CS301 Algorithms Subset Sum Problem

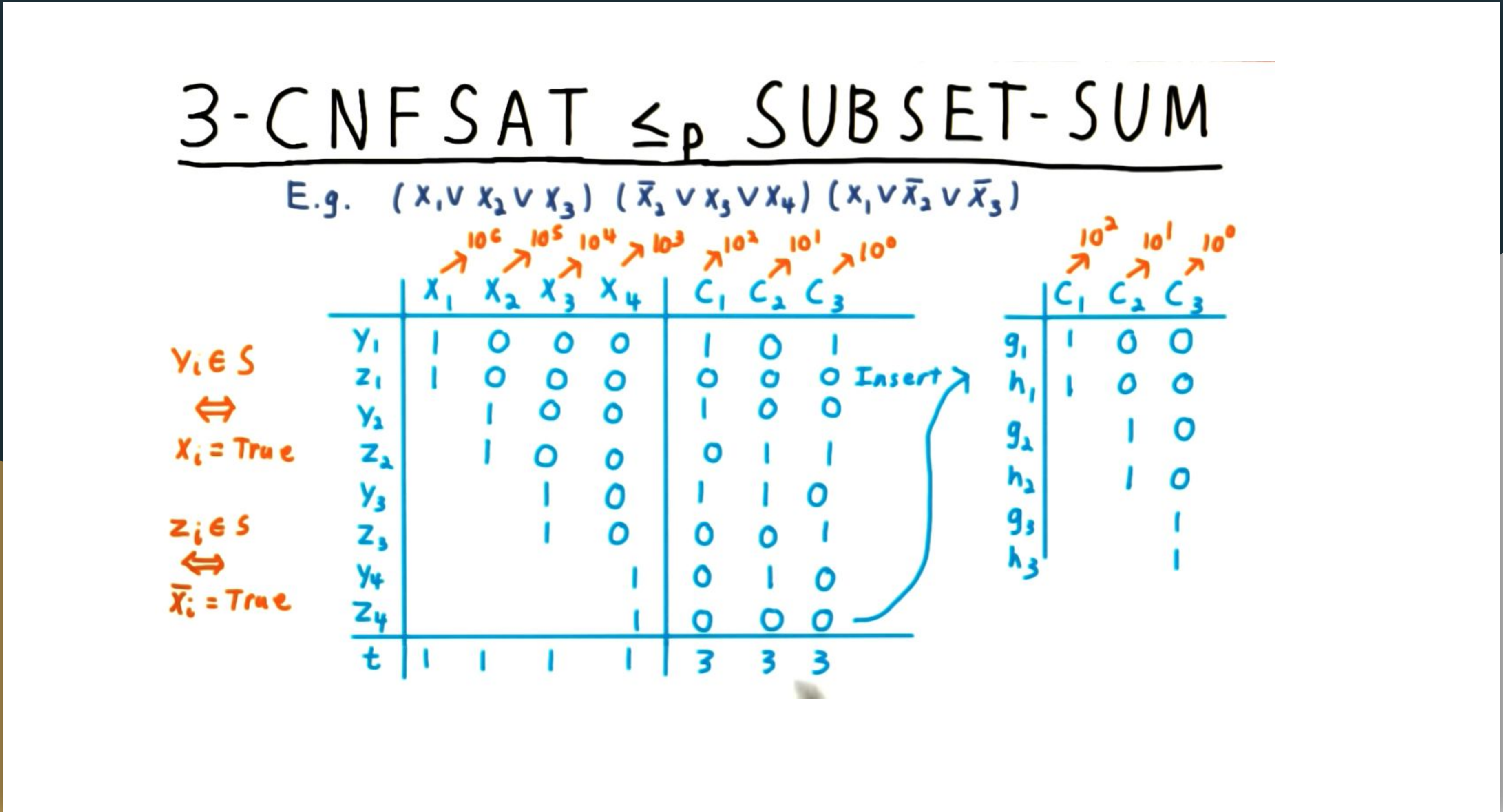
Berk Öztürk
Çağhan Köksal
Furkan Çelik
Hakan Buğra Erentuğ
Nidanur Günay

Outline

- Problem Description
- Algorithm Description
- Algorithm Analysis
- Experimental Analysis
- Testing
- Conclusion

Problem Description

- An arithmetic type of NP-complete
- Given =
  - finite set (S) of positive integers
  - target integer (T)
- Target =
  - Find a subset of S (S’) whose elements sums up to



3-CNFSAT $\leq_p$ SUBSET-SUM

E.g. $(x_1 \vee x_2 \vee x_3)\ (\bar{x}_2 \vee x_3 \vee x_4)\ (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$

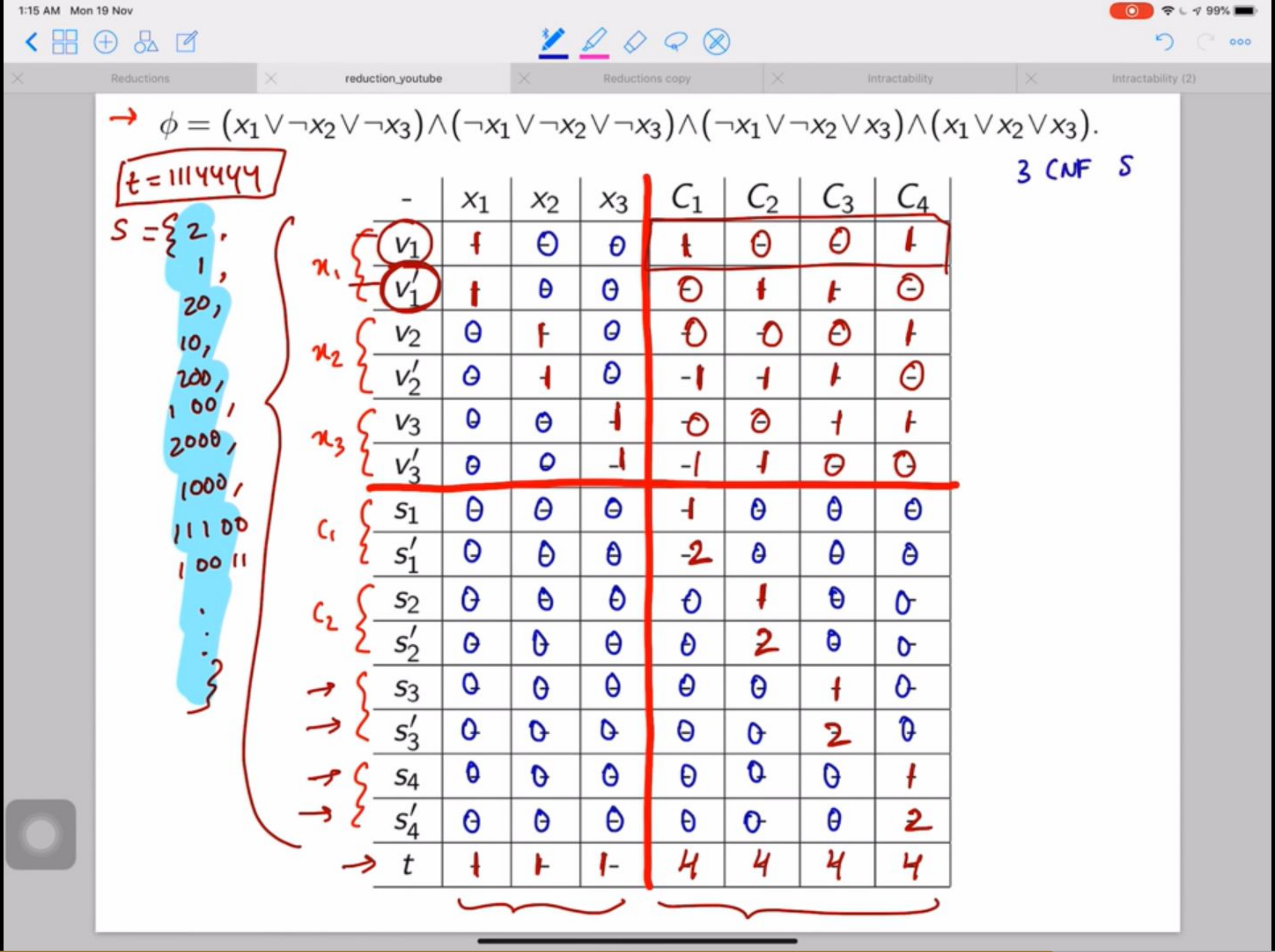
$y_i \in S$ ⇔ x_i = True

$z_i \in S$ ⇔ $\bar{x}_i$ = True

	10^6	10^5	10^4	10^3	10^2	10^1	10^0
	x_1	x_2	x_3	x_4	C_1	C_2	C_3
y_1	1	0	0	0	1	0	1
z_1	1	0	0	0	0	0	0
y_2		1	0	0	1	0	0
z_2		1	0	0	0	1	1
y_3			1	0	1	1	0
z_3			1	0	0	0	1
y_4				1	0	1	0
z_4				1	0	0	0
t	1	1	1	1	3	3	3

