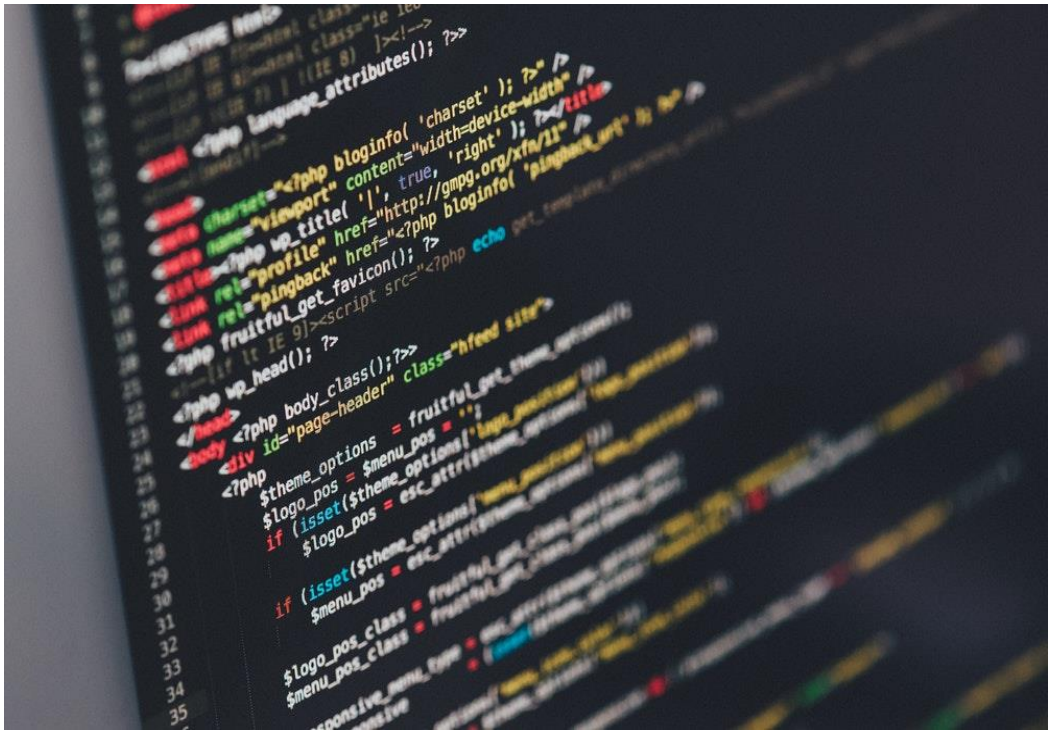
Project Report GEL103
(Dr. Sudarshan Iyengar)
Coupon Collector



SUBMITTED BY:

Karan Deep 2019csb1093

Kshitiz Arora 2019csb1095

Contents

I. Introduction

In the following project, we try to simulate the *coupon collector problem* starting from the very basic case of ‘one person choosing coupons from an infinite set of coupons (unbiased)’ to the seemingly real case of ‘r people taking part with a biased set of coupons’.

In the *coupon collector problem*, the goal is to purchase distinct objects in order to make a complete set of objects.

"Coupon" is just a placeholder word; the objects collected can be any kind of object.

II. A. Basic Case (1 person)

Conditions:

- Only one person collects coupons.
- Every coupon has an equal probability of selection.
- Selection of every coupon is independent of previous selections.

Time Complexity (Mathematical Proof):

Theory:

If the probability of occurrence of an event is equal to p.

$$P(E) = p$$

Then the number of turns after which the event occurs is $\frac{1}{p}$.

$$T(E) = \frac{1}{p}$$

Now, let us denote number of coupons by n.

Our final aim is to collect atleast one coupon of each type.

First, we want to select a coupon of any type (1st selection).

