Modify the APPROX-SUBSET-SUM procedure from chapter 35.5 of the textbook such that it will also return the subset of S which sums up to z\* (rather than just the value z\*).

You may want to actually implement the algorithm so that you can test the output, but pseudocode is acceptable.

The algorithm from the textbook which you will be modifying is as follows (please see the textbook for further details):

Let be a set of integer values.

Let be a target integer we want some subset of S to add up to.

Let be and approximation parameters where

The following algorithm will return a value which is within a factor of of the subset sum closest to the value .

1. // this is the merge procedure from merge sort
2. //detailed on next page
3. Remove from every element >
4. Let be the largest value in
5. Return

Let be a sorted list of integer values.

Let be some floating point value .

The follow algorithm will output a sorted list such that for every element removed from there is another element left behind which approximates. Specifically for each but there will be some and such that.

1. Let be the length of
2. // since L is sorted
3. Append onto the end of
5. Return

#Had to import the deepcopy function because python lists are strange.

from copy import deepcopy

#I realize this is not how the proper merge works, but for this problem it doesn’t matter

#The only important change is that merge is modified to be able to handle nested lists

def merge(L,L\_PRIME):

MERGELIST = L + L\_PRIME

#Sorting the nested list. O(nlogn) according to:

#https://stackoverflow.com/questions/14434490/what-is-the-complexity-of-this-python-sort-method

MERGELIST.sort(key=lambda x: sum(x))

return(MERGELIST)

#This function appends the value of S[i] to all values

#currently in L, time complexity of ~O(len(L\_PRIME))

def append\_values(L\_PRIME, S\_VAL):

for i in range (len(L\_PRIME)):

L\_PRIME[i].append(S\_VAL)

return(L\_PRIME)

#The main function. Instead of just storing the integer value of the S elements summed. It stores the values from S used to make the element in L, so its a list of subset lists.

def approx\_subsetSum(S, t, epsilon):

trim\_lambda = epsilon / (2 \* len(S))

L = [[0]]

L\_PRIME = deepcopy(L)

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

L\_PRIME = append\_values(L\_PRIME, S[i-1]) #Append S[i] to all element lists

L = merge(L, L\_PRIME)

L = trim(L, trim\_lambda)

for j in range(len(L)-1, 0, -1):

if(sum(L[j]) > t):

del(L[j])

L\_PRIME = deepcopy(L)

#This dual return could easily just return the subset list and use sum in main

return(sum(L[-1]),L[-1][1:])

#The only real changes to trim were using sum(L[i]) instead of just L[i]

#due to us passing subset lists instead of just the already calculated int.

def trim(L, gamma):

R = [[0]]

last = sum(L[0])

for i in range(1, len(L)):

if (sum(L[i]) > (last \* (1 + gamma))):

R.append(L[i])

last = sum(L[i])

return(R)

#Main for testing the above functions. Im sure there are cases that this code breaks, but it worked every time I had tinkered with it up until this point.

if \_\_name\_\_ == "\_\_main\_\_":

S = [16, 32, 15, 109, 41, 67, 37, 30, 19, 6, 10, 3, 93, 110, 63, 18, 77, 69, 36, 70]

t = 665

epsilon = 0.10

Sum, Subset = approx\_subsetSum(S, t, epsilon)

print('Approximate Subset Sum with', epsilon \* 100, '% allowed error:')

print('Sum =', Sum, ', From Subset of S:',Subset)