**Course Name:** 2302 **Author:** Olugbenga Iyiola **ID:** 80638542 **Instructor:** Olac Fuentes **TA:** Nath Anindita/ Malileh Zargaran **LAB #8 Report**

**Introduction**

The purpose of this lab is to use randomized algorithm to discover trigonometric identities. The program tests all combinations of selected trigonometric expressions to detect the equalities that exist between any pair.

The lab also involves using a Backtracking algorithm to determine if there is a way to partition a set of integers S into two subsets S1 and S2 such that P S1 = P S2, where S1 and S2 are a partition of S if and only if S1∪S2 = S and S1∩S2 = {}.

A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits.(Wikipedia 2019)

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).(GeeksforGeeks 2019)

**Proposed Solution Design and Implementation**

The following pseudocode illustrates how to check for equalities of the trigonometric functions:

*for i in range of number of tries*

*randomly generate numbers between -pi and +pi*

*y1= Evaluate the first passed string*

*y2= Evaluate the second passed string*

*if absolute difference(y1-y2)>tolerance level*

*return False*

*Else return True*

The following pseudocode illustrates how to determine if there is a way to partition a set of integers S into two subsets S1 and S2 such that P S1 = P S2:

*Divide set into two subsets*

*Find summation of left list*

*Find summation of right list*

*Join left and right list*

*check if summation of left list = summation of right list*

*return True if true*

*if leftSum > rightSum and List content yet to checked:*

*remove minimum element of the bigger list*

*Append the minimum element of the bigger list to the smaller list*

*Store list that has already been checked*

*Recursive call on SubsetSum(S,leftList,rightList, checkList) # Backtracking*

*return result, leftList, rightList*

*if rightSum > leftSum and List content yet to checked:*

*remove minimum element of the bigger list*

*Append the minimum element of the bigger list to the smaller list*

*Store list that has already been checked*

*Recursive call on SubsetSum(S,leftList,rightList, checkList) # Backtracking*

*return result, leftList, rightList*

*Else return False, leftList, rightList*

**Experimental Result**

System Specification: HP Windows 10, 1.60GHZ Intel® Celeron® , 4.GB RAM, 64-bit operating system

The results of the various test cases using different sizes from the file for each of the algorithms are shown below:

**Comparing Trigonometric Functions**

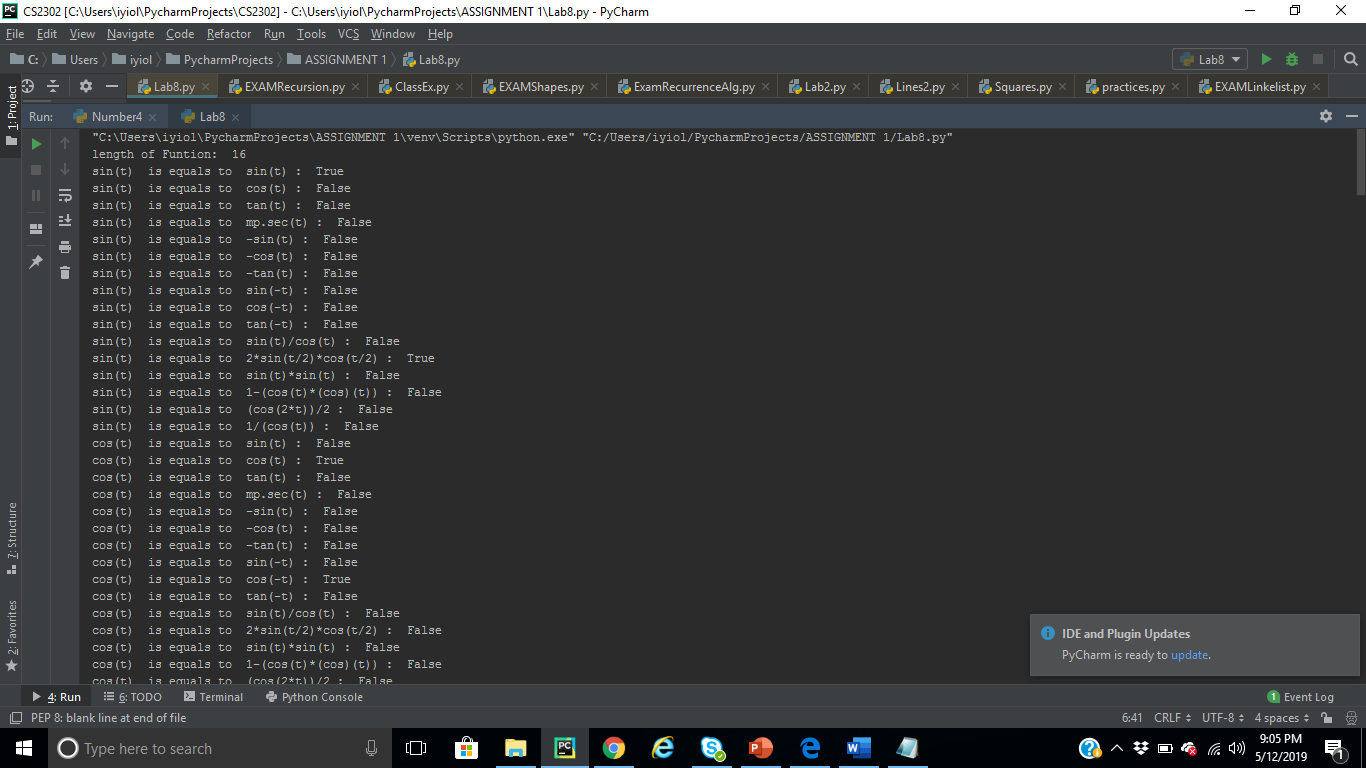
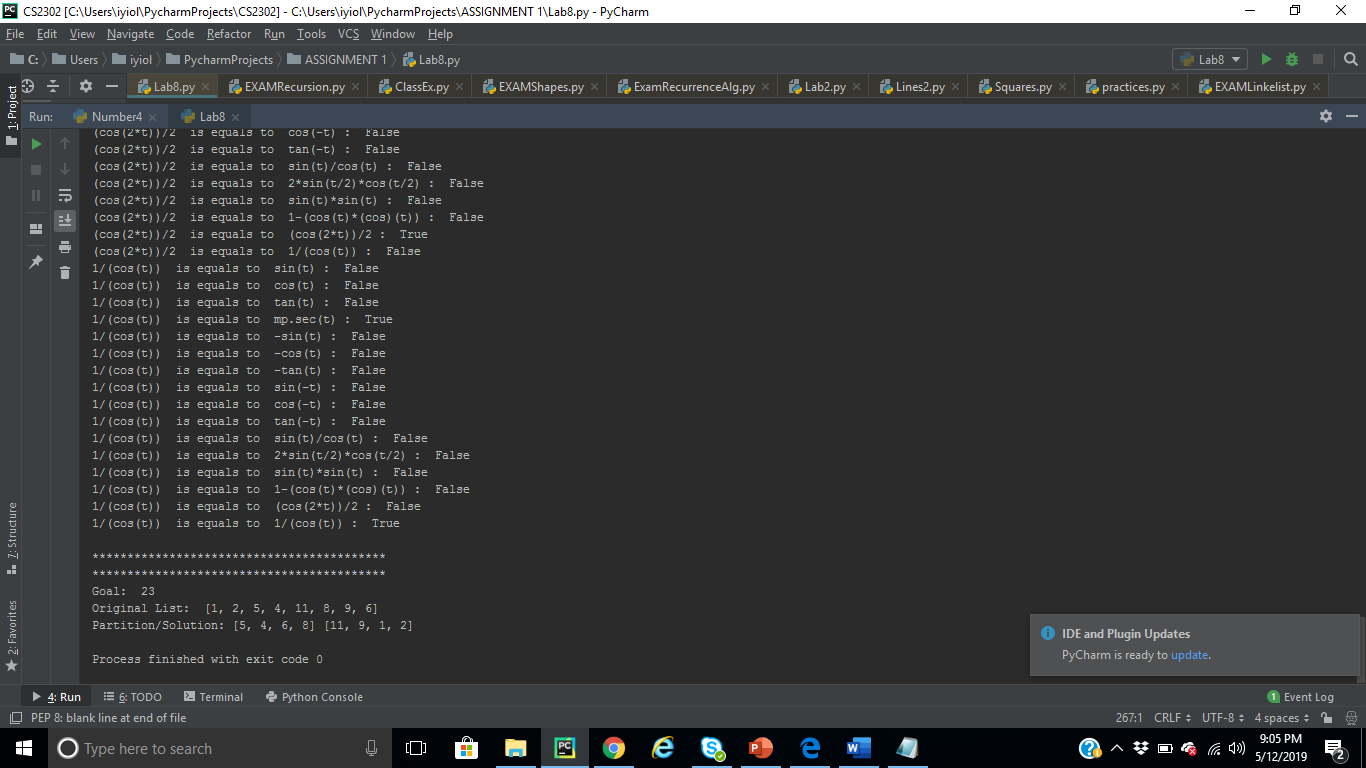