$P(S_1) = \frac{n}{n}$, because we can choose any coupon.

$$T(S_1) = \frac{n}{n}$$

Next, we want to select a coupon of any type other than the one already selected (2nd selection).

$$P(S_2) = \frac{n - 1}{n}$$

$$T(S_2) = \frac{n}{n - 1}$$

Similarly, for r^{th}selection,

$$P(S_r) = \frac{n - r + 1}{n}$$

$$T(S_r) = \frac{n}{n - r + 1}$$

Now to collect all coupons, total number of turns required are:

$$N_T = N(S_1) + N(S_2) + \ldots\ldots + N(S_r) + \ldots\ldots + N(S_n)$$

$$N_T = \sum_{i=1}^{n} N(S_i)$$

$$N_T = \sum_{i=1}^{n} \frac{n}{n - i + 1}$$

We can replace i by $n + 1 - i$

$$N_T = \sum_{i=1}^{n} \frac{n}{n - (n + 1 - i) + 1}$$

$$N_T = \sum_{i=1}^{n} \frac{n}{i}$$

$$N_T = n \cdot \sum_{i=1}^{n} \frac{1}{i}$$

$$N_T = n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots\ldots + \frac{1}{n}\right)$$

$$N_T \simeq nlog(n)$$

Therefore, the number of turns required to collect n coupons is equal to $nlog(n)$

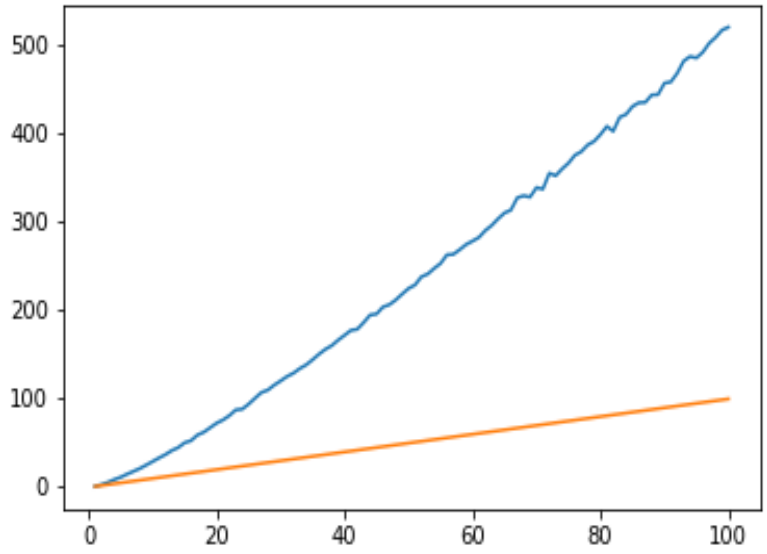
Hence, time complexity is of the order $O(nlogn)$.

Code To Check Time Complexity (Number Of Turns):

```
import random

def simulate(n):
    l = []; count = 0
    while (len(l) < n) :
        count = count+1
        k = random.randint(1,n)
        if k not in l:
            l.append(k)
    return count

def avrg (n,a):
    av=0
    for i in range(a):
        av = av + simulate(n)
    avg = av/a
    return avg
```

Simulation:

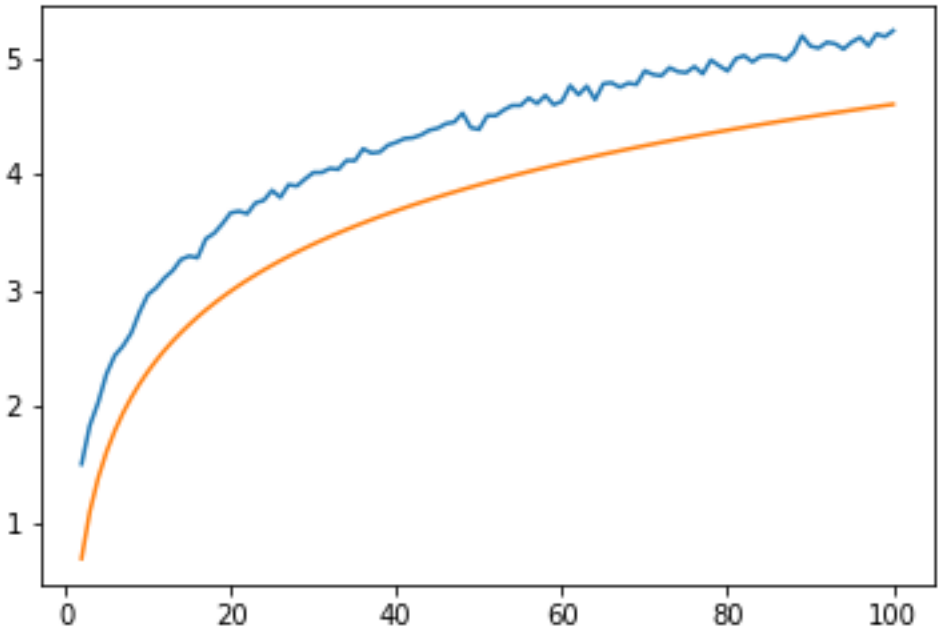
Now we varied the number of type of coupons n from 1 to 100 and plotted the average number of turns for 1000 iterations.



