1. 1. Pseudo

Knapsack(n, W, wt, val){ // dynamic

V = [][]

for w=0 to W{

V[0, w] = 0

}

for i=1 to n{

V[i, 0] = 0

for w=1 to W{

if(){

if(){

}

else{

V[i, w]=V[i-1, w]

}

}

else{

V[i, w] = V[i-1, w]

}

}

}

}

Knapsack(W, n){ // recursive

//base

if(n=0 or W=0){

return0;

}

if(){

return Knapsack(W, n-1)

}

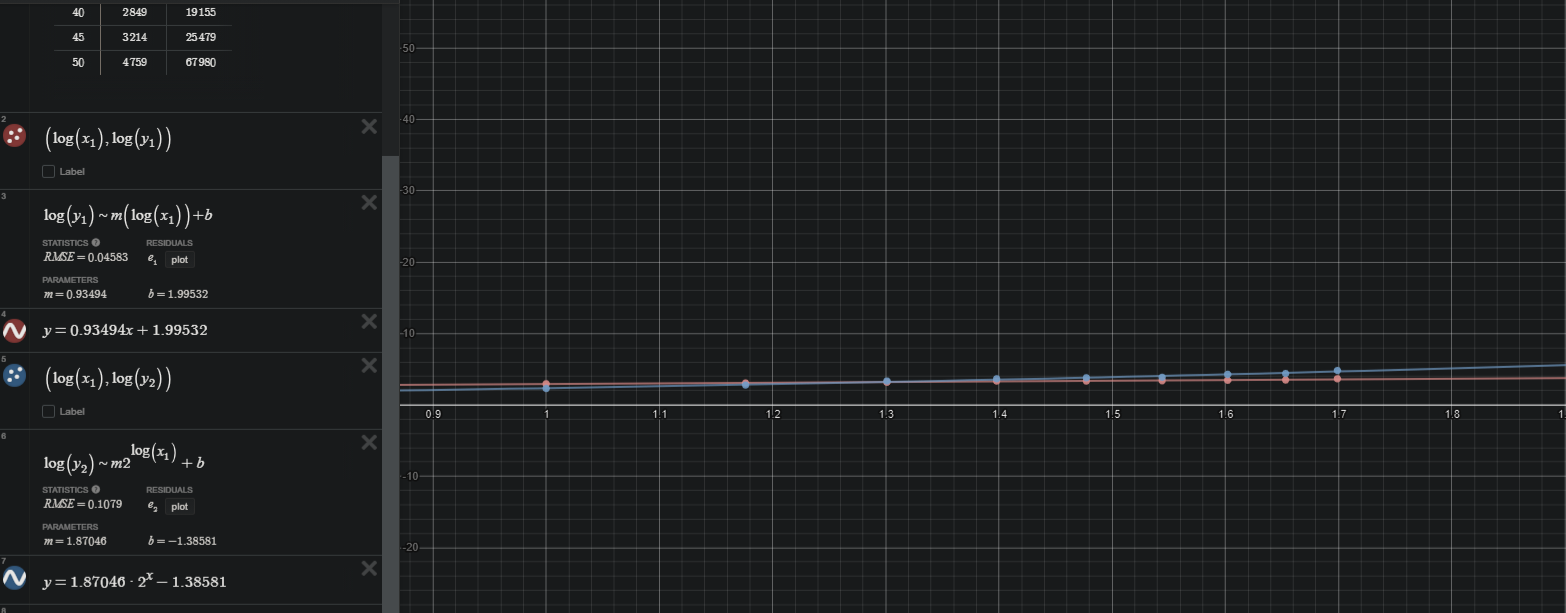
else{

return max(Knapsack(W-, n-1), Knapsack(W, n-1))

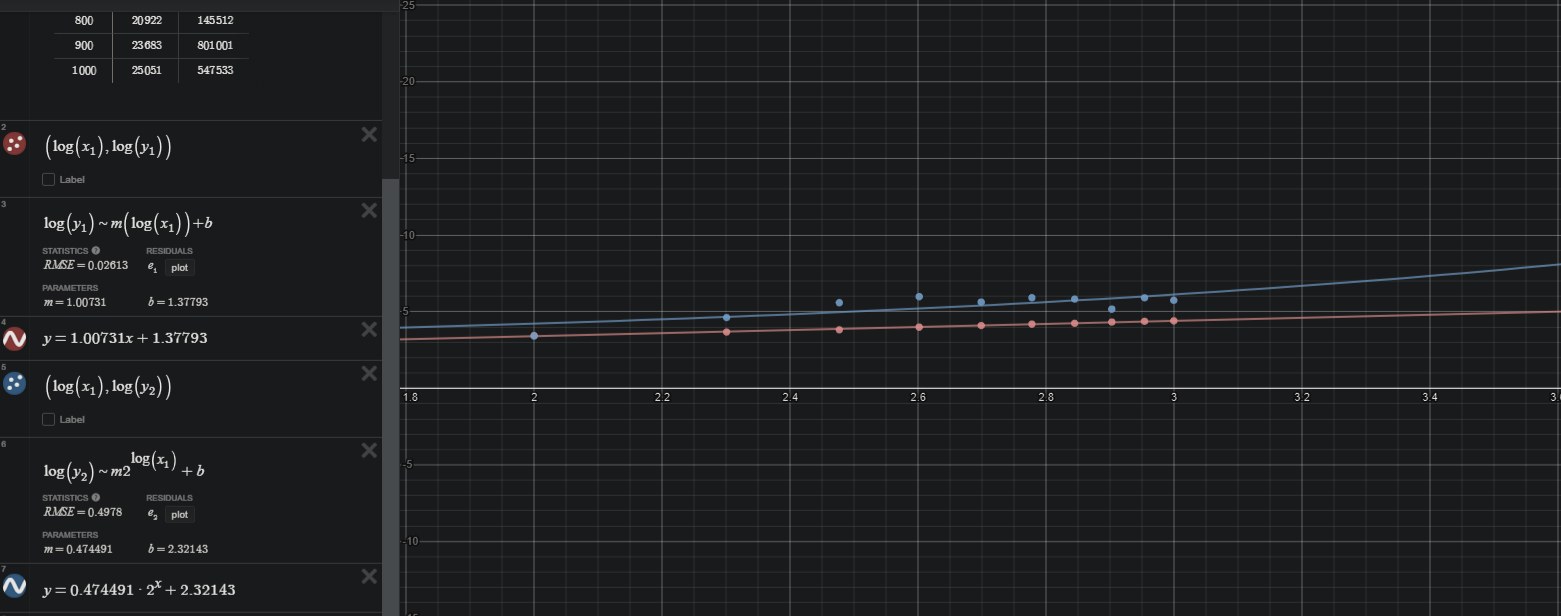
}

}

* 1. knapsack.py
  2. Gather data

N changing:

W changing:



* 1. I run the program, first varying n then W, start time, call the knapsack function, then end time and output in microseconds. I add those numbers into a sheets table and move them into desmos to plot and fit. The W recur it all over the place so much so that the line cant be fitted. In general W makes the run time much longer because of the more combinations it needs to try.

1. Shopping Spree
   1. Pseudo

// pseudo from above, change return to call unpack(n, W, wt, val, V)

unpack(n, W, wt, val, V){

Items = []

for i in n{

if(V[i,k] != V[i-1,k]){

i = i-1, k = k-wi

}

else{

i = i-1

}

}

Return items

}

* 1. Test case of N, F, for 1 i F:
     1. F \* O(n) => O(n)
  2. shopping.py