PAKISTAN INSTITUTE OF ENGINEERING AND APPLIED SCIENCES
Department of Computer and Information Sciences (DCIS)

Dated: 11-07-2017

Assignment: Artificial Intelligence

Name: Muhammad Dawood
Registration Number: 30251

Question # 1:- Write an (efficient!) Python program that, given a stick of integer length N, breaks it up into smaller sticks of integer but all un-equal lengths. For example: For N = 10, one possibility is (1, 2, 3, 4). However (1, 1, 1, 3, 4) is not valid due to the repeated 1s.
How many unique ways are there to break up the stick of length N? Please note that permutations of a break-up are not to be counted, i.e., you are to count, (1, 2, 3, 4) and (2, 1, 3, 4) as a single breakup.

1 Solution:

1. Given a stick of size N, break the stick to partition of two number e.g: in N=10, partition is (1,9) and so on add this partition to a list.
2. Divide the list into partition of different length at start we only have partition of length 2 call the cutSub() function for partition of length 2,means list of size 2.
3. The cutSub() function will create another partition by deleting last element of the list and add find two different number greater that number at n-1 location and sum of these two numbers not greater that number at n position of list. In this way find all partition of size 3 and so on, and add them to main list where we have all the combination. Next time main function will call the cutSub() function for partition of length 4 if possible and so on.

2 Program source code:

```
# -*- coding: utf-8 -*-
"""
    Created on Thu Jul  6 10:40:29 2017
    Artificial Intelligence Assignment
    @author: Muhammad Dawood
```

```
    Prgoram Complexity n^k
    Given a stick of length N break this into small stick of different length
"""
import time
import matplotlib.pyplot as plt
import matplotlib.transforms as mtrans

total_partitions=[]
processing_time=[]


def plotResult(X=[],Y=[]):
    """
    This method will plot the result of two array
    X dimension and Y dimension
    """
    fig = plt.figure(figsize=(12, 14))
    ax = plt.subplot(2, 1, 2)
    trans_offset = mtrans.offset_copy(ax.transData, fig=fig,
                                      x=0.03, y=0.4, units='inches')
    plt.plot(stick_length_input,processing_time,'ro-')
    plt.title('Time Complexity VS Input size')
    plt.xlabel('Input size (n)')
    plt.ylabel('Time (t)')
    plt.yscale('log')
    plt.grid()
    for x, y in zip(X, Y):
        plt.plot((x,), (y,), 'ro')
        plt.text(x, y, '(%d, %.4f)' % (int(x), float(y)), transform=trans_offset)
```

```
    plt.show()
    return



def cutSub(sub_partition=[],*args):
    """
        sub_partition[]:- Show partition which can further be divided into sub
partition of higher length
        Deleting last element from the higher partition and append multiple
number whose sum is
        equal to the last element. At the end the modified_partition is added to
        total_partitions list
    """
    first=sub_partition[len(sub_partition)-2]
    last=sub_partition[len(sub_partition)-1]
    modified_partition=sub_partition[:]
    for q in range(first+1,last//2+1):
        if(last-q>q):
            modified_partition=sub_partition[:]
            del modified_partition[len(modified_partition)-1]
            modified_partition.extend([q,last-q])
            total_partitions.append(modified_partition)
    return


def cutRod(stick_length):
    """
        Divide the sub_partition into further partitions
        stick_lenght :- represent length of the stick to be divided into pieces
        total_partitions:- global List of all possible partition
```

```
        partition_lenght:- lenght of partion to be splitted into nested sub
partion
        piece_index:- show index of the selected sub partition
    """


    #recording starting time of program
    start_time=time.clock()


    for i in range(1,stick_length//2+1):
        if(stick_length-i>i):
            total_partitions.append([i,stick_length-i])
    partition_length=2
    piece_index=0
    while(partition_length<stick_length//2):
        for piece in range(piece_index,len(total_partitions)):
            if len(total_partitions[piece])==partition_length:
                cutSub(total_partitions[piece][:])
            else:
                piece_index=piece
                continue
        partition_length=partition_length+1
#    print("Total Number of possible combination ",len(total_partitions))
#    print(time.clock()-start_time," Time consumed in second :: Input
size",stick_length_input)
    processing_time.extend([time.clock()-start_time])
    return



stick_length_input=[10,20,40,80,100,120]
for i in stick_length_input:
```

```
    cutRod(i)
plotResult(stick_length_input,processing_time)
```