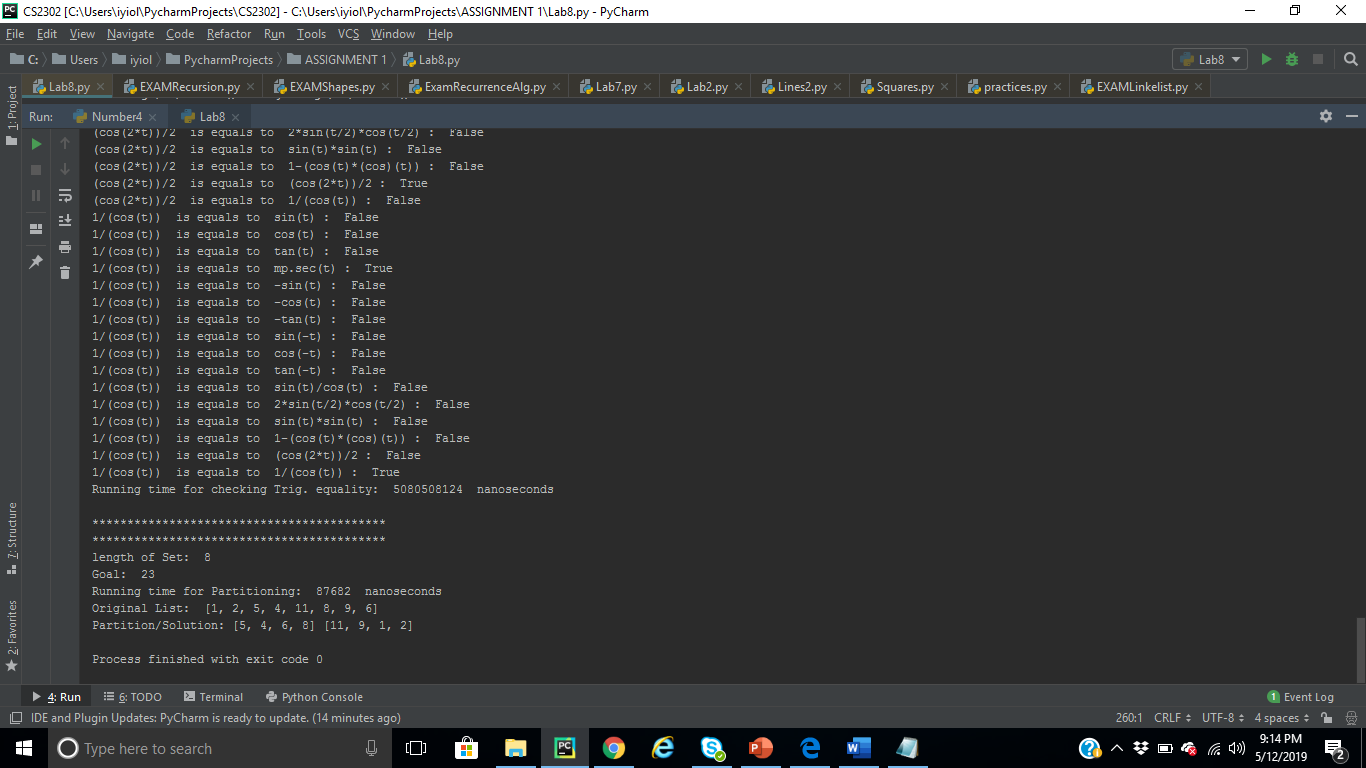
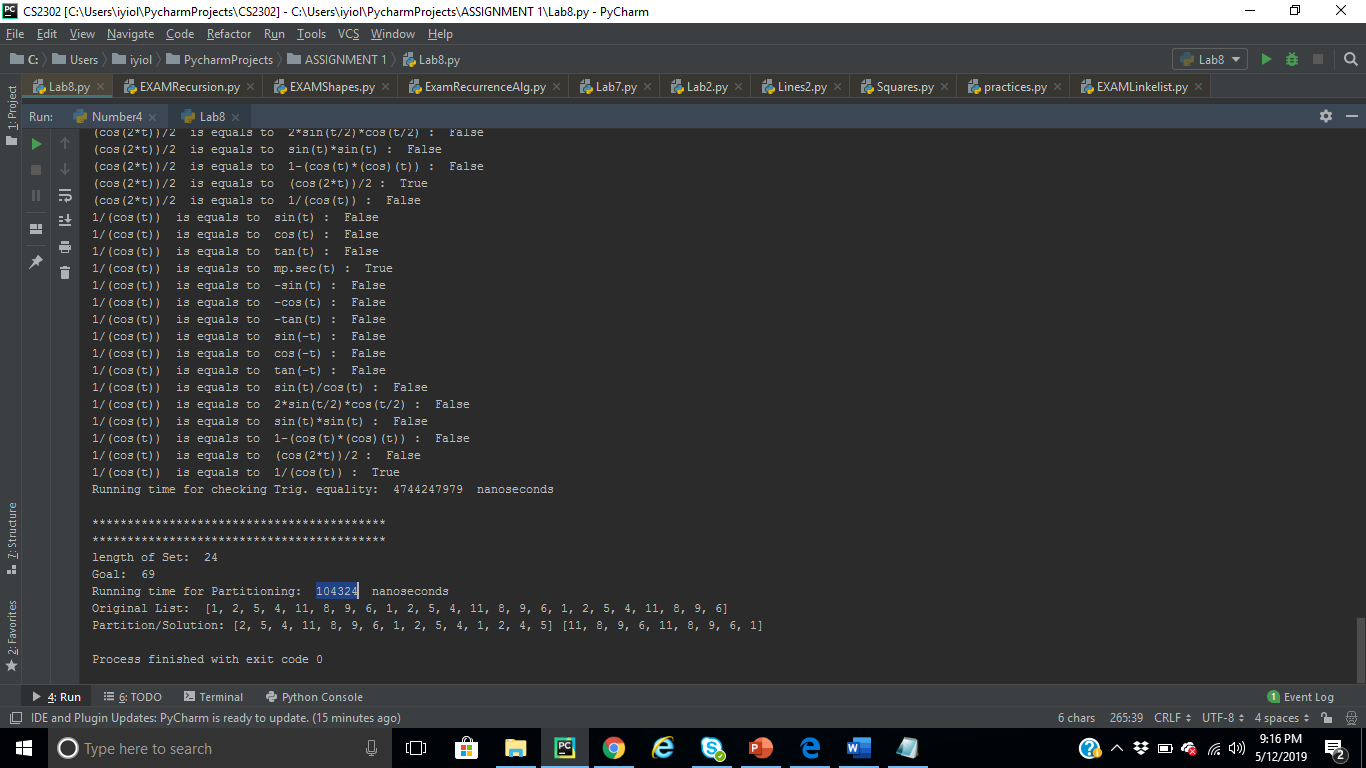
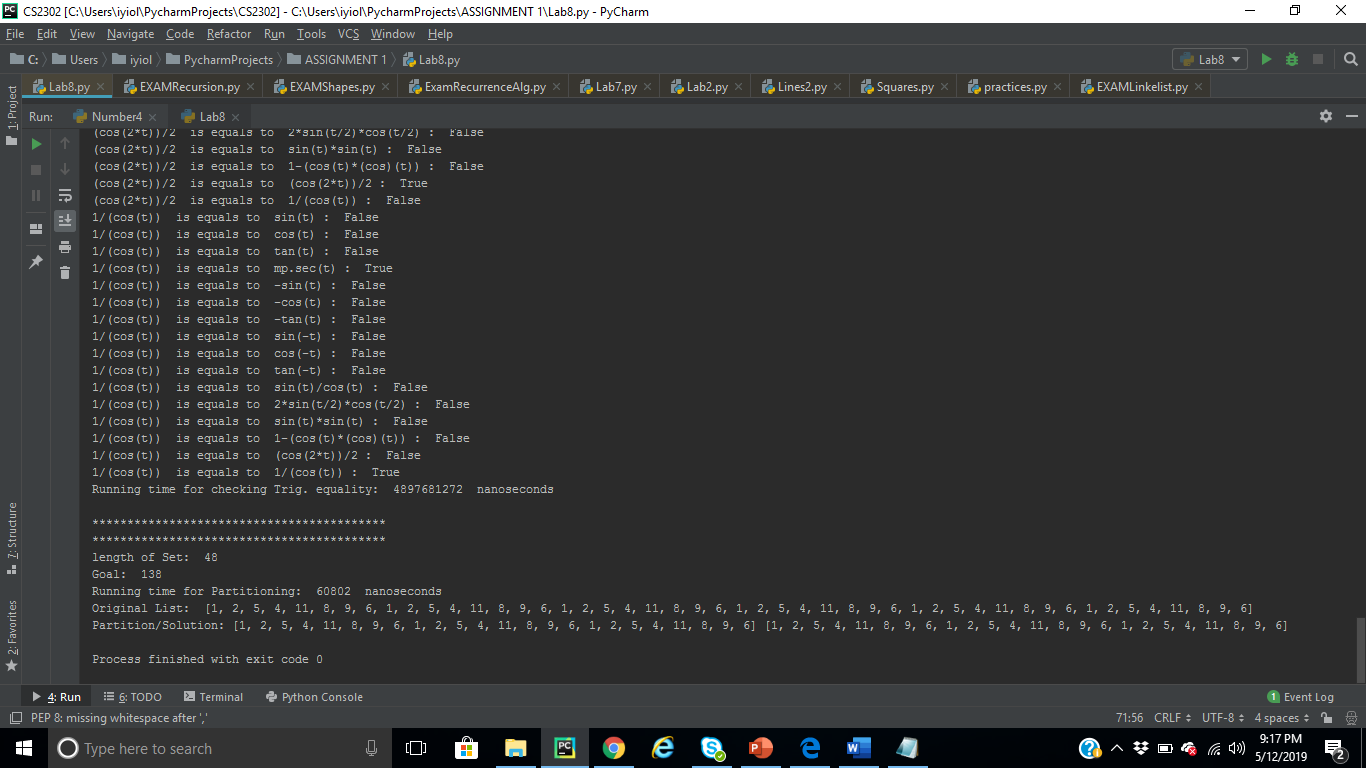