y = x

y = number of turns for x coupons

To confirm that this graph follows the $xlog(x)$ curve, also plotted the curve for number of turns divided by the number of coupons.



y = $logx$

y = (number of turns for x coupons)/x

From above graph we can confirm that the number of turns is of order $O(xlogx)$. ($i.e.\ N = constant * xlogx$

Also, we found that the constant factor to be equal to ***1.185***.

II.B. Practical Case Level 1

(1 person Biased coupons)

Conditions:

- Now, we have limited the number of coupons of each type to specific number (say, 100), but not necessarily equal.
- Only one person collects coupons.
- The number of coupons of each type vary, hence, the probability of selection of each type is different.
- Selection of every coupon depends on previous selections.

Code To Check Time Complexity (Number Of Turns):

```
import list_maker as lm #list_maker at last page
import random

def onep_dc(lst):
    n=len(lst)

    #defining a list containing the 'sample space'
    ss=[]
    for i in range(n):
        for j in range(lst[i]):
            ss.append(i+1)

    coup = []
    turns = 0
    while len(coup) != n:

        r=list(set(ss))
        if len(r)<n:
            turns = onep_dc(lst)
            break

        rand = random.choice(ss)
        ss.remove(rand)
```

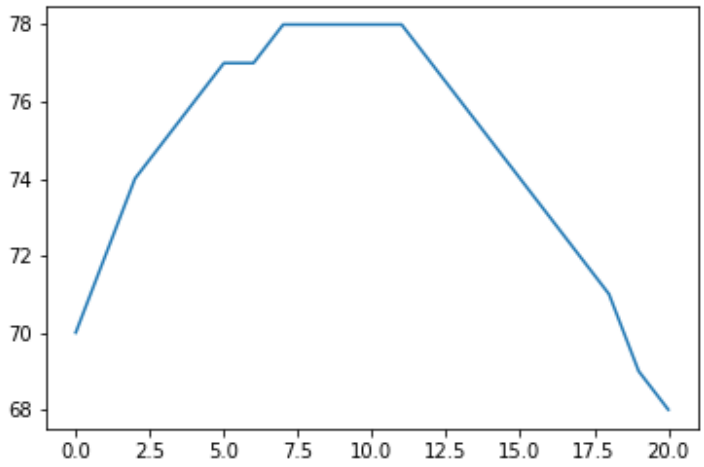
```
        turns += 1

        if rand not in coup:
            coup.append(rand)

    return turns

def avg_turns(lst, itr):
    avg = 0
    for i in range(itr):
        avg += onep_dc(lst)
    return avg / itr
```

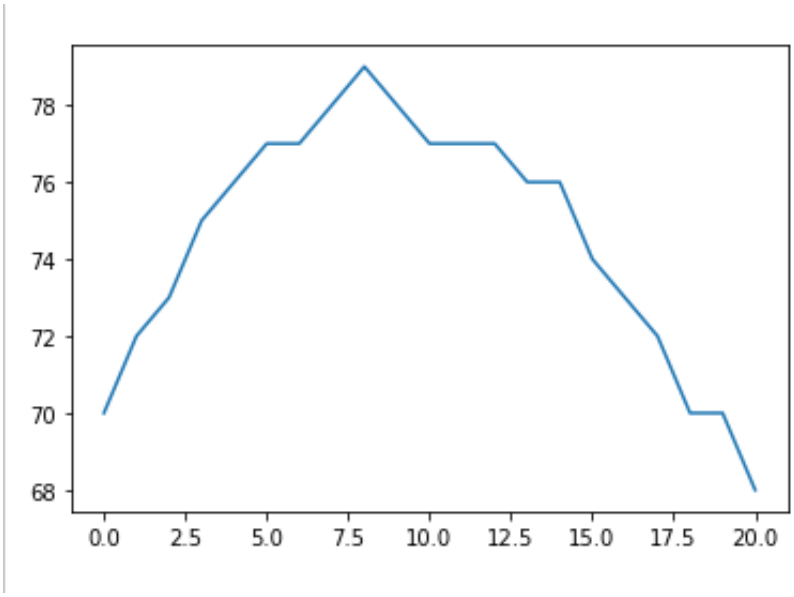
Simulation:

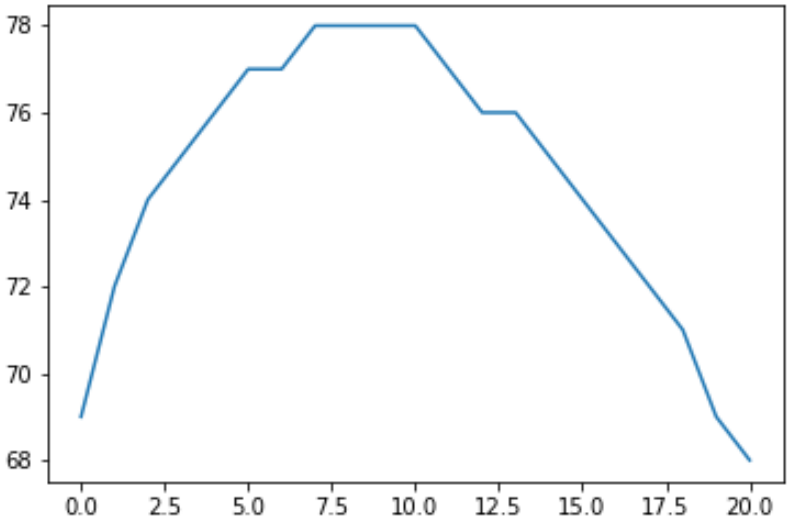
We took 20 different types of coupons, each 50 in number.

Then we started removing 20 coupons of one type at once until we removed 20 coupons of each type, i.e., 30 coupons of each type remained, and we plotted the curve for each step.

We repeated the above operation twice, and obtained the following curves:







Observation:

S. No.	Maximum Turns	Types of coupons decreased	Initial Turns	Final Turns
1	78	7,8,9,10,11	70	68
2	79	8	70	68
3	78	7,8,9,10	69	68

We observe that the graph attains maxima in the middle of the domain and then dips to the value lower than its initial value.

Possible explanation –

As we start removing coupons of each type, two factors get affected. First, the probability of selecting the reduced coupon decreases and, second, the total number of coupons also decrease.

In the first half, first factor is predominant and, hence, the number of turns increase but as almost half of the coupons are reduced we observe that the second factor comes into play and the number of turns remain almost constant, and as we progress further, the second factor outweighs the first one predominantly and we observe that the number of turns start decreasing rapidly and eventually at the end, it is less than the value at the beginning.

II.C. Practical Case Level 2

(R people Unbiased coupons)

Conditions:

- Here, we have limited the number of coupons of each type and all are equal.
- Now, consider that r people collect coupons (in order).
- The number of coupons of each type vary, hence, the probability of selection of each type is different.
- Selection of every coupon depends on previous selections.

Code To Check Time Complexity (Number Of Turns):