3 Time Complexity Graph:

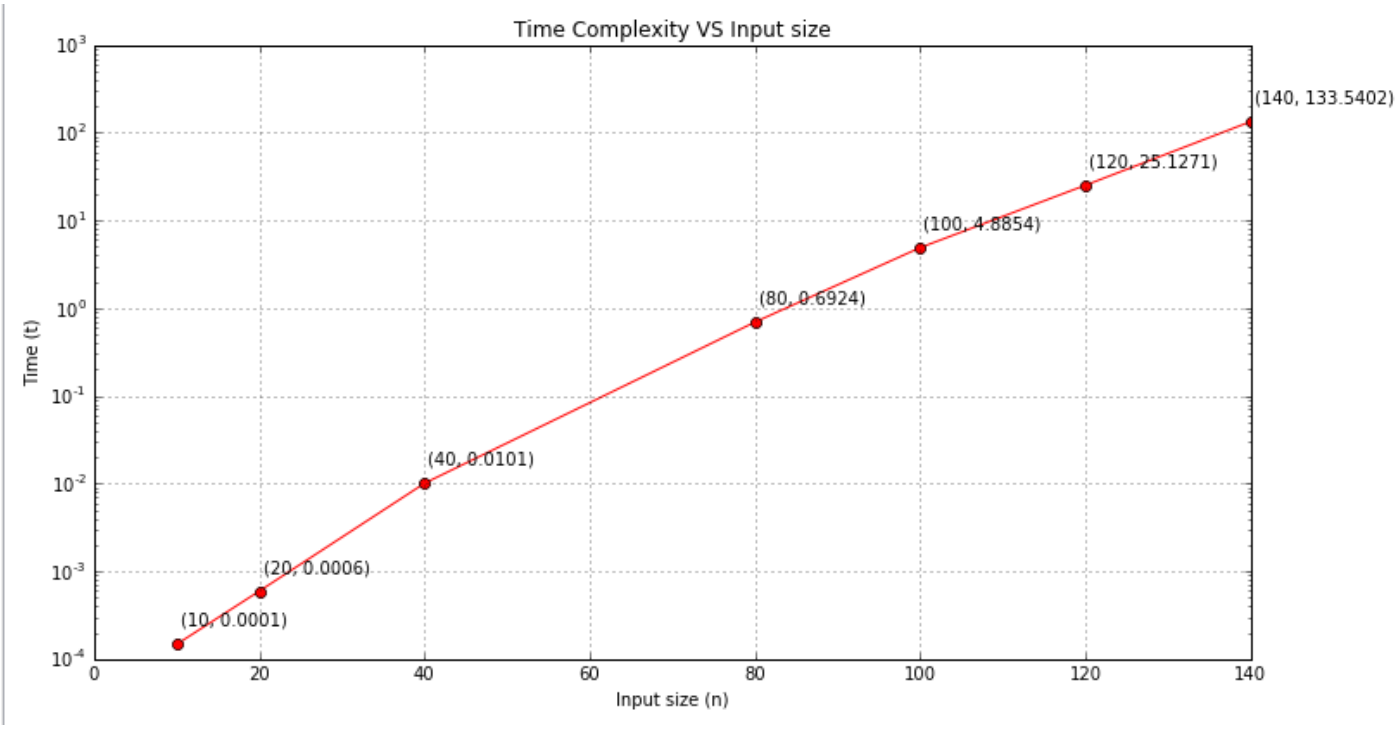

Figure 1 Time Complexity

4 Number of combination vs Input size:

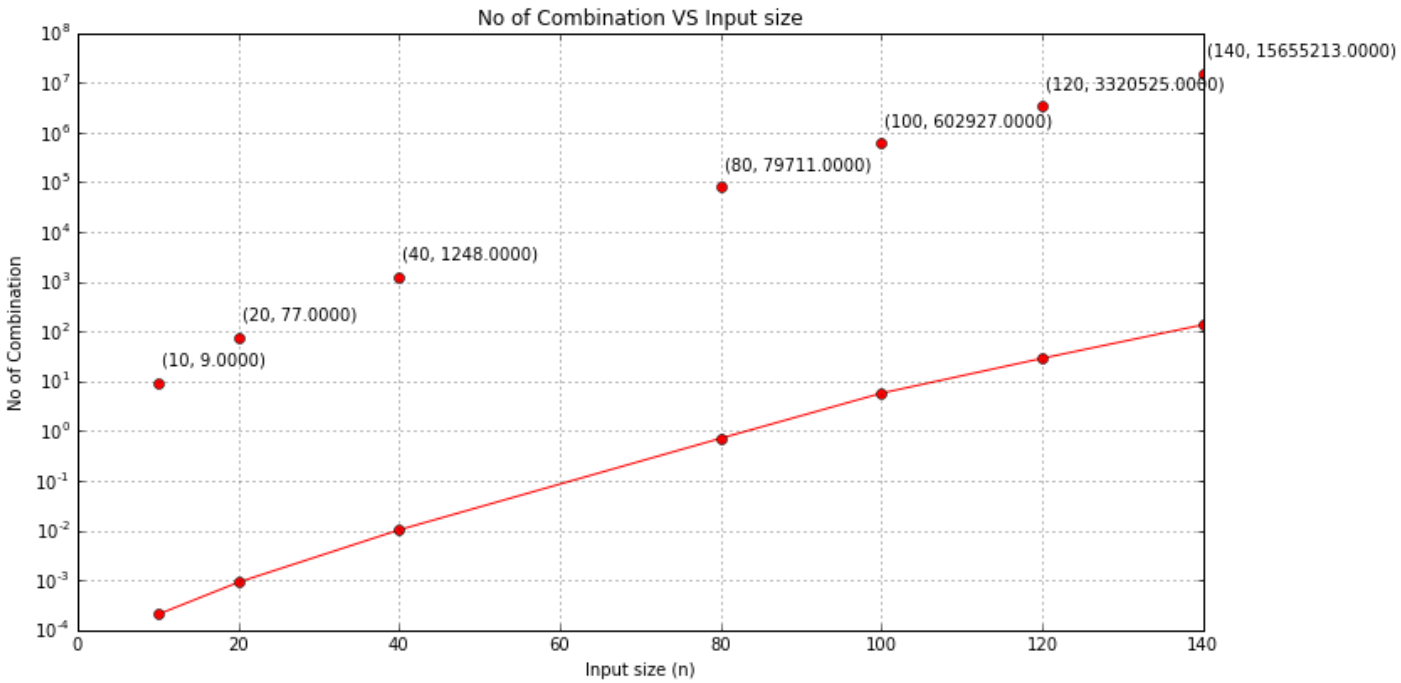

Figure 2 no of combination vs input size

Question # 2:- Write a Python program that, given a square with side length N breaks it into squares of integer but all unequal lengths?