Insert

	10^2	10^1	10^0
	C_1	C_2	C_3
g_1	1	0	0
h_1	1	0	0
g_2		1	0
h_2		1	0
g_3			1
h_3			1



→ $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3).$

$\boxed{t = 1114444}$

3 CNF S

$S = \{2, 1, 20, 10, 200, 100, 2000, 1000, 11100, 10011, \ldots\}$

	-	x_1	x_2	x_3	C_1	C_2	C_3	C_4
x_1	v_1	1	0	0	1	0	0	1
	v_1'	1	0	0	0	1	1	0
x_2	v_2	0	1	0	0	0	0	1
	v_2'	0	1	0	1	1	1	0
x_3	v_3	0	0	1	0	0	1	1
	v_3'	0	0	1	1	1	0	0
c_1	s_1	0	0	0	1	0	0	0
	s_1'	0	0	0	2	0	0	0
c_2	s_2	0	0	0	0	1	0	0
	s_2'	0	0	0	0	2	0	0
→	s_3	0	0	0	0	0	1	0
→	s_3'	0	0	0	0	0	2	0
→	s_4	0	0	0	0	0	0	1
→	s_4'	0	0	0	0	0	0	2
→	t	1	1	1	4	4	4	4

Problem Description

To show that Subset Sum is NP-Complete:

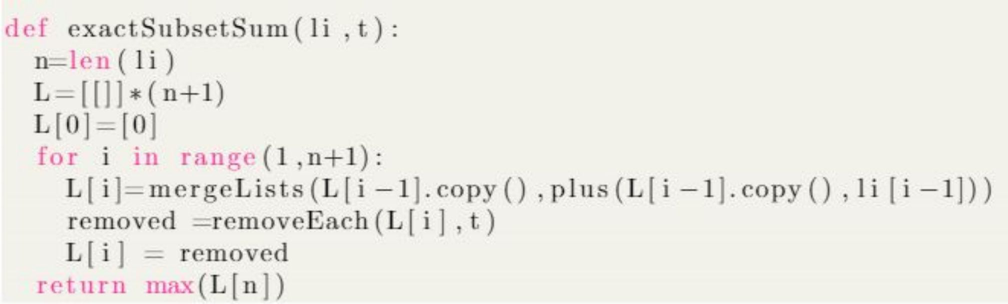
1. Subset Sum problem is in NP:Given a set S, we need to verify if $\sum_{i \epsilon S} w_i$ is equal to W.
2. A known NP-Complete problem can be transformed into subset sum problem in polynomial time

Algorithm Description

- Since Subset Sum is NP-Complete, there is no exact answer in polynomial time
- EXACT-SUBSET-SUM is an answer working in exponential time
    - Run time depends on # bits needed to represent the input
    - Note: There are some special cases for polynomial time
- We used some auxiliary functions as well (plus, mergeLists, removeEach)
- exactSubsetSum is our main function

Algorithm Description: exactSubsetSum

1. Adds first element to all elements list
2. Looks for new numbers
3. Tries to sum up
    a. If still lower than expected => let them stay in list
    b. If becomes larger than expected => delete from list
4. Adds second element to all elements list
5. Continues

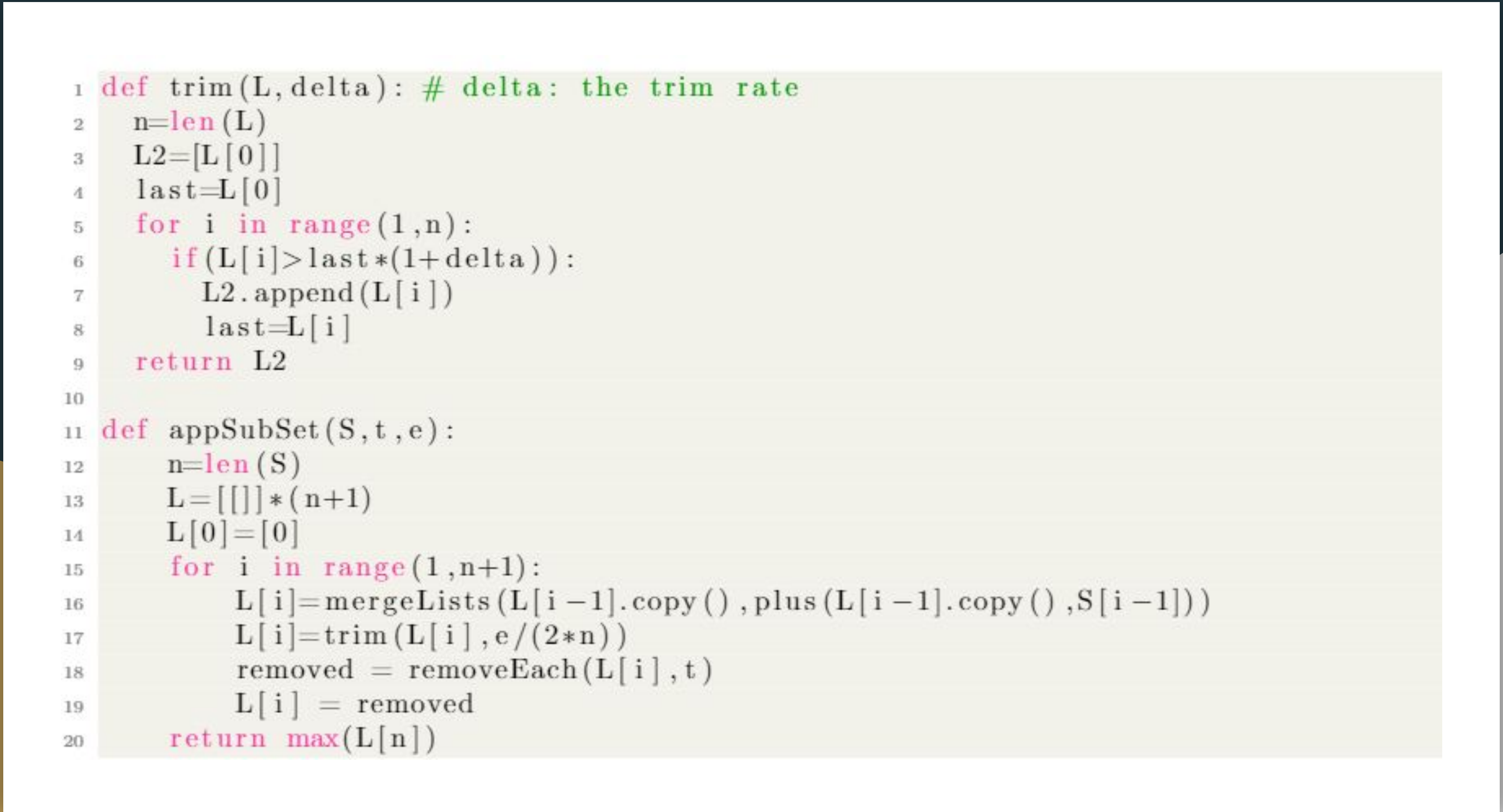


```
def exactSubsetSum(li ,t):
  n=len(li)
  L=[[]]*(n+1)
  L[0]=[0]
  for i in range(1,n+1):
    L[i]=mergeLists(L[i-1].copy(),plus(L[i-1].copy(),li[i-1]))
    removed =removeEach(L[i],t)
    L[i] = removed
  return max(L[n])
```

