**Introduction**

**Trig\_equality:**

Proposed solution and implementation:

This algorithm was created very similar to the “equal” function from the algorithm\_design.py file. However, some different guidelines were taken when building the function.

First a global list of trigonometric identities was created to access them from the function and not clutter its body with only trig identities. At the beginning of the function a random float number is chosen from a range of -π to π to use for equality testing for each trig identity. Then a for loop is used to try different types of comparisons set by the number of tries given by the functions parameters.

Inside the loop, *random\_trig\_func()*, is used to obtain a random trig identity and returns its string to store in a temporary variable. Then both variables in which the trig identities are stored, its equations are evaluated and stored in *x and y*. Consequently, an if statement is used to compare the subtraction of both evaluated functions, and if they are very close to the definite zero value it adds to a counter and prints the trig identities. The for loop repeats this process for the amount of tries given, but once finished, the function finally returns the count number representing the amount of identities equal.

Running time and time complexity:

Experiment: amount of tries 0

Result: Low running time and 0 matched trig functions

Experiment: Change the functions tries to 100

Result: Low running time and only about 10 matched trig functions

Experiment: Increment the amount of tries by multiplying 10 (Ex. 1000, 10000, 100000)

Result: Running times change according to big O complexity of algorithm and amount of matched trig functions are about 10% each time

*Big O complexity: O(2n)*

Outputs:

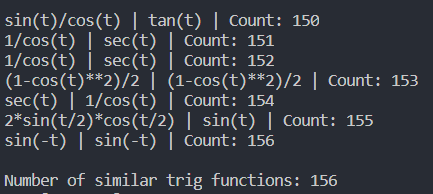


Figure 1: Number of tries set to 1000

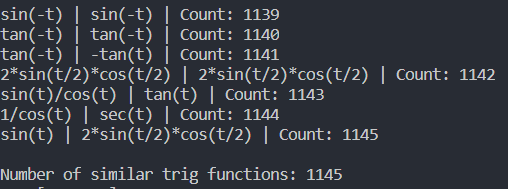


Figure 2: Number of tries set to 10000

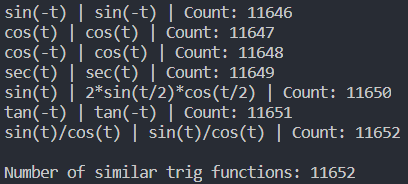


Figure 3: Number of tries set to 100000

**Partition:**

Proposed solution and implementation:

This algorithm determines if there is a partition from a given set of integers.

The algorithm simply starts with the summation of the Set of numbers by using a for loop, traversing through each number and adding it to the sum variable. The sum is then used to determine if the sum is odd or even. An if statement helps determine the sum’s property by finding if the sum is odd, then it returns false for the function. However, if the sum is even, then we find a new goal by dividing it by 2 and returning it. Finally, the partition function’s output is stored in a variable by which it is then used for the subsetsum for the goal parameter.

Running time and time complexity:

Experiment: Creating an empty set

Result: Returned “Set does not contain partition”

Experiment: Creating set as [2,4,5,9,12]

Result: Returned s1 [2,5,9] and s2 [4,12]

Experiment: Creating set as [2,4,5,9,13]

Result: Returned “Set does not contain partition”

*Big O complexity: O(2n)*

Outputs:



Figure 4: Having the set [2,4,5,9,12]



Figure 5: Having the set [2,4,5,9,13]



Figure 6: Having set [4, 6, 47, 48, 49, 50]

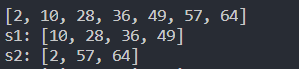


Figure 7: Having set [2,10,28,36,49,57,64]

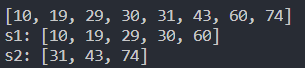


Figure 8: Having set [10,19,29,30,31,43,60,74]

Appendix

*import* random

*import* numpy *as* np

*from* math *import* \*

*from* mpmath *import* sec

*import* math

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(t)\*\*2)/2',

'1/cos(t)'

]

def random\_trig\_func():

x = random.choice(trig)

*return* x

def trig\_equality(*tries*=1000,*tolerance*=0.001):

count = 0

t = random.uniform(-math.pi,math.pi)

*for* i in range(tries):

s1 = random\_trig\_func()

s2 = random\_trig\_func()

x = eval(s1)

y = eval(s2)

*if* np.abs(x-y) < tolerance:

count += 1

print(s1,"|",s2,"| Count:",count)

print()

*return* count

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 partition(*n*):

s = 0

*for* i in n:

s += i

*if* s % 2 != 0:

*return* False

goal = s // 2

*return* goal

print("Number of similar trig functions:",trig\_equality())

S = [2,4,5,9,12]

p = partition(S)

a,s2 = subsetsum(S,len(S)-1,p)

s1 = []

*if* p:

*for* i in S:

*if* i not in s2:

s1.append(i)

print('s1:',s1)

print('s2:',s2)

*else*:

print('')

Academic dishonesty

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.