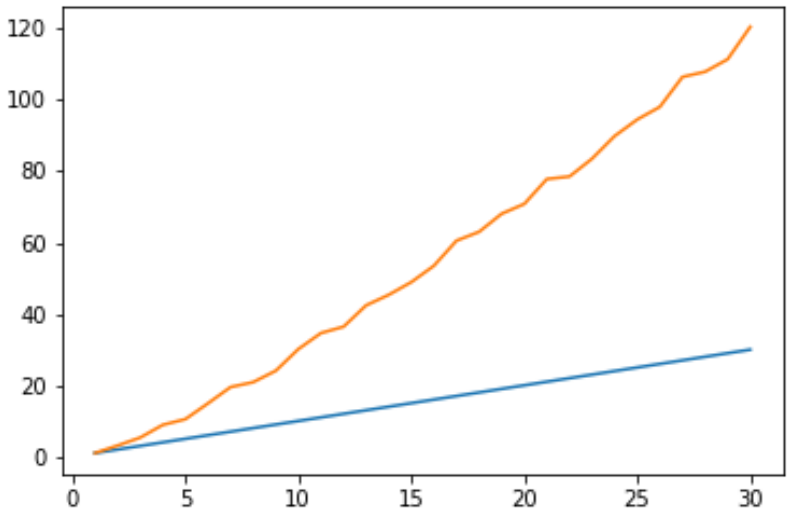
```
import random

def r_people_same_coupon_individual(l,r):
```

```
    n=len(l)



    #defining a list containing the 'sample space'
    k=[]
    for i in range(n):
        for j in range(l[i]):
            k.append(i+1)

    # making list for one person
    coup = []
    maal = []

    turn = 0;ind=0

    while(len(coup)!=n):
        rn=random.choice(k)
        k.remove(rn)
        if ind%r==0:
            turn += 1
            maal.append(rn)
            if rn not in coup:
                coup.append(rn)
        ind += 1

    return turn

"""a = no of iterations"""

def avg_turns(l, a, r):
    av=0
```

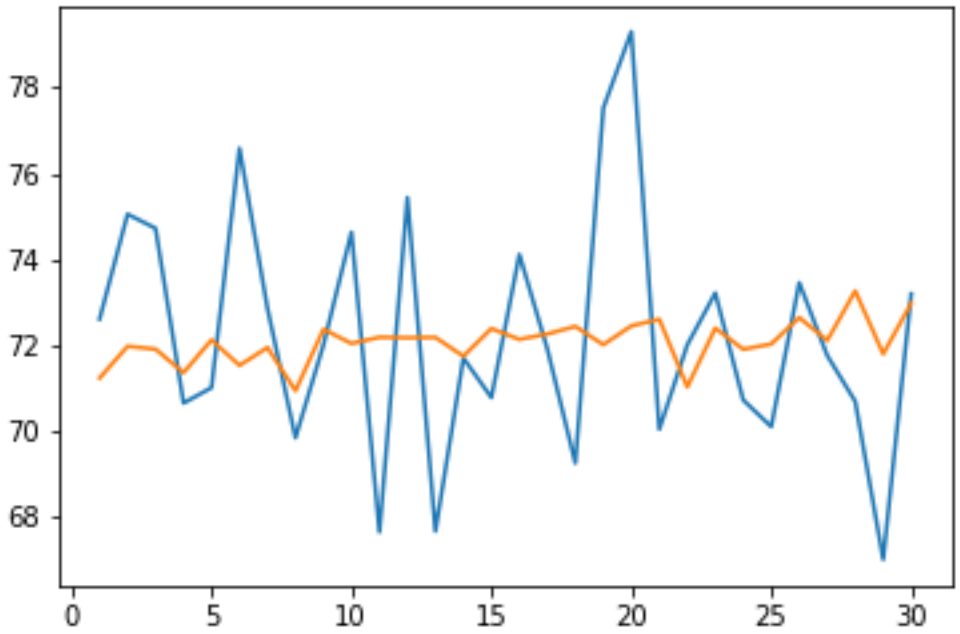
```
    for i in range(a):
        av = av + r_people_same_coupon_individual(l,r)
    avg = av/a
    return avg

def l_maker(n):
    l = []
    for i in range(n):
        l.append(500)
    return(l)
```

Simulation:

Case 1 – When the number of people is same (30) and we plot the graph for number of turns vs types of coupons (1 to 30). (compared with y=x)



Case 2 – When the types of coupons are same (20) and we plot the graph for number of turns vs number of people (1 to 30). (compared with the output of Basic Case)



II.D. Practical Case Level 3a

(R people Biased coupon)

(Customer's Perspective)

Conditions:

- Here, we have limited the number of coupons of each type and all aren't necessarily equal.
- We consider that r people collect coupons (in order).
- The number of coupons of each type vary, hence, the probability of selection of each type is different.
- Selection of every coupon depends on previous selections.

Customer's Perspective means that we are interested in knowing the number of turns after which a specific person collects all coupons, following above conditions.