Algorithm Description: appSubSet

appSubSet trims given data to reduce exactSubSet algorithm from exponential time to logarithmic time

1. Adds first element to all elements list
2. Trims some numbers
    a. Deletes elements who is 1+delta lower than previous entry
3. Tries to sum up
    a. If still lower than expected => let them stay in list
    b. If becomes larger than expected => delete from list
4. Adds second element to all elements list
5. Continues

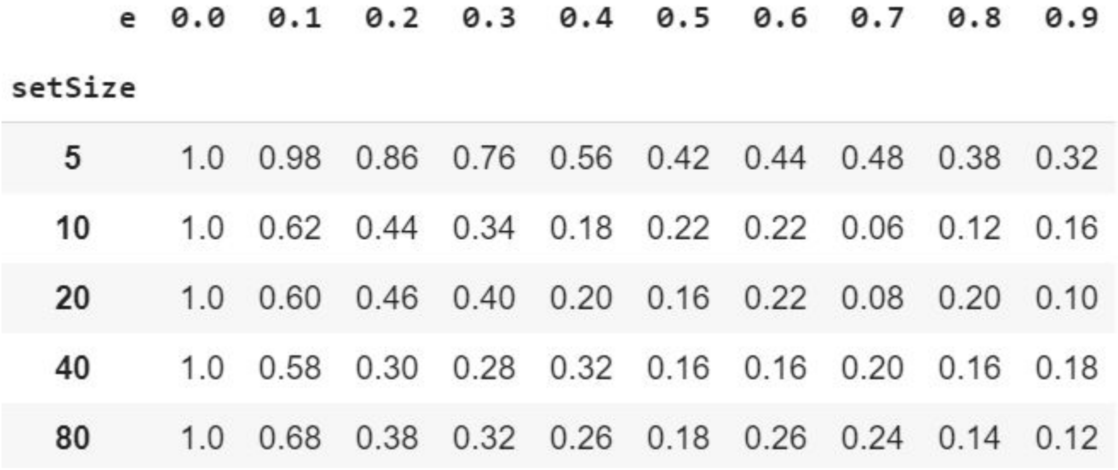


```python
def trim(L,delta): # delta: the trim rate
  n=len(L)
  L2=[L[0]]
  last=L[0]
  for i in range(1,n):
    if(L[i]>last*(1+delta)):
      L2.append(L[i])
      last=L[i]
  return L2

def appSubSet(S,t,e):
    n=len(S)
    L=[[]]*(n+1)
    L[0]=[0]
    for i in range(1,n+1):
        L[i]=mergeLists(L[i-1].copy(),plus(L[i-1].copy(),S[i-1]))
        L[i]=trim(L[i],e/(2*n))
        removed = removeEach(L[i],t)
        L[i] = removed
    return max(L[n])
```

Experimental Analysis

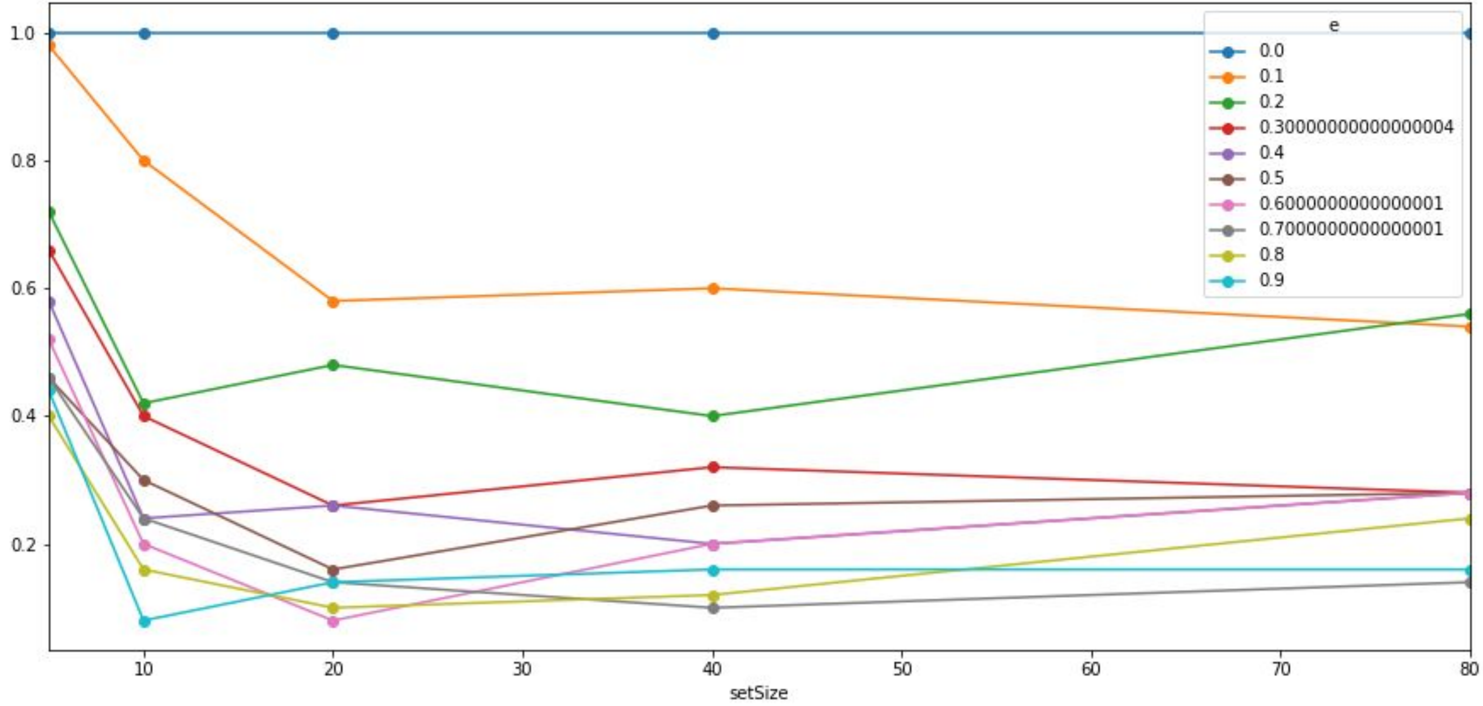
Success Rate

- Since exactSumSet generates true results for all cases, we used it in our comparisons for success cases of appSumSet. Even having one failure case is enough to say run has failed. This failure mainly comes to appSumSet not finding a result even though there is a result.
- Size of lists selected as 10, 20, 40 and 80 and populated randomly from 0 to 100, 1000 and 10000



e	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
setSize										
5	1.0	0.98	0.86	0.76	0.56	0.42	0.44	0.48	0.38	0.32
10	1.0	0.62	0.44	0.34	0.18	0.22	0.22	0.06	0.12	0.16
20	1.0	0.60	0.46	0.40	0.20	0.16	0.22	0.08	0.20	0.10
40	1.0	0.58	0.30	0.28	0.32	0.16	0.16	0.20	0.16	0.18
80	1.0	0.68	0.38	0.32	0.26	0.18	0.26	0.24	0.14	0.12

Success Rates (number range:[1,100], setSizes:{5,10,20,40,80})

