Guillermo Rivera
Project 3
CS 2223

For our project, we were asked to test out three programing methods and compare them. These methods were the Exhaustive, Dynamic and Greed methods. Using python 3.7.0a2 and the text editor IDLE, the methods were implemented and tested. After testing the methods with different values, the max value for each and time to execute were displayed. When it comes to an optimal solution, both Exhaustive and Dynamic deliver the ideal solution trying to return the possible highest value to be taken. On the other hand, the Greed algorithm since it is using floor division, it in some cases does not give the same max value as the other two methods. The better knapsack solution would be dynamic. The reason being that it is somewhat faster at computing the values, and it does not suffer the same fate as the greed method.
The efficiencies are calculated in the following pages.

Exhaustive search

Time eff.

A x = 0

nB while x < n

⋮

C x = 0

nD while x < Len(Perm)

nE x = x+1

⋮

n nF while itemIndex x < n

runs in $\Theta(n^2)$

$A + nB + C + nD + nE + nnF$

$\underbrace{A+C}_{K_1} + \underbrace{n(B+D+E}_{K_2} + \underbrace{nF}_{K_3})$

Space eff

A exhaust(B C D
(, W, V)

while
while
while → Item index = Item index+1

Space exhaustive $\Theta(n^2)$

Dynamic

time eff

for i . . .
    for w . . .

i · w

$O(i \cdot W)$

if we use n

$O(n^2)$

---

space eff

for i in range
    for w in range
        K[i][w]=0

$O(n^2)$

Greed

Time eff

n while
n for

$O(n^2)$

Space

n while
n for
if
nmax_i = item[0]

$O(n^3)$

```
#CS2223 Project 3
#Guillermo Rivera

#import
import time
import sys
import math
import copy
import ast
from itertools import permutations

print("Welcome to Guillermo's Closest Pair program")
print("Please follow the on screen Instructions")
print(" ")
i=0

def dynamic(capacity,weights,values):#Dynamic Programming
    n=len(weights)
    K = [[0 for x in range((capacity) + 1)] for x in range(n + 1)]
    items=[]
    for i in range(n + 1):
        for w in range(capacity + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif (weights[i - 1]) <= w:
                K[i][w] = max((values[i - 1]) + K[i - 1][w - ( weights[i - 1])], K[i - 1][w])
            else:
                K[i][w] = K[i - 1][w]
    return values,K[n][capacity],values

def exhaust(capacity,weights,values):#Exhaustive Search
    n=len(weights)
    itemResult=[]
    x=0
    array=[]
    lastAttempt=[]
    while x < n:
        w=(weights[x])
        v=(values[x])
        thingsappend=(w,v)
        array.append(thingsappend)
        x=x+1
    perm=list(permutations(array))
    highestValue=0
    highestWeight=0
    x=0
    while x< len(perm):
```

```
        attempt=perm[x]
        x=x+1
        itemIndex=0
        runningWeight=0
        runningValue=0
        while itemIndex<n:
            if(((attempt[itemIndex][0])+runningWeight)<=(capacity)):
                runningWeight=runningWeight+(attempt[itemIndex][0])
                runningValue=runningValue+(attempt[itemIndex][1])
                if(runningValue>=highestValue):
                    itemResult=attempt
                    highestValue=runningValue
                    highestWeight=runningWeight
                    itemResult=attempt[0:itemIndex+1]
            itemIndex=itemIndex+1
    result=(itemResult,highestValue,highestWeight)
    return result

def greed(capacity, weights, values):#Greed Programming
    value = 0
    valuePerWeight = valuePerWeight = sorted([[v // w, w] for v,w in zip(values,weights)],
reverse=True)
    while capacity > 0 and valuePerWeight:
        maxi = 0
        idx = None
        for i,item in enumerate(valuePerWeight):
            if item [1] > 0 and maxi < item [0]:
                maxi = item [0]
                idx = i

        if idx is None:
            return 0.
        v = valuePerWeight[idx][0]
        w = valuePerWeight[idx][1]
        if w <= capacity:
            value += v*w
            capacity -= w
        else:
            if w > 0:
                value += capacity * v
                return value
        valuePerWeight.pop(idx)
    return value

def effex(Capacity,weights,values):
    print('\nTesting Exhaustive Search\n')
    t0=time.time()
```

```
    resultex=exhaust(Capacity,weights,values)
    t1=time.time()
    T=t1-t0
    print('Selected Items: '+str(resultex[0]))
    print('Max Value: '+str(resultex[1]))
    print('Max Weight: '+str(resultex[2]))
    print('The answer was calculated in time: '+str(T)+' seconds.\n')

def effdy(Capacity,weights,values):
    print('\nTesting Dynamic Programming\n')
    t0=time.time()
    resultDP=dynamic(Capacity,weights,values)
    t1=time.time()
    T=t1-t0
    print('Max Value: ' + str(resultDP[1]))
    print('The answer was calculated in time: ' + str(T) + ' seconds.\n')

def effgr(Capacity,weights,values):
    print('\nTesting Greed Algorithm\n')
    t0=time.time()
    resultgreed=greed(Capacity,weights,values)
    t1=time.time()
    T=t1-t0
    print('Max Value: ' + str(resultgreed))
    print('The answer was calculated in time: ' + str(T) + ' seconds.\n')


def effAll(capacity,weights,values):
    effex(capacity,weights,values)
    effdy(capacity,weights,values)
    effgr(capacity,weights,values)
    return


while(True):
    userval= str(input("input the name of the file IE= 'input-1.txt' :"))
    print(" ")
    print("Your input was: "+userval+"\n Testing 3 methods...")
    print(" ")
    f = open(userval, "r")
    array = f.read()
    weights=[]
    values=[]
    temparray=[]
    capacity=0
    temparray=array.splitlines(0)
    capacity=temparray[0]
```

```
weights=temparray[1]
values=temparray[2]
weights = weights.split(',')
values = values.split(',')
setvalues = [int(v) for v in values]
setweights = [int(w) for w in weights]
capacity = int(capacity)
print ('\nThe input from ' + userval +' is: ')
print ('   Carry Capacity: ' +str(capacity))
print ('   Weights: ' +str(weights))
print ('   Values:  ' +str(values))
print ('   Number of items: '+str(len(weights)))
effAll(capacity,setweights,setvalues)
```