Code To Check Time Complexity (Number Of Turns):

```
import list_maker as lm
import random

def np_dc_cus(lst, r):
    n=len(lst)

    #defining a list containing the 'sample space'
    ss=[]
    for i in range(n):
        for j in range(lst[i]):
            ss.append(i+1)

    coup = []
    turns = 0
    while len(coup) != n:

        var=list(set(ss))
        if len(var)<n:
            turns = np_dc_cus(lst, r)
            break

        rand = random.choice(ss)
        ss.remove(rand)
```

```
        turns += 1
        if rand not in coup:
            coup.append(rand)

        # for other people
        for i in range(r-1):
            a = random.choice(ss)
            ss.remove(a)

    return turns

def avg_turns(lst, r, itr):
    avg = 0
    for i in range(itr):
        avg += np_dc_cus(lst, r)
    return avg / itr
```

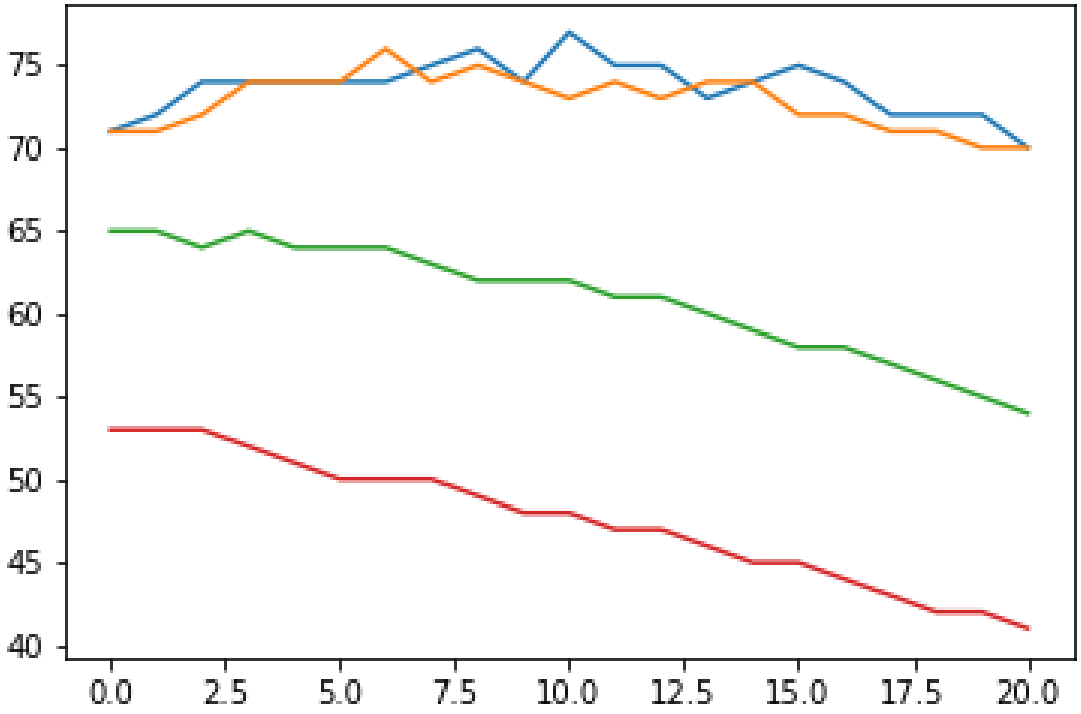
Simulation:

We took 20 different types of coupons, each 100 in number.

Then we remove 30 coupons each time until we have 70 coupons of each type left.

Now we plot the graph between the number of turns and number of biased coupons,

when the number of people other than the customer are 0, 10, 20, 30.



Observation:

We observe that as the number of people increases to 10 there is no significant change (similar to II.C.) but as it increases to 20 or 30 the number of turns decrease. On the other hand, effect change in number of coupons is also significant for 20 or 30 people.

II.E. Practical Case Level 3b

(R people Biased coupons)

(Seller's Perspective)

Conditions:

- Here, we have limited the number of coupons of each type and all aren't necessarily equal.
- We consider that r people collect coupons (in order).
- The number of coupons of each type vary, hence, the probability of selection of each type is different.
- Selection of every coupon depends on previous selections.