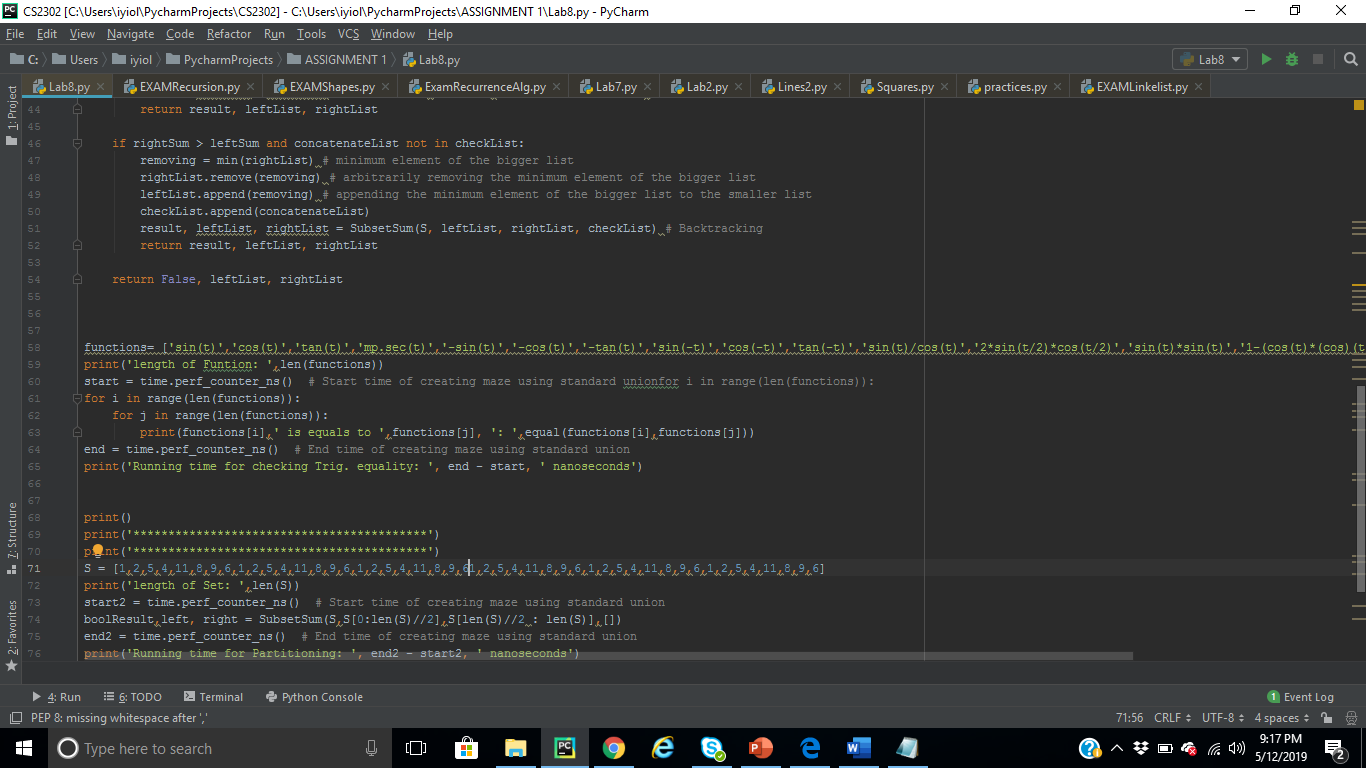
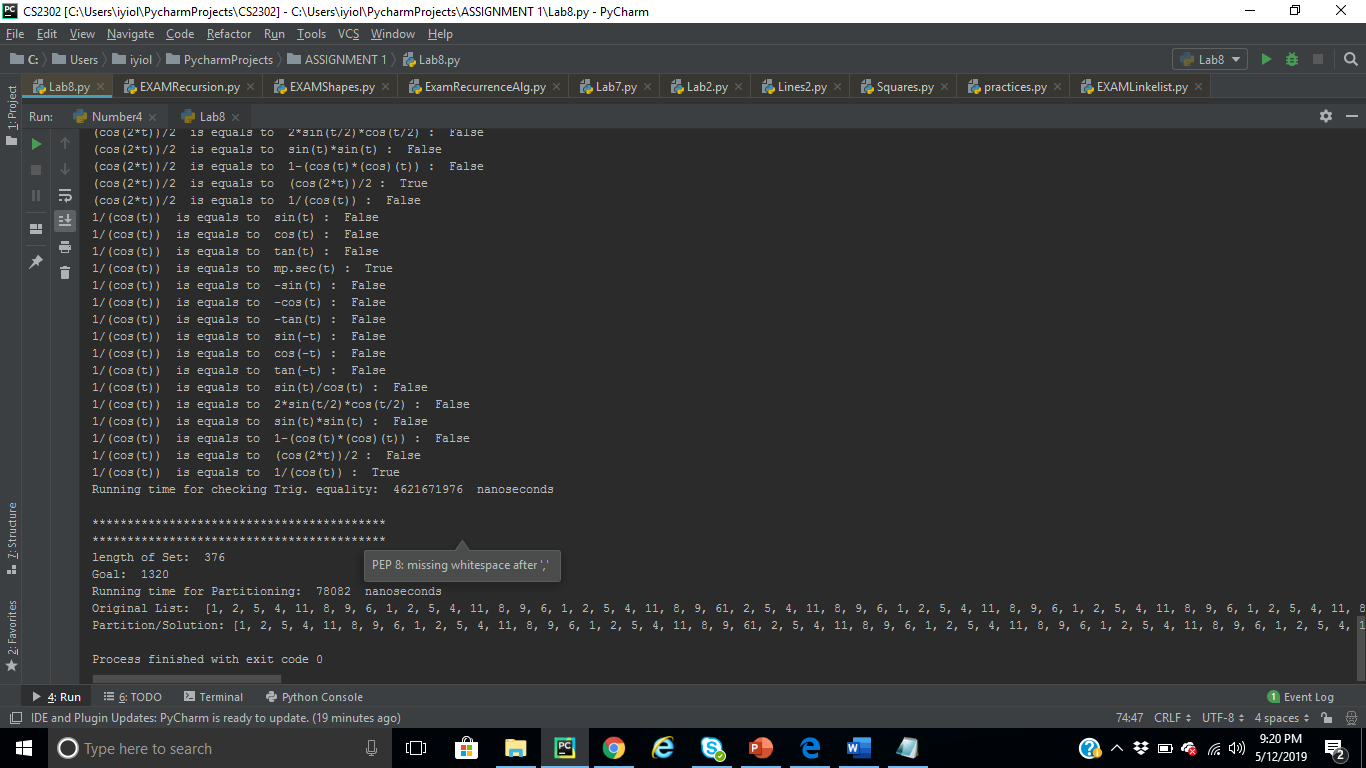
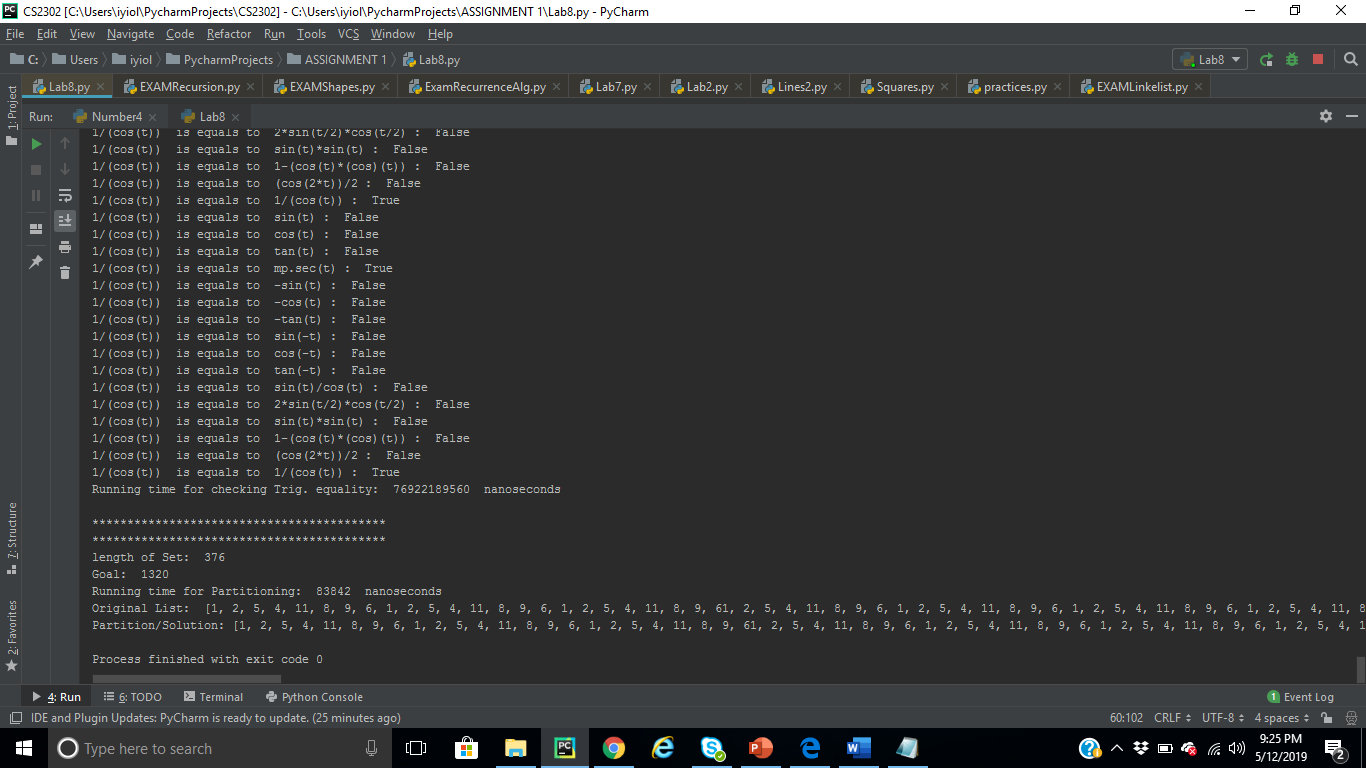
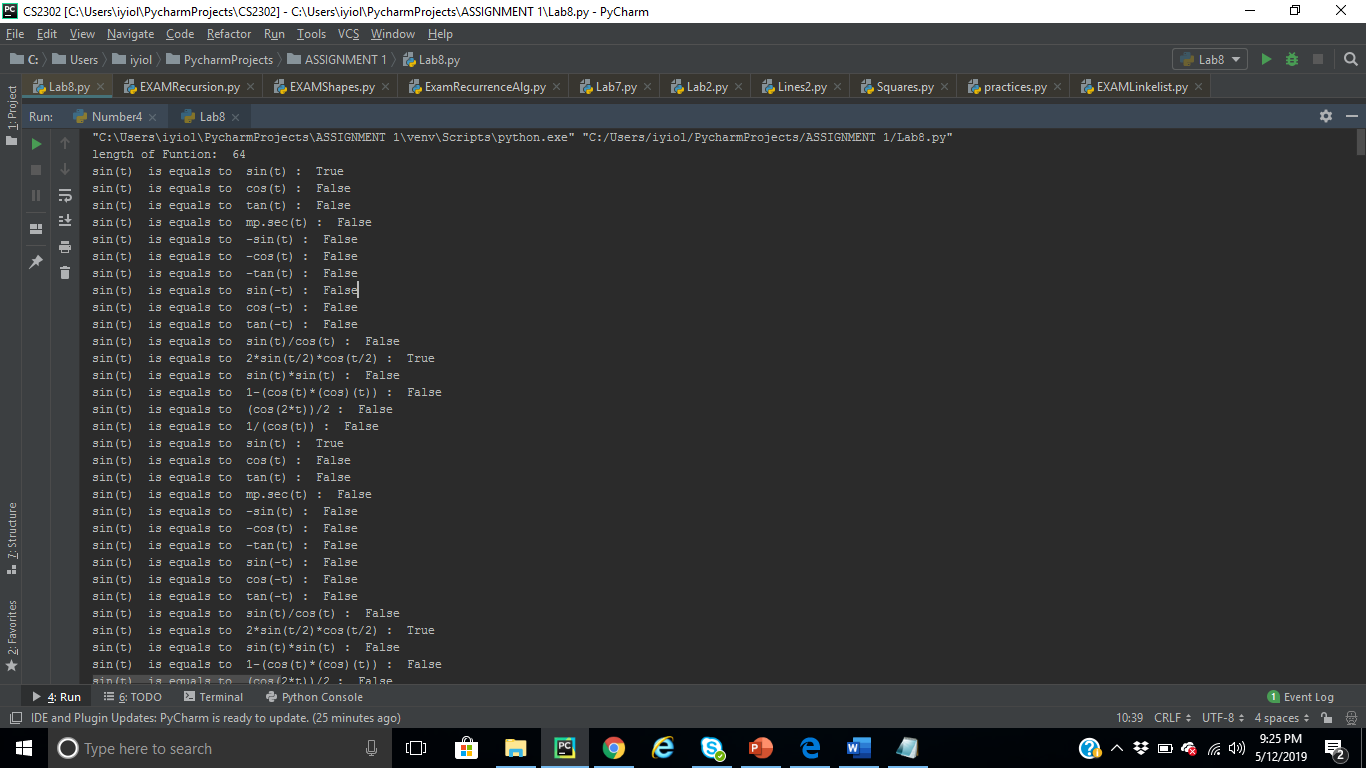
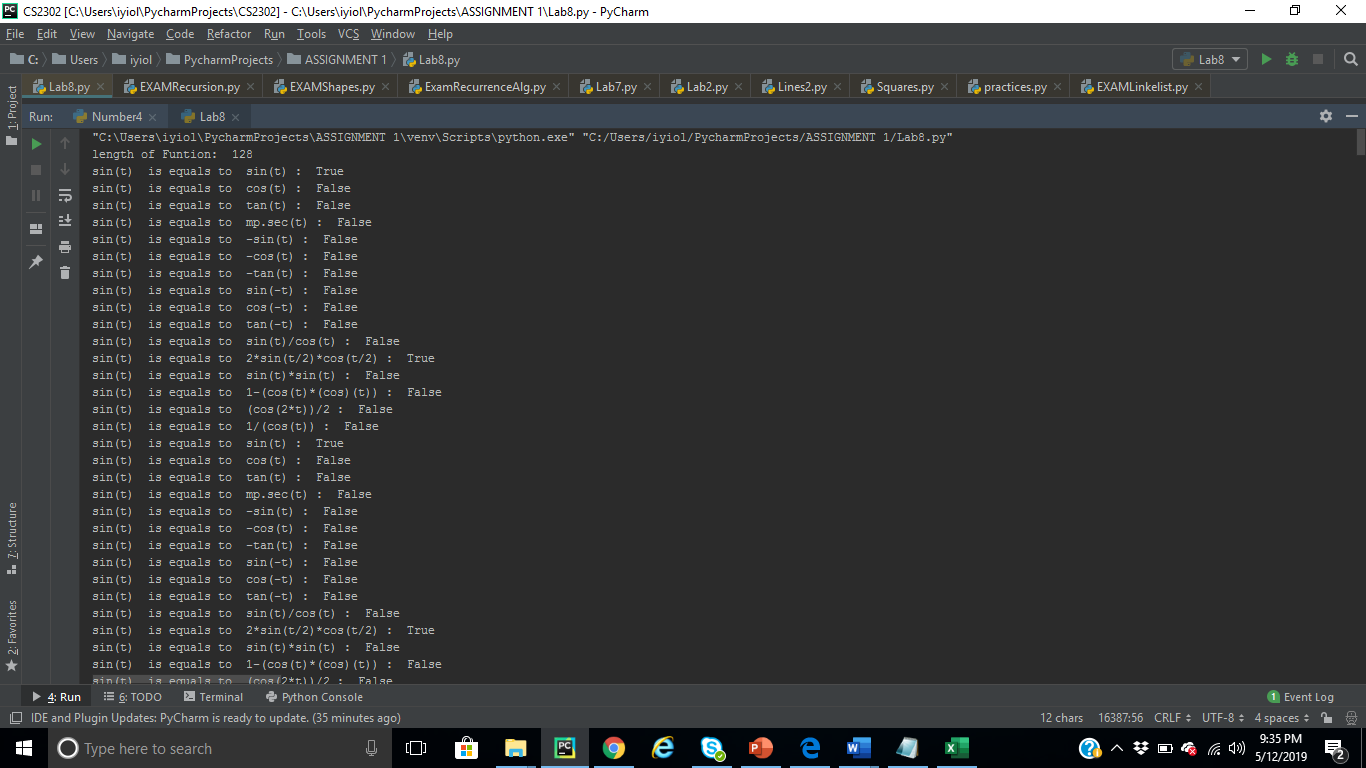
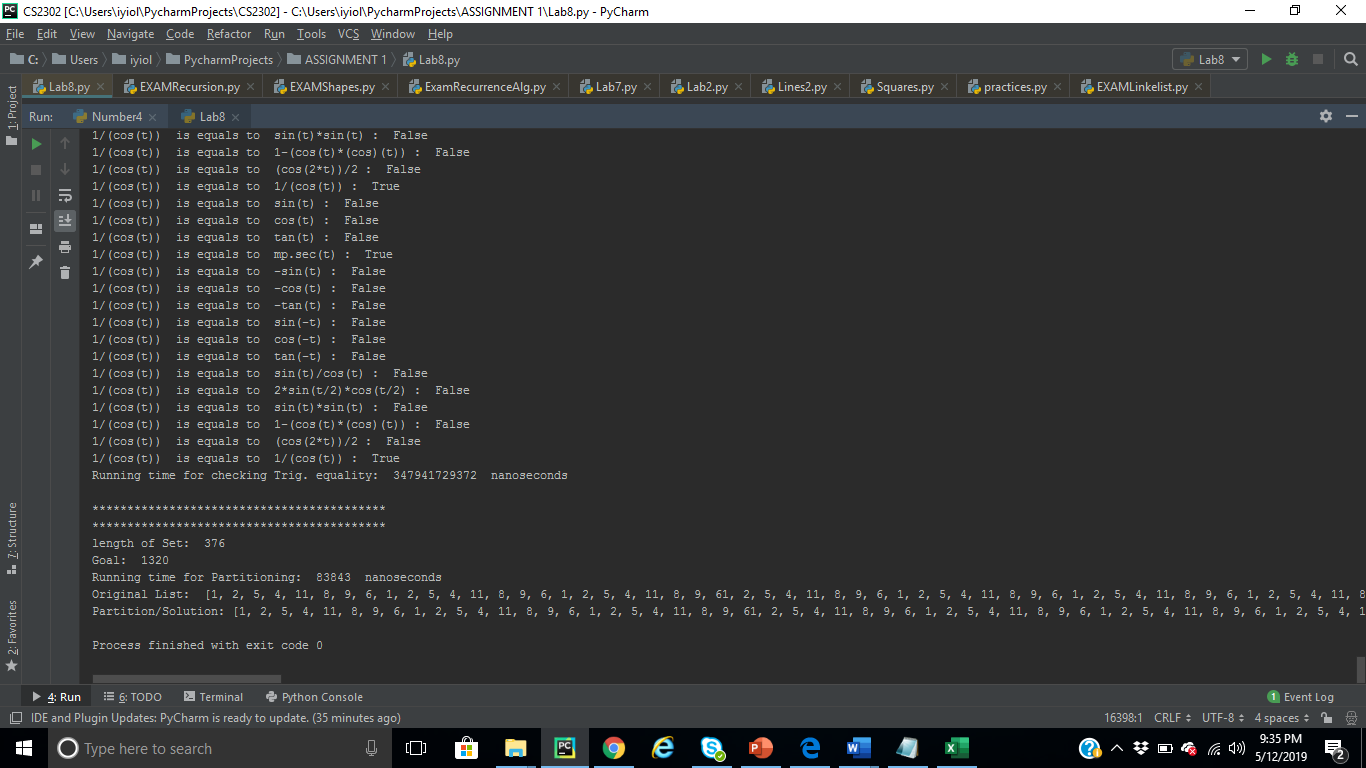
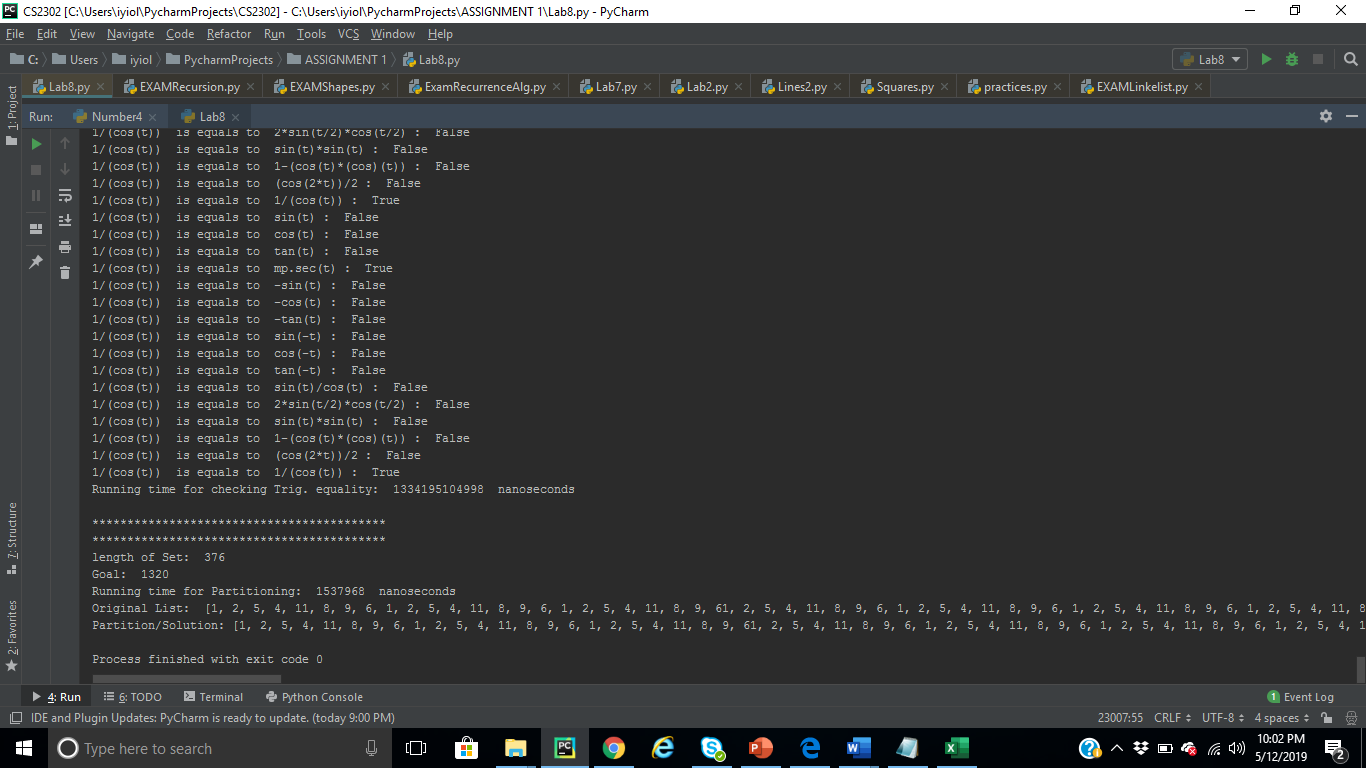
|  |  |
| --- | --- |
| **Number of Trig. Functions** | **Runtime in nanoseconds** |
| **16** | **5080508124** |
| **32** | **19688321106** |
| **64** | **76922189560** |
| **128** | **347941729372** |
| **256** | **1334195104998** |
|  |  |

O(n ^ 2)

**Checking Set Partition**

|  |  |
| --- | --- |
| **Number of Input/ Set Size** | **Runtime in nanoseconds** |
| **8** | **87682** |
| **24** | **104324** |
| **48** | **60802** |
| **94** | **60162** |
| **188** | **69762** |
| **376** | **78082** |
|  |  |

Recurrence: T(n-1) + 1 O(n log n)



**CONCLUSION**

In summary, Backtracking is just a way to attempt a partial solution (e.g. by adding constraints) and, when things don’t work well (or a better solution is sought), backtrack to the state before the attempt. In the worst case, a backtracking algorithm can take an exponential amount of time to complete

Similarly**,** a **randomized algorithm** is a technique that uses a source of randomness as part of its logic. It is typically used to reduce either the running time, or [time complexity](https://brilliant.org/wiki/big-o-notation/); or the memory used, or space complexity, in a standard algorithm. A randomized algorithm could help in a situation of doubt e.g. flipping a coin or a drawing a card from a deck in order to make a decision. Similarly, this kind of algorithm could help speed up a [brute force](https://brilliant.org/wiki/brute-force-algorithm/) process by randomly sampling the input in order to obtain a solution that may not be totally optimal, but will be good enough for the specified purposes.

.

**Appendix**

***Programmed by Olac Fuentes***

import random

import numpy as np

from math import \*

def equal(f1, f2,tries=1000,tolerance=0.0001):

for i in range(tries):

x = random.random()

y1 = eval(f1)

y2 = eval(f2)

if np.abs(y1-y2)>tolerance:

return False

return True

def prime(n,tries=1000):

m = int(np.sqrt(n))

for i in range(tries):

x = random.randint(2, m)

if n%x==0:

print(x)

return False

return True

def subsetsum(S,last,goal):

if goal ==0:

return True, []

if goal<0 or last<0:

return False, []

res, subset = subsetsum(S,last-1,goal-S[last]) # Take S[last]

if res:

subset.append(S[last])

return True, subset

else:

return subsetsum(S,last-1,goal) # Don't take S[last]

def edit\_distance(s1,s2):

d = np.zeros((len(s1)+1,len(s2)+1),dtype=int)

d[0,:] = np.arange(len(s2)+1)

d[:,0] = np.arange(len(s1)+1)

for i in range(1,len(s1)+1):

for j in range(1,len(s2)+1):

if s1[i-1] ==s2[j-1]:

d[i,j] =d[i-1,j-1]

else:

n = [d[i,j-1],d[i-1,j-1],d[i-1,j]]

d[i,j] = min(n)+1

#print(d)

return d[-1,-1]

f1 = 'x\*x + x - 12'

f2 = '(x+4)\*(x-3)'

print(equal(f1,f2))

f1 = 'sin(x)/cos(x)'

f2 = 'tan(x)'

print(equal(f1,f2))

f1 = 'sin(x)\*sin(x) + cos(x)\*cos(x)'

f2 = '1'

print(equal(f1,f2))

f1 = '(x+1)\*(x-1)'

f2 = 'x\*x-1'

print(equal(f1,f2))

f1 = '(x+10)/10'

f2 = 'x'

print(equal(f1,f2))

print(prime(997))

print(prime(1008))

s2='MONEY'

d = edit\_distance('MINERS','MONEY')

print(d)

S = [2,5,8,9,12,21,33]

for i in range(100):

print('Goal =',i)

a,s = subsetsum(S,len(S)-1,i)

if a:

print('Solution:',s)

else:

print('There is no solution')

**GeekforGeeks**

<https://www.geeksforgeeks.org/partition-set-k-subsets-equal-sum/>

**Wikipedia**

[**https://en.wikipedia.org/wiki/Sorting\_algorithm#Comparison\_of\_algorithms**](https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms)

**Academic Dishonesty**

This work was done by me without any act or practice of academic dishonesty

**SIGNATURE**

**OLUGBENGA IYIOLA(OT)**

**……………………………………………….**