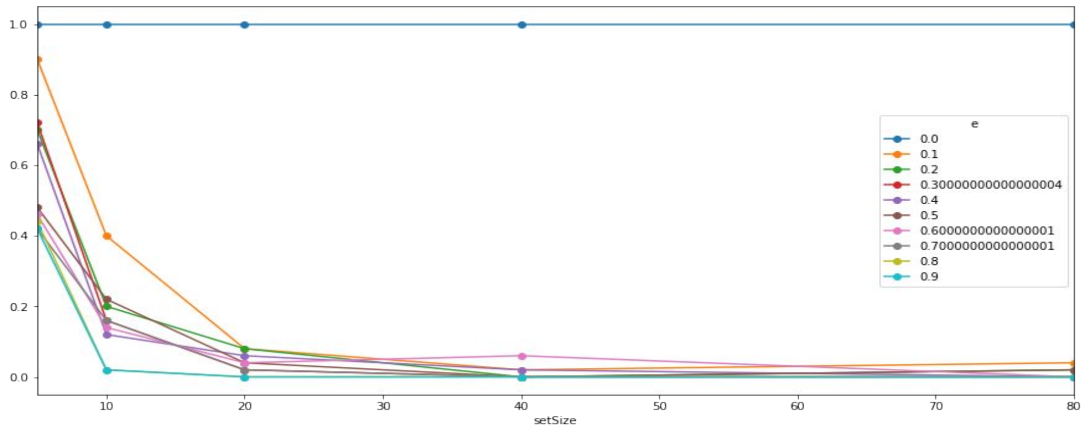


Experimental Analysis

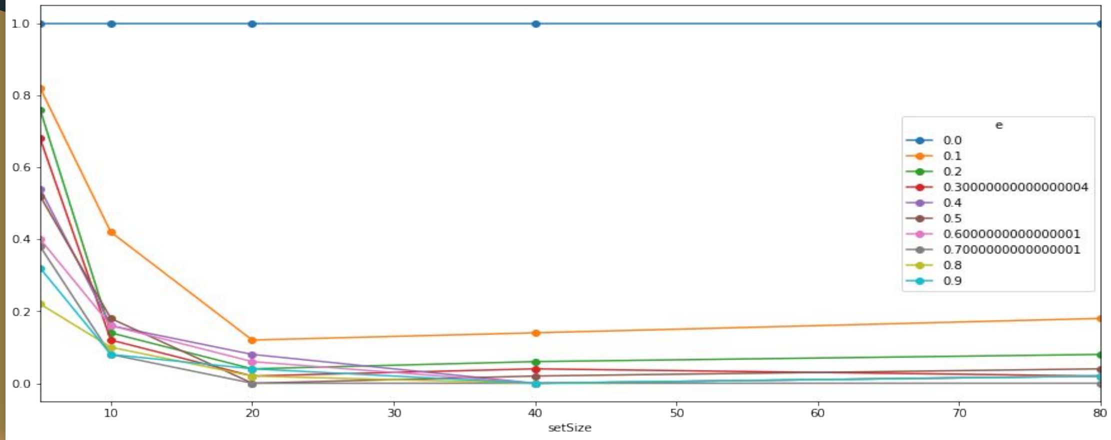
- Increasing epsilon value significantly decreases correctness
- Increasing setsize mostly does not affect success rate

Increasing range don’t have a significant change on success rate
Hence data gathered from 100,1000 and 10000 are similar to each other

Success Rates (number range:[1,10000], setSizes:{5,10,20,40,80})

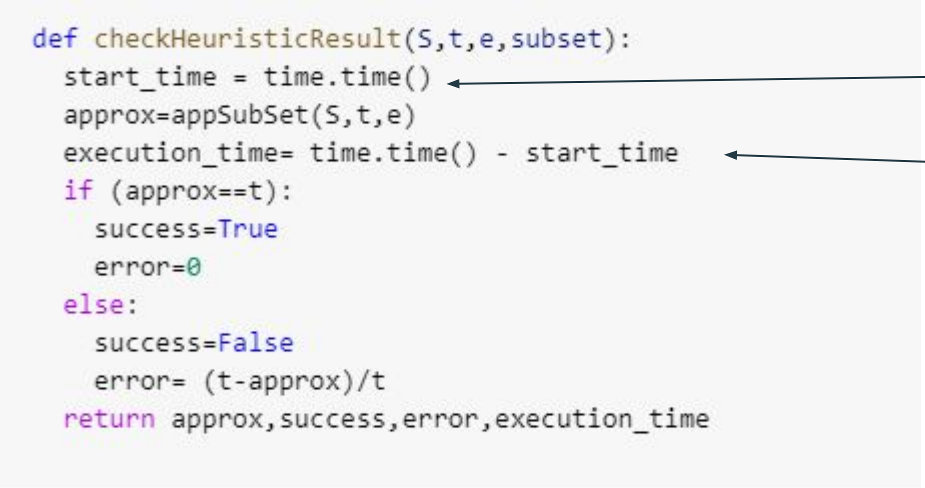


Success Rates (number range:[1,1000], setSizes:{5,10,20,40,80})



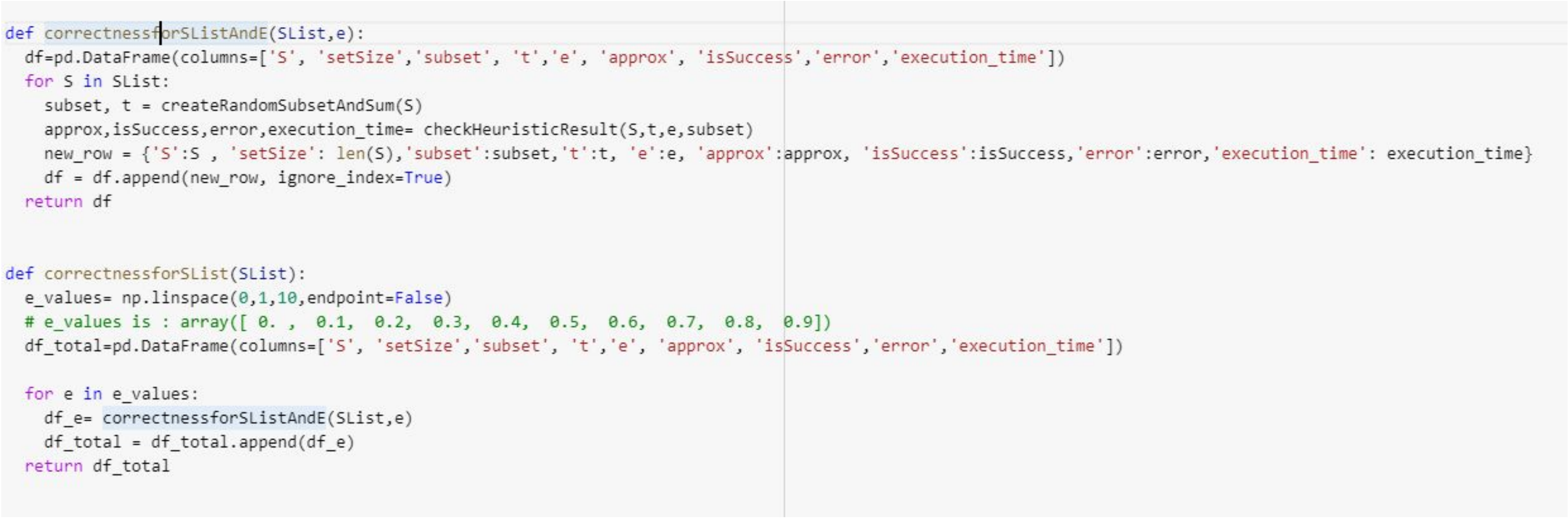
What About Running Time?