Seller's Perspective means that we are interested in knowing the number of turns after which any person (among all that take part) collects all coupons, following above conditions.

Code To Check Time Complexity (Number Of Turns):

```
import list_maker as lm
import random

def np_dc_shop(lst, r):
    n=len(lst)

    #defining a list containing the 'sample space'
    ss=[]
    for i in range(n):
        for j in range(lst[i]):
            ss.append(i+1)

    coupons = []
    for i in range(r):
        l=[]
        coupons.append(l)

    mx=0
    turns = 0
    while mx<n:

        var=list(set(ss))
        if len(var)<n:
            turns = np_dc_shop(lst, r)
            break
```

```
        for i in range(r):
            rand = random.choice(ss)
            ss.remove(rand)
            if rand not in coupons[i]:
                coupons[i].append(rand)

        mx=0

        for i in range(len(coupons)):
            if mx < len(coupons[i]):
                mx = len(coupons[i])

        turns += 1

    return turns

def avg_turns(lst, r, itr):
    avg = 0
    for i in range(itr):
        avg += np_dc_shop(lst, r)
    return avg / itr
```

Simulation:

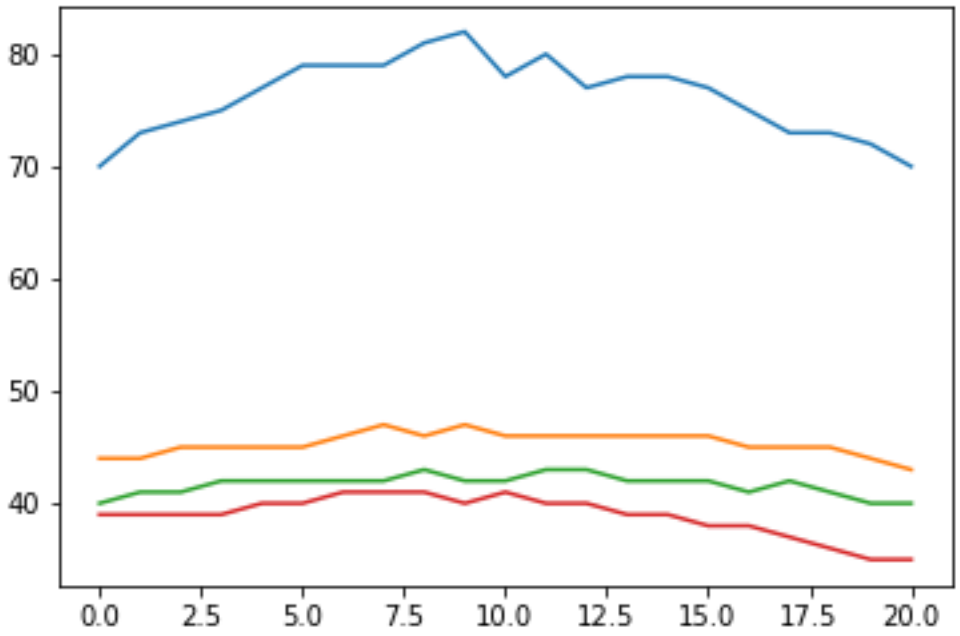
We took 20 different types of coupons, each 100 in number.

Then we remove 40 coupons each time until we have 60 coupons of each type left.

Now we plot the graph between the number of turns and number of biased coupons,

when the number of people other than the customer are 0, 10, 20, 30.

This is similar to last simulation except for the fact that here we are using the minimum number of turns after which any one person wins rather than the turns after which a specific person wins.



Observation:

This plot is similar to the previous one (II.D.) except for the fact that the average turns are less as we focused only on the first win.

Code for **list_maker**:

```
#defining a function to easily make a list to input
in next function

#....n is the number of different coupons
#....max_c is the initial number of coupons of each
type
#....t_err is the total error in number of coupons
#....t is the number of type of coupons to be
affected

def lst(n,max_c,t_err,t):

    l=[int(max_c) for x in range(n)]

    if t>0:
        err=t_err/t
```

```
        #t=x denotse k/x less coupons of x type
        for i in range(t):
            l[-1-i]-=int(err)

    return l
```