Answer: For this question I write program but I did not get any combination except the number itself. Searching the internet I find the solution given in the Figure 3. The area of square is 12*112 we have the following possible combination each of different area. The smaller square in the figure not numbered is of length 2.The details about the solution in available on the following site:

http://www2.stetson.edu/~efriedma/mathmagic/1298.html

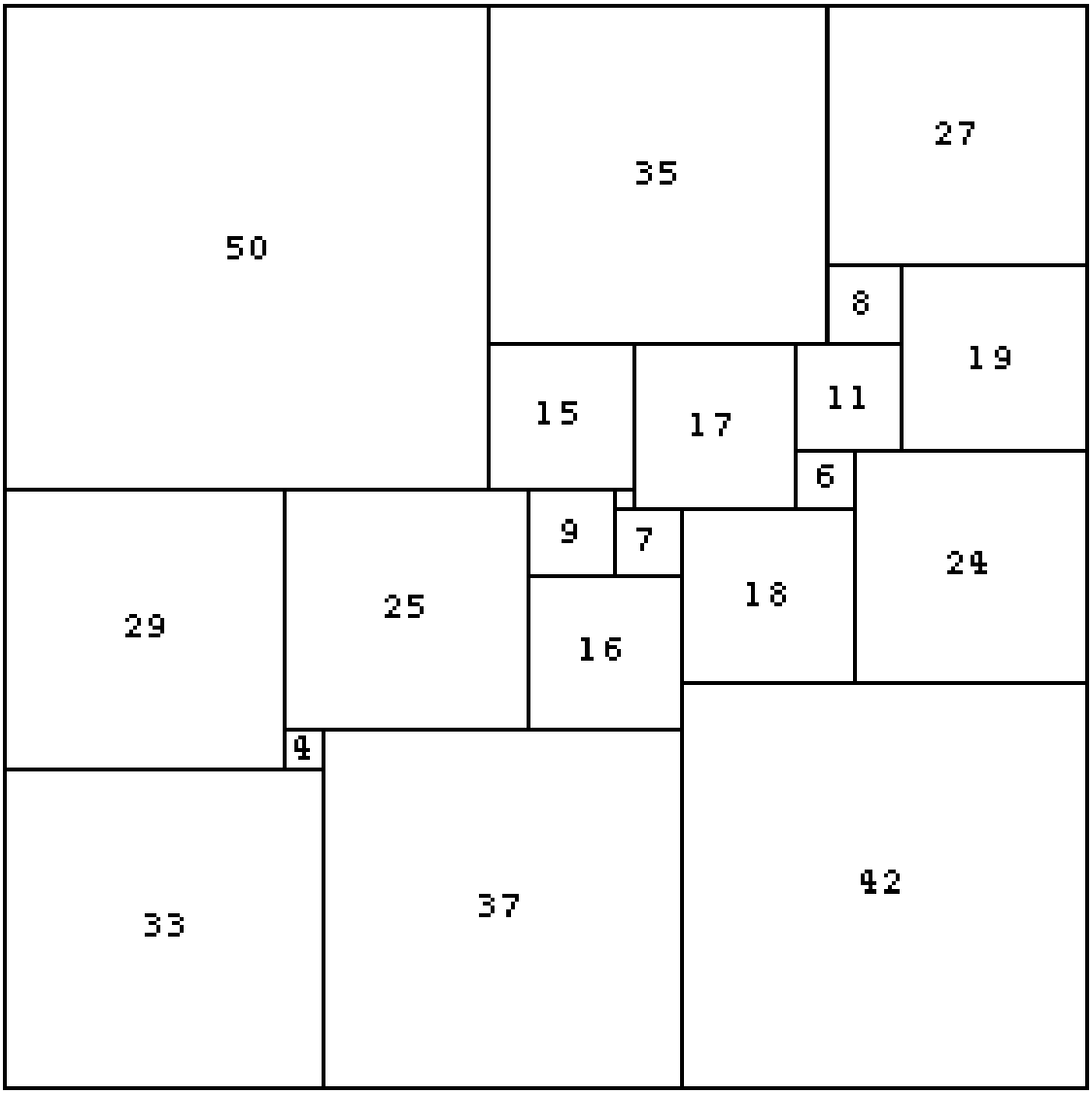

Figure 3 Possible combination for square

I try my best to generalize this solution for square of all sides but I was unable to implement this solution