In order to compute running time of each case, we established some benchmark functions



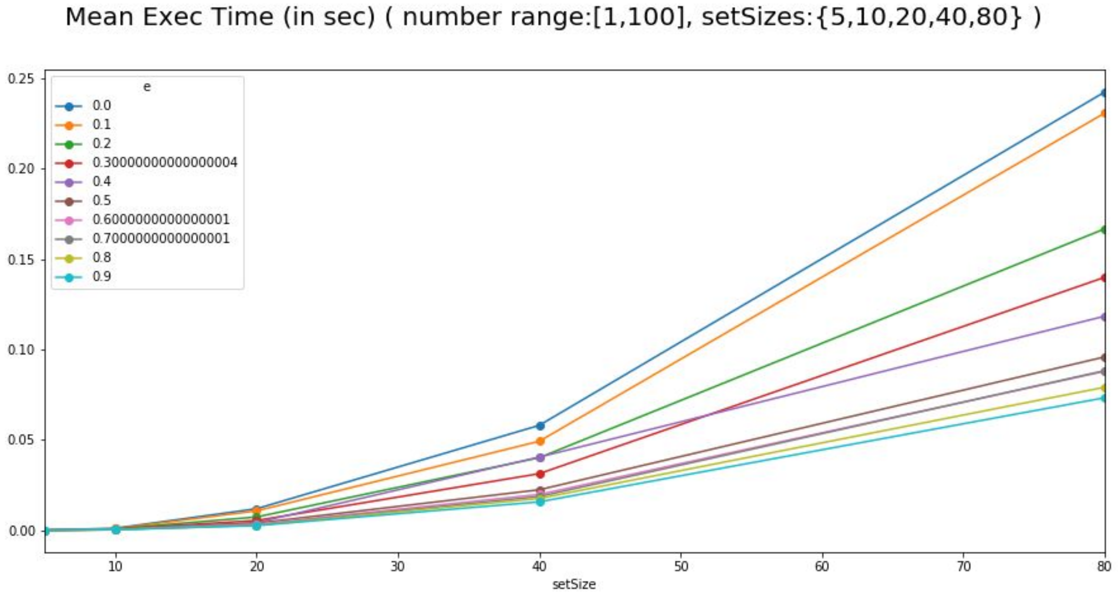
```
def checkHeuristicResult(S,t,e,subset):
  start_time = time.time()
  approx=appSubSet(S,t,e)
  execution_time= time.time() - start_time
  if (approx==t):
    success=True
    error=0
  else:
    success=False
    error= (t-approx)/t
  return approx,success,error,execution_time
```

What About Running Time?



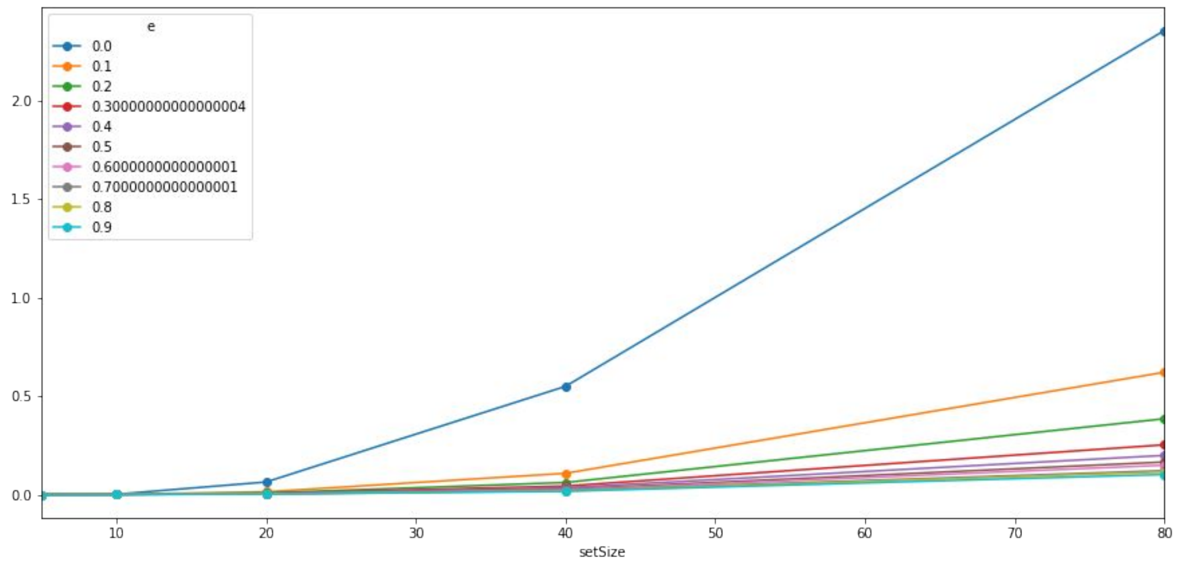
```python
def correctnessforSListAndE(SList,e):
  df=pd.DataFrame(columns=['S', 'setSize','subset', 't','e', 'approx', 'isSuccess','error','execution_time'])
  for S in SList:
    subset, t = createRandomSubsetAndSum(S)
    approx,isSuccess,error,execution_time= checkHeuristicResult(S,t,e,subset)
    new_row = {'S':S , 'setSize': len(S),'subset':subset,'t':t, 'e':e, 'approx':approx, 'isSuccess':isSuccess,'error':error,'execution_time': execution_time}
    df = df.append(new_row, ignore_index=True)
  return df


def correctnessforSList(SList):
  e_values= np.linspace(0,1,10,endpoint=False)
  # e_values is : array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
  df_total=pd.DataFrame(columns=['S', 'setSize','subset', 't','e', 'approx', 'isSuccess','error','execution_time'])

  for e in e_values:
    df_e= correctnessforSListAndE(SList,e)
    df_total = df_total.append(df_e)
  return df_total
```

Range = 100



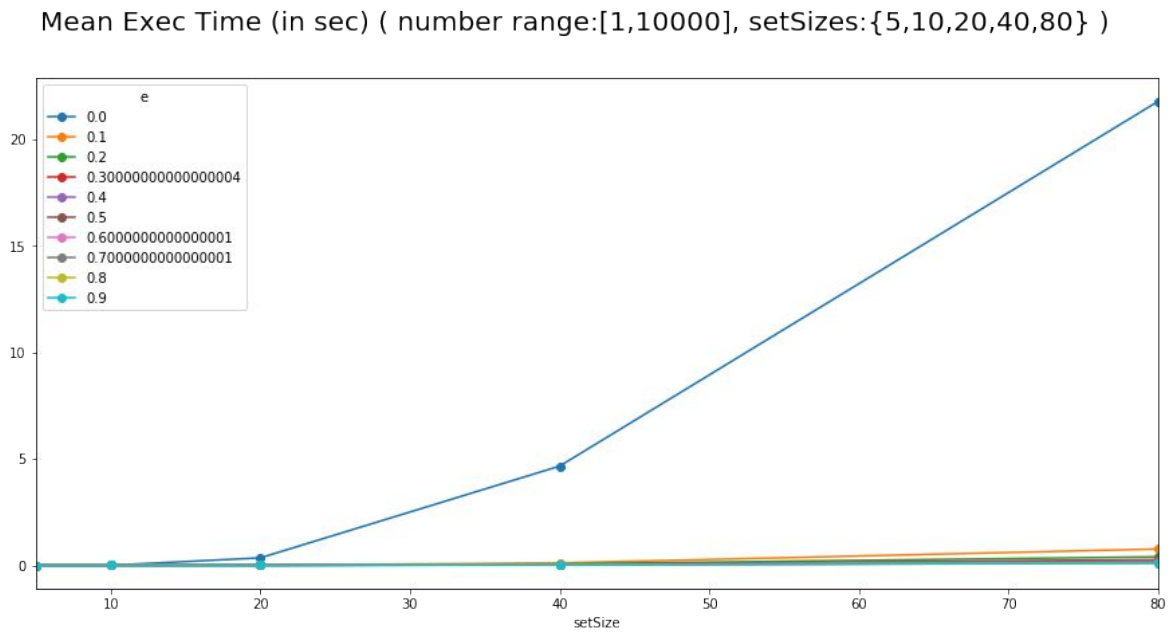
Mean Exec Time (in sec) (number range:[1,100], setSizes:{5,10,20,40,80})
e
0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6000000000000001
0.7000000000000001
0.8
0.9
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
50
60
70
80
setSize

Range = 1000



Mean Exec Time (in sec) (number range:[1,1000], setSizes:{5,10,20,40,80})

