
esent in generated graph]
  for i in edge_list.keys():
    degree_of_nodes[i]=0
  for i in edge_list.keys():    #degeree is total adjancent nodes present arou
nd partricular nodes
    degree_of_nodes[i]=len(edge_list[i])



  degree_frequncy={}   #frequncy count of the occurence of the degree of the n
odes
  for key,value in degree_of_nodes.items():
    if value not in degree_frequncy.keys():
      degree_frequncy[value]=0
      degree_frequncy[value]+=1
    else:
      degree_frequncy[value]+=1
  for i in degree_frequncy.keys():
    if i not in unique_degrees.keys():
      unique_degrees[i]=0

  overall_100_graphs_degree_distribution[count]=degree_frequncy   #store frqun
cecy
```

In [4]:

```
print("Avg cluster coeff")
for i in avg_cl:
  print(i)

print("  ")
print("  ")
print("Avg path length")
for i in avg_pl:
  print(isinstance)
```

```
Avg cluster coeff
0.2817388414904154
0.17204435068412205
0.22402562543815863
0.3192325073548348
0.1752638399293466
0.11292995062035388
0.24671054700652842
0.2648068035292921
0.25485506791304724
0.21888865309261762
0.23349892550729215
0.0752305730058931
0.22575925643796896
0.17468777225620563
0.17093283427770475
0.1864884087697623
0.1892437427504664
0.17523070631808504
0.253581269946104
0.2744920741888978
0.27640448351925323
0.1870630665723137
0.15388094146123485
0.361336665977901
0.14327956619028764
0.23132769434659384
0.2838254068109355
0.240822735208724
0.13209053862932388
0.2656813106802698
0.3013016197211106
0.34179841812515993
0.1940340033888733
0.15869982499040602
0.18437297437922287
0.22250258776656612
0.10683270199720939
0.2049240172691871
0.19269907170045567
0.22797093444069866
0.3283514324754082
0.1619736918467123
0.1808569303625836
0.14444636930945254
0.145663877489675
0.22632191935728205
0.18894650366102272
0.17962419068470356
0.20815704514034564
0.1565098768664189
0.20221976534772676
0.12791989592901745
0.2492608556613363
0.46111369827789167
0.30470259771294655
0.40244970448536466
```

```
0.17910597333262845
0.18769841638911663
0.322541731053961
0.24136008969445494
0.19138664666889973
0.1955380602917161
0.2811996723581505
0.19058832244062052
0.350685281002785
0.10746330667654329
0.13074583919961139
0.2718647837082416
0.37032902340649004
0.1434449723063458
0.28845487808312725
0.3274395607626364
0.28022961103688776
0.2089691480612533
0.18590372996225224
0.14958630306749404
0.28908254025335967
0.24138590860961118
0.28405621520841084
0.19646410434942954
0.19013398578822238
0.33542778325307604
0.3045341722811665
0.17894184341080938
0.3522933881354717
0.31701307751868335
0.18666267856831723
0.16162095168522905
0.2719861213028172
0.24567733835659278
0.1825533292372987
0.09850288254710743
0.2128216042993431
0.15863789315816473
0.19692640869202666
0.19950972755161764
0.23070762125702335
0.22903018519684298
0.18667197453056708
0.2810393479120715


Avg path length
2.3256565656565655
2.386060606060606
2.325050505050505
2.451919191919192
2.366868686868687
2.403636363636364
2.336969696969697
2.2543434343434345
2.3913131313131313
2.3638383838383836
```

```
2.3123232323232323
2.4216161616161616
2.3414141414141416
2.321010101010101
2.3507070707070707
2.3608080808080807
2.387272727272727
2.4327272727272726
2.2903030303030305
2.3159595959595958
2.357979797979798
2.3953535353535353
2.4353535353535354
2.4143434343434342
2.3624242424242423
2.4284848484848487
2.402828282828283
2.321010101010101
2.4775757575757575
2.2664646464646463
2.435151515151515
2.335757575757576
2.249090909090909
2.421818181818182
2.382020202020202
2.394343434343434
2.4705050505050505
2.4464646464646465
2.48989898989899
2.3626262626262626
2.3234343434343434
2.3323232323232324
2.4064646464646464
2.323232323232323
2.454141414141414
2.5468686868686867
2.418989898989899
2.494747474747475
2.391717171717172
2.3496969696969696
2.3464646464646464
2.411919191919192
2.4026262626262627
2.416969696969697
2.278989898989899
2.431111111111111
2.410707070707071
2.412121212121212
2.4517171717171715
2.3686868686868685
2.3925252525252527
2.394343434343434
2.3266666666666667
2.3327272727272725
2.2955555555555556
2.3913131313131313
2.359191919191919
```

```
2.2953535353535353
2.34989898989899
2.346262626262626
2.3064646464646463
2.24989898989899
2.356969696969697
2.44
2.4149494949494947
2.4331313131313133
2.275757575757576
2.514949494949495
2.302020202020202
2.287272727272727
2.433939393939394
2.507878787878788
2.469292929292929
2.3175757575757574
2.3024242424242423
2.3777777777777778
2.3254545454545457
2.381818181818182
2.3923232323232324
2.3076767676767678
2.32989898989899
2.383030303030303
2.365858585858586
2.312121212121212
2.4135353535353534
2.4143434343434342
2.294343434343434
2.32020202020202
2.315151515151515
2.3022222222222224
```

In [ ]:

```
for graph,deg in overall_100_graphs_degree_distribution.items():   #here count
ing frquncy of all the unique degrees in overall random gernerated graph
  # graph 1
  for key,value in deg.items():
    unique_degrees[key]+=value
```

In [ ]:

```
kmax=0
cmax=0
for key,value in unique_degrees.items(): #for finding maximum degree kmax in t
he graph
  if cmax< value:
    kmax=key
    cmax=value
```

In [ ]:

```
for i in unique_degrees.keys():  #take out mean of the degrees
  unique_degrees[i]/=100
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
fg, ax = plt.subplots(figsize =(12, 6))
x_axis=[]
y_axis=[]
for key,value in  unique_degrees.items():
  x_axis.append(key)
  y_axis.append(value/kmax)
  ax.scatter(x_axis,y_axis,s=np.pi*3.2,c=("red"), alpha=0.5)

# plt.xlabel("DEGREE OF NODES", fontsize=12)
# plt.ylabel(" PROBABILITY OF DEGREE (MEAN) ", fontsize=12)
# plt.title("Scaled Degree Distribution(Mean)",fontsize=18)
# plt.show()
```

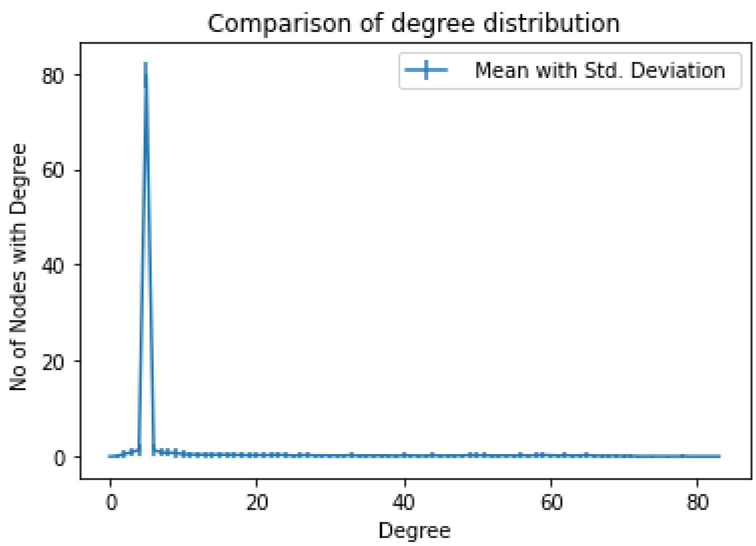
In [ ]:
```
import numpy as np
standard_deviation_overall={} #here we are finding out standard deviation for
 all the degree of the graph using
standard_deviation_N={}  #using formula sqrt((x-mean(x))/N(total number))
for i in unique_degrees.keys():
  standard_deviation_overall[i]=0
  standard_deviation_N[i]=0
for graph,deg in overall_100_graphs_degree_distribution.items(): #lloop for ov
erall degrees of 100 graphs
  for key,value in deg.items():
    standard_deviation_overall[key]+=np.square(value - unique_degrees[key])  #
square(x-mean of x) calculation here
    standard_deviation_N[key]+=1
```

In [ ]:
```
for i in unique_degrees.keys():
  standard_deviation_overall[i]=np.sqrt(standard_deviation_overall[i]/standard
_deviation_N[i]) # sqrt(sqare(x-mean of x)/N) calculation
```

In [ ]:
```
kmax=0
cmax=0
for key,value in standard_deviation_overall.items(): #for finding maximum degr
ee kmax in the graph
  if cmax< value:
    kmax=key
    cmax=value
kmax=max(list(standard_deviation_overall.values()))
```

In [7]:
```
plt.errorbar(x,mean_list, yerr = variance_list, label ='  Mean with Std. Devia
tion ')
plt.title("Comparison of degree distribution")
plt.xlabel('Degree')
plt.ylabel("No of Nodes with Degree ")
plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0x7f2ff5063610>



In [ ]:

In [1]:
```
print("Theory:")
print("1.clustering coefficient increases anmd shortest path length decreses w
ith order because hubs are increaese. ")
print("2.hubs are increases in clustering coefficent as hub is more brcause it
is connected with more number of nodes")
print("3.hub are more because it is increasing exponenetially")
print("4.Because hub increse exponentially the higher probable are having more
probability and lower probable become less")
print("having less and less probabilty")
```

```
Theory:
1.clustering coefficient increases anmd shortest path length decreses with or
der because hubs are increaese.
2.hubs are increases in clustering coefficent as hub is more brcause it is co
nnected with more number of nodes
3.hub are more because it is increasing exponenetially
4.Because hub increse exponentially the higher probable are having more proba
bility and lower probable become less
having less and less probabilty
```

In [2]:
```
print("5.In average path length decreases because for heigher order rich nodes getting more richer concept is applies like more nodes are connected so neighbors getting increases, so when hubs neighbor increase then average shortest path length decrease.")
```

```
5.In average path length decreases because for heigher order rich nodes getting more richer concept is applies like more nodes are connected so neighbors getting increases, so when hubs neighbor increase then average shortest path length decrease.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: