Author: *Nguyen, Abram*
Assignment: *Lab 8 Report*
Course: *CS 2302 - Data Structures*
Instructor: *Fuentes, Olac*
T.A.: *Nath, Anindita*

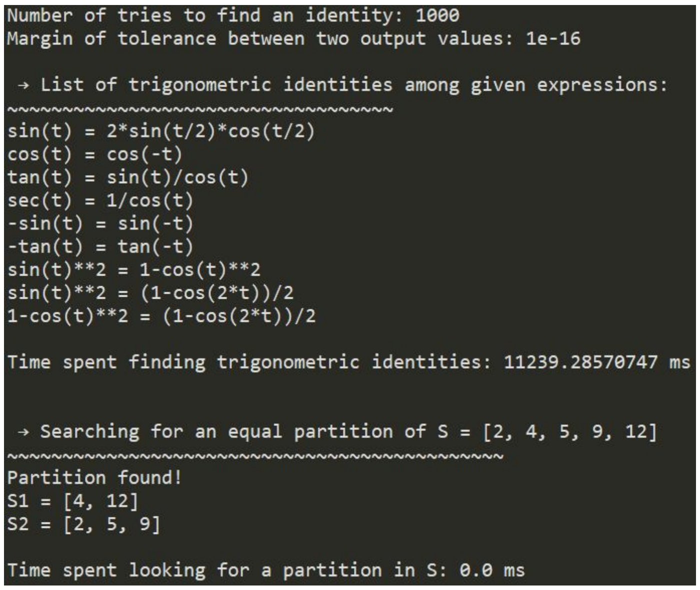
Introduction:

In this lab, I practice and demonstrate the use of a randomized algorithm and a backtracking algorithm. I use a randomized algorithm to 'discover' trigonometric identities by looking for expressions that are close or equal in value. I use a backtracking algorithm to find a partition in a set in which there are two subsets whose values add up to the same number.

Proposed solution design and implementation:

1) A randomized algorithm uses random values to guide its steps, using randomness as a part of its logic. To discover equalities between two expressions, I would plug in an arbitrary value 'x' (between -pi and pi, in this case) into both expressions and record function 1 as equal to function 2. To increase accuracy, however, I tested each expression against every other expression with 1000 random values between -pi and pi. A variable 'tolerance' is used to grade the accuracy of the difference in output of two expressions. This means that if the difference between two expression's output (when given the same value) is less than the value of 'tolerance', those two expressions will be considered equal. I have to test them this way because python's 'eval' function is only so accurate.
2) A backtracking algorithm will make a decision and, if that decision ends in a satisfactory result, it'll make another decision and keep going. It'll keep making decisions to solve a problem until it finds an output that it doesn't like. If that happens, it'll backtrack to undo a decision it's made, and make a different decision. I must use backtracking to find 2 separate subsets of a set such that both sets add up to the same number. In other words, I must find a subset of 'S' in which its values add up to half of the original sum of 'S'. To do this, I used a recursive approach. If my goal is reached exactly, then I've found my subset and will return True. If the sum of a subset is more than my goal, I backtrack and try a different subset. In the end, if I've found my subset, I will return a list of that subset, the second subset (the other half of 'S'), and my boolean value, True. Otherwise, I'll indicated that I couldn't find valid partition.

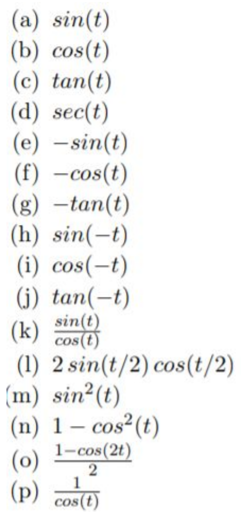
Results of a normal run of my program:



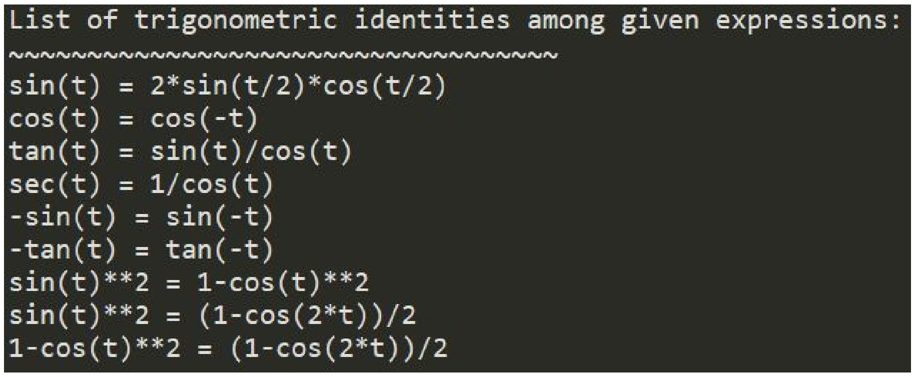
```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 1e-16

 → List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2

Time spent finding trigonometric identities: 11239.28570747 ms


 → Searching for an equal partition of S = [2, 4, 5, 9, 12]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Partition found!
S1 = [4, 12]
S2 = [2, 5, 9]

Time spent looking for a partition in S: 0.0 ms
```

<u>Experimental results:</u>

1) My first algorithm depends heavily on accuracy, so what if I played with the accuracy of the function?
   Below are the given list of expressions and the list of expected equalities.

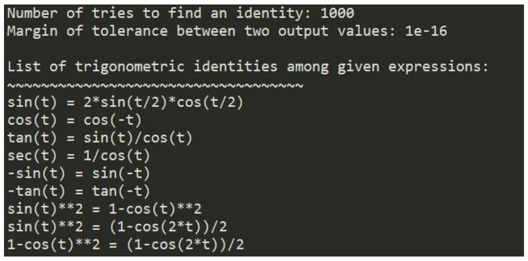


(a) $sin(t)$
(b) $cos(t)$
(c) $tan(t)$
(d) $sec(t)$
(e) $-sin(t)$
(f) $-cos(t)$
(g) $-tan(t)$
(h) $sin(-t)$
(i) $cos(-t)$
(j) $tan(-t)$
(k) $\frac{sin(t)}{cos(t)}$
(l) $2\,sin(t/2)\,cos(t/2)$
(m) $sin^2(t)$
(n) $1 - cos^2(t)$
(o) $\frac{1-cos(2t)}{2}$
(p) $\frac{1}{cos(t)}$

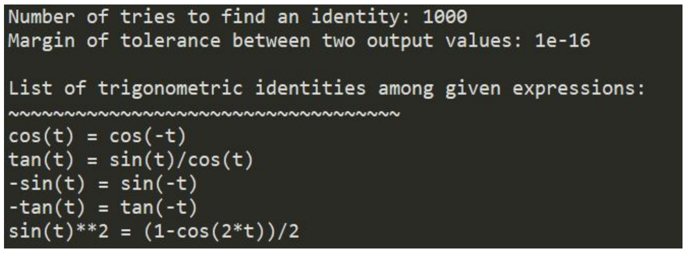


```
List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2
```

   a) Original results, when ‘tries’ is 1000, ‘tolerance’ is 1e-16 (0.0000000000000001): Every time I run the function, the results seem to be different every few runs, as shown below. This was something that I overlooked that could be fixed by increasing the accuracy of my function. Below are a couple examples of the differences between runs of the same input. The first output exactly what I had expected. The second output correct results, but not all of the expected results.



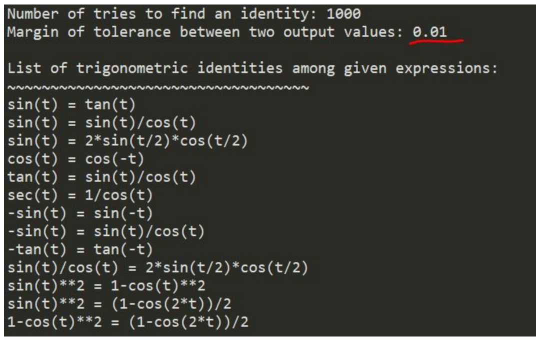
```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 1e-16

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2
```



```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 1e-16

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = (1-cos(2*t))/2
```

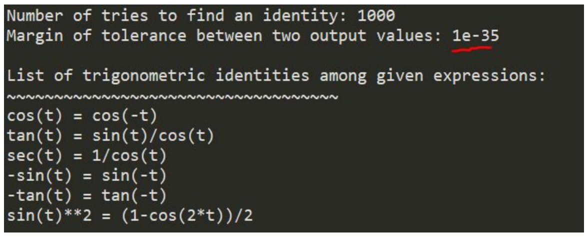
b) Changing ‘tolerance’:

   i) Increasing tolerance to 0.01 does what I had expected it to do. The function will output more equalities, including some that are true, but more that are false.

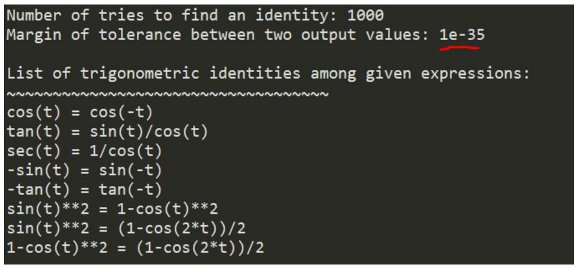


```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 0.01

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = tan(t)
sin(t) = sin(t)/cos(t)
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-sin(t) = sin(t)/cos(t)
-tan(t) = tan(-t)
sin(t)/cos(t) = 2*sin(t/2)*cos(t/2)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2
```

   ii) Decreasing tolerance to a certain point is beneficial, but after a certain point, the function won’t read numbers that low accurately. This usually results in less number of equalities.

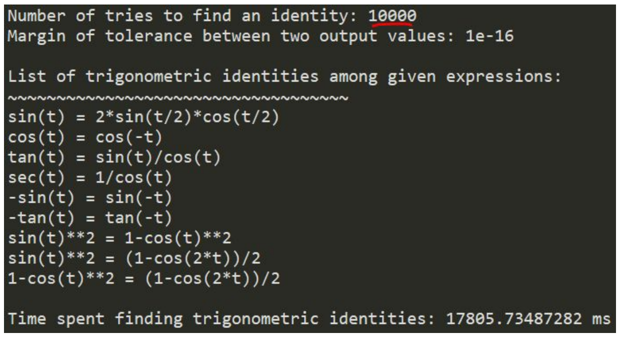


```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 1e-35

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = (1-cos(2*t))/2
```

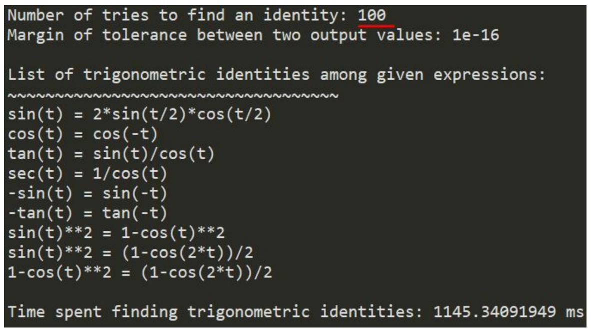


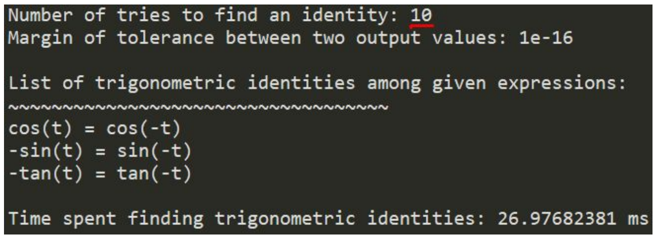
```
Number of tries to find an identity: 1000
Margin of tolerance between two output values: 1e-35

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2
```

c) Changing ‘tries’: Increasing the number of tries to determine the equality of two expressions usually resulted in more accurate results. I increased the number of tries from 1,000 to 10,000, which only sometimes resulted in a shorter list of equalities, but usually more accurate.


```
Number of tries to find an identity: 10000
Margin of tolerance between two output values: 1e-16

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2

Time spent finding trigonometric identities: 17805.73487282 ms
```

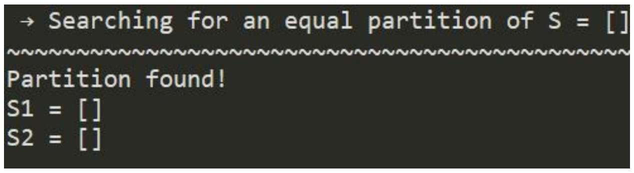
d) When I decreased the number of tries from 1,000 to 100, I got the same results most of the time. When I decreased the number of tries down to 10, it resulted in a shorter list of equalities.


```
Number of tries to find an identity: 100
Margin of tolerance between two output values: 1e-16

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
sin(t) = 2*sin(t/2)*cos(t/2)
cos(t) = cos(-t)
tan(t) = sin(t)/cos(t)
sec(t) = 1/cos(t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)
sin(t)**2 = 1-cos(t)**2
sin(t)**2 = (1-cos(2*t))/2
1-cos(t)**2 = (1-cos(2*t))/2

Time spent finding trigonometric identities: 1145.34091949 ms
```

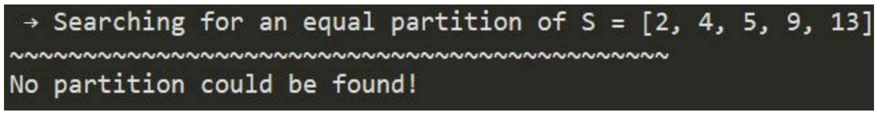

```
Number of tries to find an identity: 10
Margin of tolerance between two output values: 1e-16

List of trigonometric identities among given expressions:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cos(t) = cos(-t)
-sin(t) = sin(-t)
-tan(t) = tan(-t)

Time spent finding trigonometric identities: 26.97682381 ms
```

2) I decided to test my second function, partition, with a few different inputs.

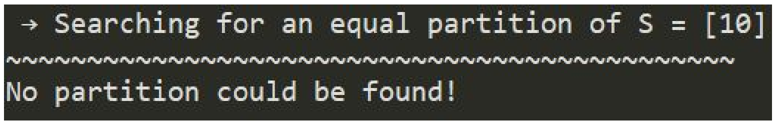
a) I began by testing my function using the simplest possible case, which was an empty list. It should indicated that there is a valid partition, which are two empty lists.


```
 → Searching for an equal partition of S = []
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Partition found!
S1 = []
S2 = []
```

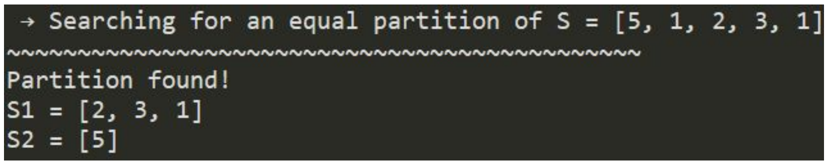
b) Given a list that adds up to an odd number, my function should indicate that an equal partition cannot be found


```
 → Searching for an equal partition of S = [2, 4, 5, 9, 13]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
No partition could be found!
```

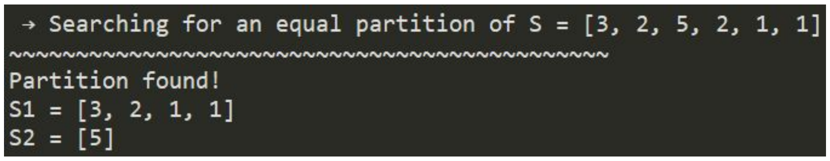
c) Given a list containing one element should also result in no valid partition, even if the sum of the list is even.


```
 → Searching for an equal partition of S = [10]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
No partition could be found!
```

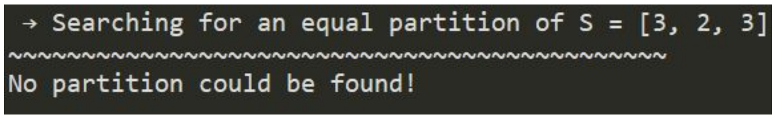
d) Given a list that is not a set, where two of the same number exists, is an interesting input that I tried. As expected, the method doesn't work correctly given a list containing repeated values. Below are a few examples of lists containing repeated values.



```
 → Searching for an equal partition of S = [5, 1, 2, 3, 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Partition found!
S1 = [2, 3, 1]
S2 = [5]
```

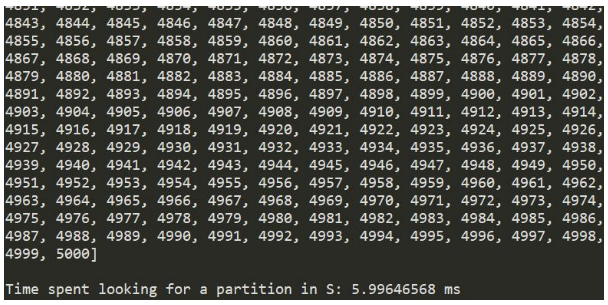


```
 → Searching for an equal partition of S = [3, 2, 5, 2, 1, 1]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Partition found!
S1 = [3, 2, 1, 1]
S2 = [5]
```

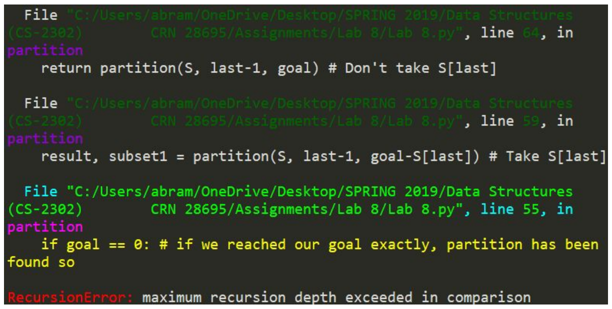


```
 → Searching for an equal partition of S = [3, 2, 3]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
No partition could be found!
```

e) The run time of my function would only significantly change if I input a list of significantly large list of numbers. For example, given 5,000 numbers, it takes somewhere between 5 and 6 milliseconds. Any list larger than about 5,000 numbers ended up exceeding the maximum recursion depth.



```
4843, 4844, 4845, 4846, 4847, 4848, 4849, 4850, 4851, 4852, 4853, 4854,
4855, 4856, 4857, 4858, 4859, 4860, 4861, 4862, 4863, 4864, 4865, 4866,
4867, 4868, 4869, 4870, 4871, 4872, 4873, 4874, 4875, 4876, 4877, 4878,
4879, 4880, 4881, 4882, 4883, 4884, 4885, 4886, 4887, 4888, 4889, 4890,
4891, 4892, 4893, 4894, 4895, 4896, 4897, 4898, 4899, 4900, 4901, 4902,
4903, 4904, 4905, 4906, 4907, 4908, 4909, 4910, 4911, 4912, 4913, 4914,
4915, 4916, 4917, 4918, 4919, 4920, 4921, 4922, 4923, 4924, 4925, 4926,
4927, 4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936, 4937, 4938,
4939, 4940, 4941, 4942, 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950,
4951, 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960, 4961, 4962,
4963, 4964, 4965, 4966, 4967, 4968, 4969, 4970, 4971, 4972, 4973, 4974,
4975, 4976, 4977, 4978, 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986,
4987, 4988, 4989, 4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998,
4999, 5000]

Time spent looking for a partition in S: 5.99646568 ms
```



```
  File "C:/Users/abram/OneDrive/Desktop/SPRING 2019/Data Structures
(CS-2302)        CRN 28695/Assignments/Lab 8/Lab 8.py", line 64, in
partition
    return partition(S, last-1, goal) # Don't take S[last]

  File "C:/Users/abram/OneDrive/Desktop/SPRING 2019/Data Structures
(CS-2302)        CRN 28695/Assignments/Lab 8/Lab 8.py", line 59, in
partition
    result, subset1 = partition(S, last-1, goal-S[last]) # Take S[last]

  File "C:/Users/abram/OneDrive/Desktop/SPRING 2019/Data Structures
(CS-2302)        CRN 28695/Assignments/Lab 8/Lab 8.py", line 55, in
partition
    if goal == 0: # if we reached our goal exactly, partition has been
found so

RecursionError: maximum recursion depth exceeded in comparison
```

Conclusion:

This lab is fairly simple and straightforward. I enjoyed working with algorithms based on randomization and backtracking. The more interesting part of the lab, to me, was the randomization part. There are many different applications of randomization, backtracking, and other types of algorithm design techniques. I would really like to see more of Dynamic Programming in the future. Working on this lab, I learned more about how algorithm design techniques can be applied to different problems.

Appendix:

```
"""
Author:             Nguyen, Abram
Assignment:         Lab 8
Course:             CS 2302 - Data Structures
Instructor:         Fuentes, Olac
T.A.:               Nath, Anindita
Last modified:
Purpose of program: The purpose of this program is to practice and demonstrate the
                       uses of randomized and backtracking algorithms, two of several
                       algorithm design techniques. I use randomization and
                       backtracking to come up with a solution to 2 given problems.
"""
import numpy as np
import random
import math
from mpmath import *
import time


# Write a program to ”discover” trigonometric identities. Your program should
# test all combinations of the trigonometric expressions shown below and use a
# randomized algorithm to detect the equalities. For your equality testing,
# generate random numbers in the −π to π range.
# What if I change 'tolerance'?
# What if I change 'tries' to a lower number?
def equal(trig, tries=1000, tolerance=0.0000000000000001):
   print("Number of tries to find an identity:", tries)
   print("Margin of tolerance between two output values:", tolerance)
   # i've modified the method so that it's as accurate as possible
   identities = []
   # compare each expression...
   for i in range(len(trig)):
       # ...to every other expression in the list
       for j in range(i, len(trig), 1):
           if i != j:
               for k in range(j, tries):
                   # test expression using a number between -pi and pi
                   t = random.uniform(-math.pi, math.pi)
                   y1 = eval(trig[i])
                   y2 = eval(trig[j])
                   # if two expressions are equal...
                   if np.abs(y1-y2)<=tolerance:
                       # ...an identity has been found! record it
                       identities.append(f"{trig[i]} = {trig[j]}")
                       break
   return identities




# Split a list of values into 2 subsets such that the sum of the numbers in
# subset 1 is equal to the sum of the numbers in subset 2. S1 and S2 must contain
# different numbers, the original set must be divided.
# Ex: {2, 4, 5, 9, 12} can be partitioned into S1 = {2, 5, 9} and S2 = {4, 12}.
# Indicate whether or not a valid partition exists.
def partition(S, last, goal):
   if goal == 0: # if we reached our goal exactly, partition has been found so
       return True, []
   if goal < 0 or last < 0: # if we've gone over our goal, no partition has been found so far
       return False, []
   result, subset1 = partition(S, last-1, goal-S[last]) # Take S[last]
   if result: # if we're still on track with our goal, add the last checked value to the current subset
       subset1.append(S[last])
```

```
        return True, subset1 # return True and the current subset we've made
    else:
        return partition(S, last-1, goal) # Don't take S[last]


# create a subset of values that are in 'S' but are not in 's1'
def set2(S, s1):
    s2 = []
    for s in S:
        if s not in s1:
            s2.append(s)
    return s2


"""
################################################################################
################################################################################
################################################################################
"""
trig = ['sin(t)', 'cos(t)' , 'tan(t)', 'sec(t)', '-sin(t)', '-cos(t)', '-tan(t)',
        'sin(-t)', 'cos(-t)', 'tan(-t)', 'sin(t)/cos(t)', '2*sin(t/2)*cos(t/2)',
        'sin(t)**2', '1-cos(t)**2', '(1-cos(2*t))/2', '1/cos(t)']


start = time.time()
identities = equal(trig)
end = time.time()


print(f"\n → List of trigonometric identities among given
expressions:\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
for i in identities:
    print(i)
print()
print(f"Time spent finding trigonometric identities: {round((end-start)*1000, 8)} ms\n")


################################################################################
#S = []
S = [2, 4, 5, 9, 12]
#S = [2, 4, 5, 9, 13]
#S = [10]
#S = [3, 2, 3]


print(f"\n → Searching for an equal partition of S = {S}")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")


start = time.time()
contains, s1 = partition(S, len(S)-1, goal=sum(S)/2)
end = time.time()


if contains:
    s2 = set2(S, s1)
    print("Partition found!")
    print(f"S1 = {s1}\nS2 = {s2}")
else:
    print("No partition could be found!")

print(f"\nTime spent looking for a partition in S: {round((end-start)*1000, 8)} ms\n")
```

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

- Abram Nguyen