Range = 10000

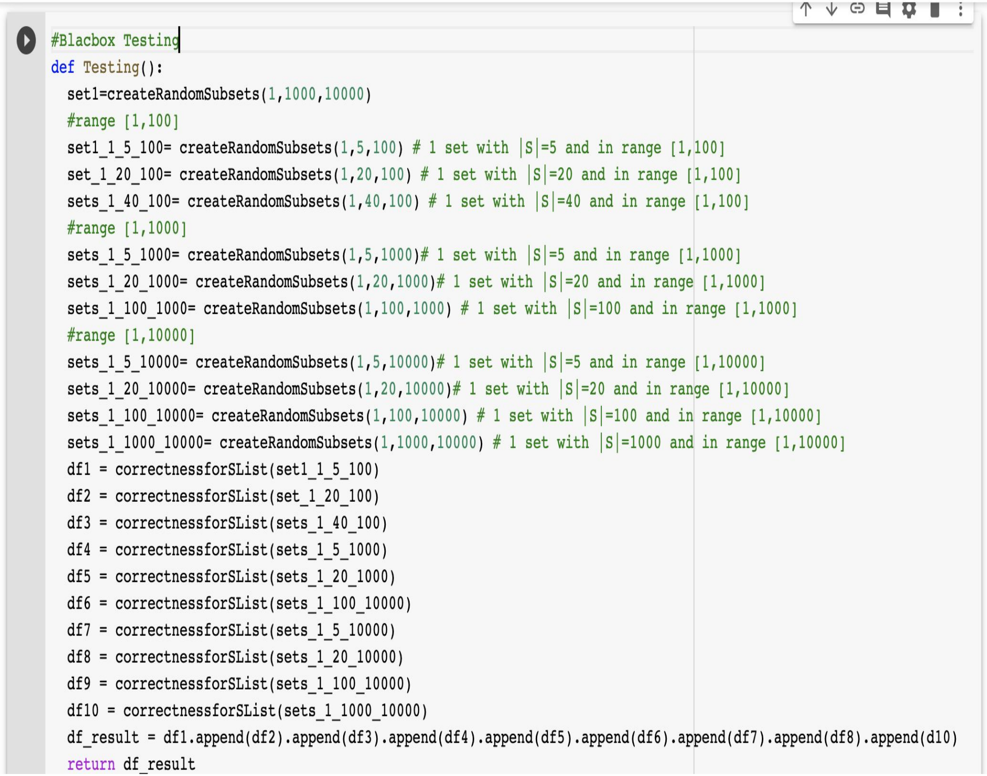

Mean Exec Time (in sec) (number range:[1,10000], setSizes:{5,10,20,40,80})
e
0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6000000000000001
0.7000000000000001
0.8
0.9
20
15
10
5
0
10
20
30
40
50
60
70
80
setSize

Testing

In order to test our algorithm,

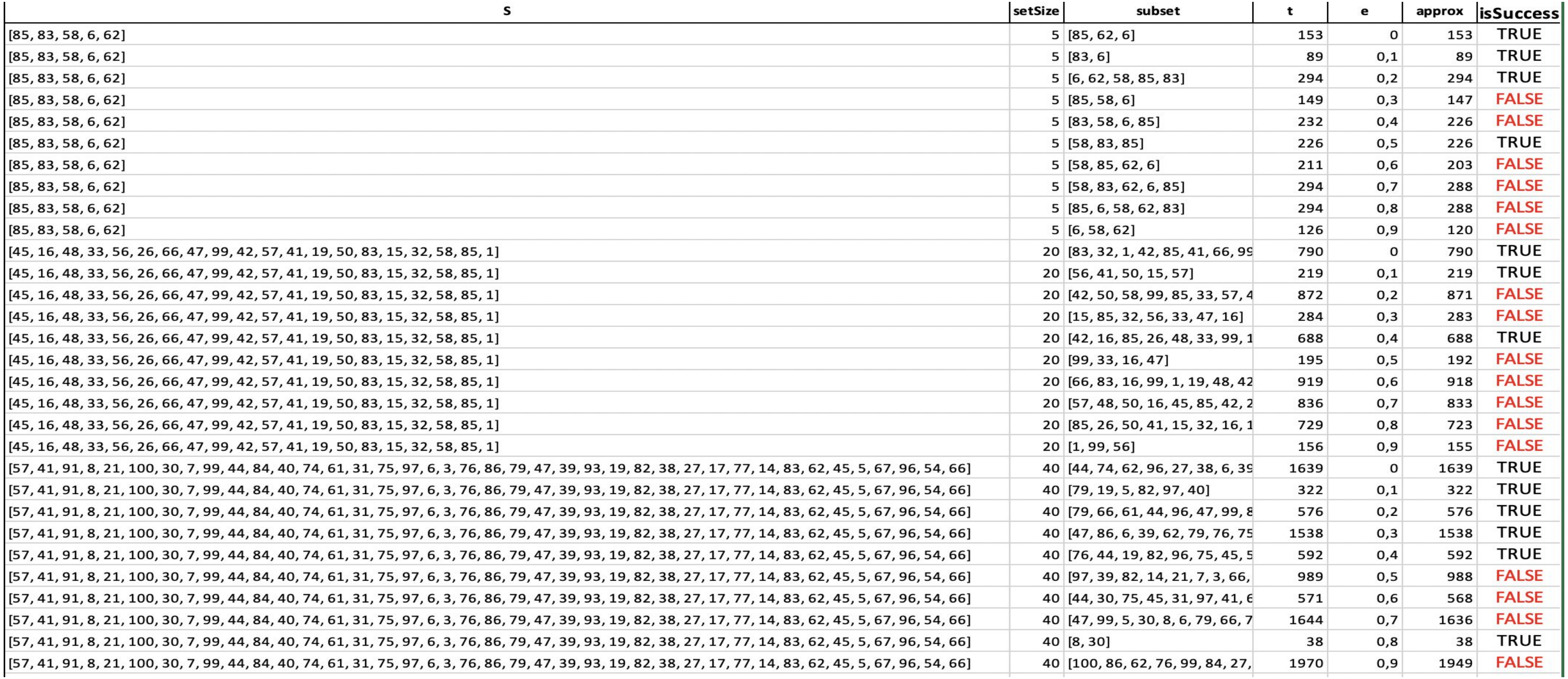
black-box technique is used.

- Randomly chosen sets in different range
- Error rates
- Number of elements in set
- Random t values



```
#Blacbox Testing
def Testing():
  set1=createRandomSubsets(1,1000,10000)
  #range [1,100]
  set1_1_5_100= createRandomSubsets(1,5,100) # 1 set with |S|=5 and in range [1,100]
  set_1_20_100= createRandomSubsets(1,20,100) # 1 set with |S|=20 and in range [1,100]
  sets_1_40_100= createRandomSubsets(1,40,100) # 1 set with |S|=40 and in range [1,100]
  #range [1,1000]
  sets_1_5_1000= createRandomSubsets(1,5,1000)# 1 set with |S|=5 and in range [1,1000]
  sets_1_20_1000= createRandomSubsets(1,20,1000)# 1 set with |S|=20 and in range [1,1000]
  sets_1_100_1000= createRandomSubsets(1,100,1000) # 1 set with |S|=100 and in range [1,1000]
  #range [1,10000]
  sets_1_5_10000= createRandomSubsets(1,5,10000)# 1 set with |S|=5 and in range [1,10000]
  sets_1_20_10000= createRandomSubsets(1,20,10000)# 1 set with |S|=20 and in range [1,10000]
  sets_1_100_10000= createRandomSubsets(1,100,10000) # 1 set with |S|=100 and in range [1,10000]
  sets_1_1000_10000= createRandomSubsets(1,1000,10000) # 1 set with |S|=1000 and in range [1,10000]
  df1 = correctnessforSList(set1_1_5_100)
  df2 = correctnessforSList(set_1_20_100)
  df3 = correctnessforSList(sets_1_40_100)
  df4 = correctnessforSList(sets_1_5_1000)
  df5 = correctnessforSList(sets_1_20_1000)
  df6 = correctnessforSList(sets_1_100_10000)
  df7 = correctnessforSList(sets_1_5_10000)
  df8 = correctnessforSList(sets_1_20_10000)
  df9 = correctnessforSList(sets_1_100_10000)
  df10 = correctnessforSList(sets_1_1000_10000)
  df_result = df1.append(df2).append(df3).append(df4).append(df5).append(df6).append(df7).append(df8).append(d10)
  return df_result
```

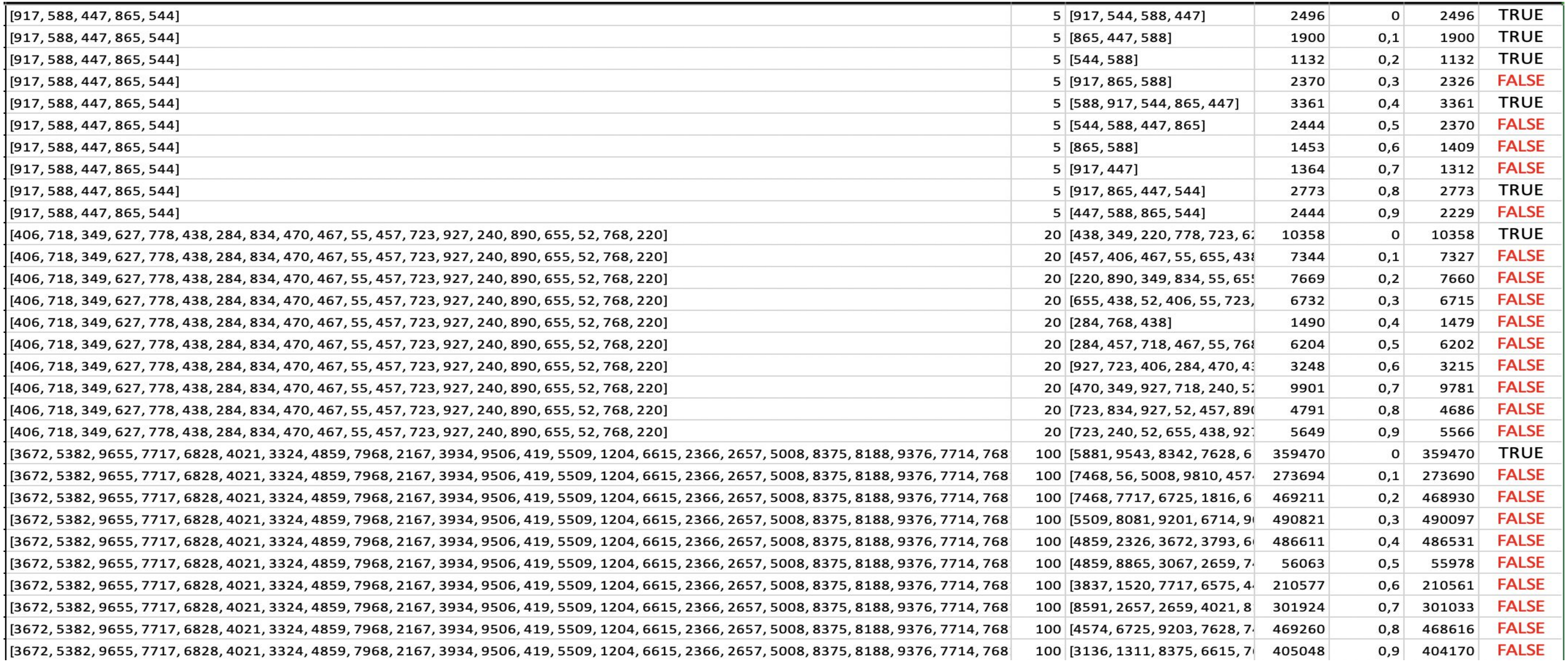
S=5, S=20, S=40 in range [0-100]



S	setSize	subset	t	e	approx	isSuccess
[85, 83, 58, 6, 62]	5	[85, 62, 6]	153	0	153	TRUE
[85, 83, 58, 6, 62]	5	[83, 6]	89	0,1	89	TRUE
[85, 83, 58, 6, 62]	5	[6, 62, 58, 85, 83]	294	0,2	294	TRUE
[85, 83, 58, 6, 62]	5	[85, 58, 6]	149	0,3	147	FALSE
[85, 83, 58, 6, 62]	5	[83, 58, 6, 85]	232	0,4	226	FALSE
[85, 83, 58, 6, 62]	5	[58, 83, 85]	226	0,5	226	TRUE
[85, 83, 58, 6, 62]	5	[58, 85, 62, 6]	211	0,6	203	FALSE
[85, 83, 58, 6, 62]	5	[58, 83, 62, 6, 85]	294	0,7	288	FALSE
[85, 83, 58, 6, 62]	5	[85, 6, 58, 62, 83]	294	0,8	288	FALSE
[85, 83, 58, 6, 62]	5	[6, 58, 62]	126	0,9	120	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[83, 32, 1, 42, 85, 41, 66, 99	790	0	790	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[56, 41, 50, 15, 57]	219	0,1	219	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[42, 50, 58, 99, 85, 33, 57, 4	872	0,2	871	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[15, 85, 32, 56, 33, 47, 16]	284	0,3	283	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[42, 16, 85, 26, 48, 33, 99, 1	688	0,4	688	TRUE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[99, 33, 16, 47]	195	0,5	192	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[66, 83, 16, 99, 1, 19, 48, 42	919	0,6	918	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[57, 48, 50, 16, 45, 85, 42, 2	836	0,7	833	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[85, 26, 50, 41, 15, 32, 16, 1	729	0,8	723	FALSE
[45, 16, 48, 33, 56, 26, 66, 47, 99, 42, 57, 41, 19, 50, 83, 15, 32, 58, 85, 1]	20	[1, 99, 56]	156	0,9	155	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[44, 74, 62, 96, 27, 38, 6, 39	1639	0	1639	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[79, 19, 5, 82, 97, 40]	322	0,1	322	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[79, 66, 61, 44, 96, 47, 99, 8	576	0,2	576	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[47, 86, 6, 39, 62, 79, 76, 75	1538	0,3	1538	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[76, 44, 19, 82, 96, 75, 45, 5	592	0,4	592	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[97, 39, 82, 14, 21, 7, 3, 66,	989	0,5	988	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[44, 30, 75, 45, 31, 97, 41, 6	571	0,6	568	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[47, 99, 5, 30, 8, 6, 79, 66, 7	1644	0,7	1636	FALSE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[8, 30]	38	0,8	38	TRUE
[57, 41, 91, 8, 21, 100, 30, 7, 99, 44, 84, 40, 74, 61, 31, 75, 97, 6, 3, 76, 86, 79, 47, 39, 93, 19, 82, 38, 27, 17, 77, 14, 83, 62, 45, 5, 67, 96, 54, 66]	40	[100, 86, 62, 76, 99, 84, 27,	1970	0,9	1949	FALSE

Figure : S=5, S=20, S=40 in range 0-100

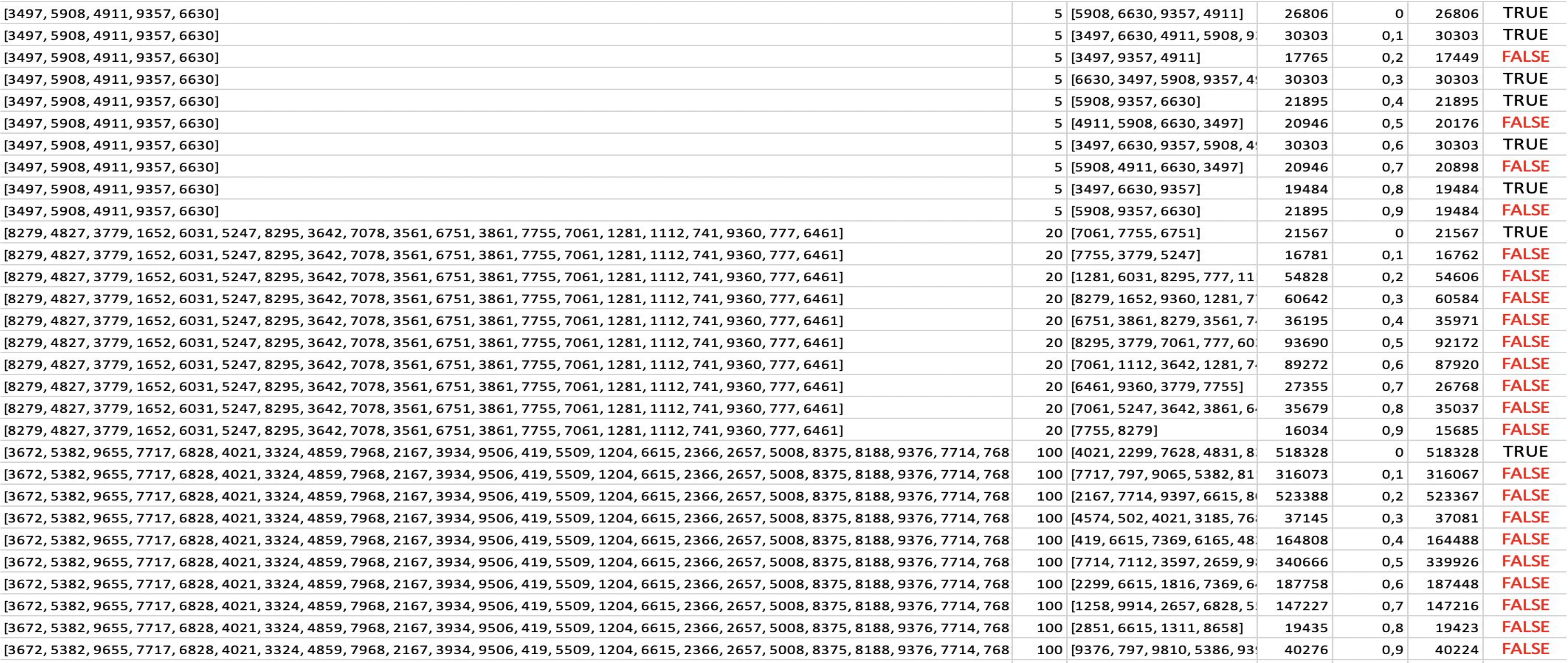
Testing



[917, 588, 447, 865, 544]	5	[917, 544, 588, 447]	2496	0	2496	TRUE
[917, 588, 447, 865, 544]	5	[865, 447, 588]	1900	0,1	1900	TRUE
[917, 588, 447, 865, 544]	5	[544, 588]	1132	0,2	1132	TRUE
[917, 588, 447, 865, 544]	5	[917, 865, 588]	2370	0,3	2326	FALSE
[917, 588, 447, 865, 544]	5	[588, 917, 544, 865, 447]	3361	0,4	3361	TRUE
[917, 588, 447, 865, 544]	5	[544, 588, 447, 865]	2444	0,5	2370	FALSE
[917, 588, 447, 865, 544]	5	[865, 588]	1453	0,6	1409	FALSE
[917, 588, 447, 865, 544]	5	[917, 447]	1364	0,7	1312	FALSE
[917, 588, 447, 865, 544]	5	[917, 865, 447, 544]	2773	0,8	2773	TRUE
[917, 588, 447, 865, 544]	5	[447, 588, 865, 544]	2444	0,9	2229	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[438, 349, 220, 778, 723, 6	10358	0	10358	TRUE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[457, 406, 467, 55, 655, 43	7344	0,1	7327	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[220, 890, 349, 834, 55, 65	7669	0,2	7660	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[655, 438, 52, 406, 55, 723,	6732	0,3	6715	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[284, 768, 438]	1490	0,4	1479	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[284, 457, 718, 467, 55, 76	6204	0,5	6202	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[927, 723, 406, 284, 470, 4	3248	0,6	3215	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[470, 349, 927, 718, 240, 5	9901	0,7	9781	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[723, 834, 927, 52, 457, 89	4791	0,8	4686	FALSE
[406, 718, 349, 627, 778, 438, 284, 834, 470, 467, 55, 457, 723, 927, 240, 890, 655, 52, 768, 220]	20	[723, 240, 52, 655, 438, 92	5649	0,9	5566	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[5881, 9543, 8342, 7628, 6	359470	0	359470	TRUE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7468, 56, 5008, 9810, 457	273694	0,1	273690	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7468, 7717, 6725, 1816, 6	469211	0,2	468930	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[5509, 8081, 9201, 6714, 9	490821	0,3	490097	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4859, 2326, 3672, 3793, 6	486611	0,4	486531	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4859, 8865, 3067, 2659, 7	56063	0,5	55978	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[3837, 1520, 7717, 6575, 4	210577	0,6	210561	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[8591, 2657, 2659, 4021, 8	301924	0,7	301033	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4574, 6725, 9203, 7628, 7	469260	0,8	468616	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[3136, 1311, 8375, 6615, 7	405048	0,9	404170	FALSE

Figure : S=5, S=20, S=100 in range 0-1000

Testing



[3497, 5908, 4911, 9357, 6630]	5	[5908, 6630, 9357, 4911]	26806	0	26806	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 4911, 5908, 9	30303	0,1	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 9357, 4911]	17765	0,2	17449	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[6630, 3497, 5908, 9357, 4	30303	0,3	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 9357, 6630]	21895	0,4	21895	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[4911, 5908, 6630, 3497]	20946	0,5	20176	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 9357, 5908, 4	30303	0,6	30303	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 4911, 6630, 3497]	20946	0,7	20898	FALSE
[3497, 5908, 4911, 9357, 6630]	5	[3497, 6630, 9357]	19484	0,8	19484	TRUE
[3497, 5908, 4911, 9357, 6630]	5	[5908, 9357, 6630]	21895	0,9	19484	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 7755, 6751]	21567	0	21567	TRUE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7755, 3779, 5247]	16781	0,1	16762	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[1281, 6031, 8295, 777, 11	54828	0,2	54606	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[8279, 1652, 9360, 1281, 7	60642	0,3	60584	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[6751, 3861, 8279, 3561, 7	36195	0,4	35971	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[8295, 3779, 7061, 777, 60	93690	0,5	92172	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 1112, 3642, 1281, 7	89272	0,6	87920	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[6461, 9360, 3779, 7755]	27355	0,7	26768	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7061, 5247, 3642, 3861, 6	35679	0,8	35037	FALSE
[8279, 4827, 3779, 1652, 6031, 5247, 8295, 3642, 7078, 3561, 6751, 3861, 7755, 7061, 1281, 1112, 741, 9360, 777, 6461]	20	[7755, 8279]	16034	0,9	15685	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4021, 2299, 7628, 4831, 8	518328	0	518328	TRUE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7717, 797, 9065, 5382, 81	316073	0,1	316067	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2167, 7714, 9397, 6615, 8	523388	0,2	523367	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[4574, 502, 4021, 3185, 76	37145	0,3	37081	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[419, 6615, 7369, 6165, 48	164808	0,4	164488	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[7714, 7112, 3597, 2659, 9	340666	0,5	339926	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2299, 6615, 1816, 7369, 6	187758	0,6	187448	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[1258, 9914, 2657, 6828, 5	147227	0,7	147216	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[2851, 6615, 1311, 8658]	19435	0,8	19423	FALSE
[3672, 5382, 9655, 7717, 6828, 4021, 3324, 4859, 7968, 2167, 3934, 9506, 419, 5509, 1204, 6615, 2366, 2657, 5008, 8375, 8188, 9376, 7714, 768	100	[9376, 797, 9810, 5386, 93	40276	0,9	40224	FALSE

Figure : S=5, S=20, S=100 in range 0-10000

Conclusion

- Subset Sum is a NP-Complete problem reduced from 3-SAT
- There is no fully correct linear time algorithm but approximate algorithm is linear time algorithm which is an approximate algorithm
- Chance of finding a correct solution to problem decreases by setsize and epsilon change
- Running time analysis of our algorithm gives consistent results which we can understand a linear time algorithm showed in incremental analysis part

References

https://www.youtube.com/watch?v=i8Kt9IBZ8FU
https://www.youtube.com/watch?v=k8RkYp5KhhU

Cormen, T. (2009). Introduction to algorithms. Cambridge, Mass.: MIT Press.

Tardos, E. (2015) Reduction from 3 SAT to Subset Sum.Retrieved from:https://www.cs.cornell.edu/courses/cs4820/2015sp/notes/reduction-subsetsum.pdf