LAB 4

2. n^2 – goes thru n lines n (shoppers) times

3. def loadBalancer(n):

array = [0]\*n

for x in range(n):

idx = rnd.randint(0,n-1)

array[idx] += 1

return max(array)

4. n – goes thru n people 1ce

5. risk of all inputs going to one place

6. longest line = n, Pr[n-length line] = (1/(n^(n-1))) b/c (1/n) for given line (to the n) for n shoppers, (times n) for n lines

7. def loadBalancerPlot():

mean\_list = []

for i in range(3, 501):

max\_list = [0]\*1000

for j in range(1000):

array = loadBalancer(i)

max\_list[j] = max(array)

mean\_list += [np.mean(max\_list)]

plt.figure(1)

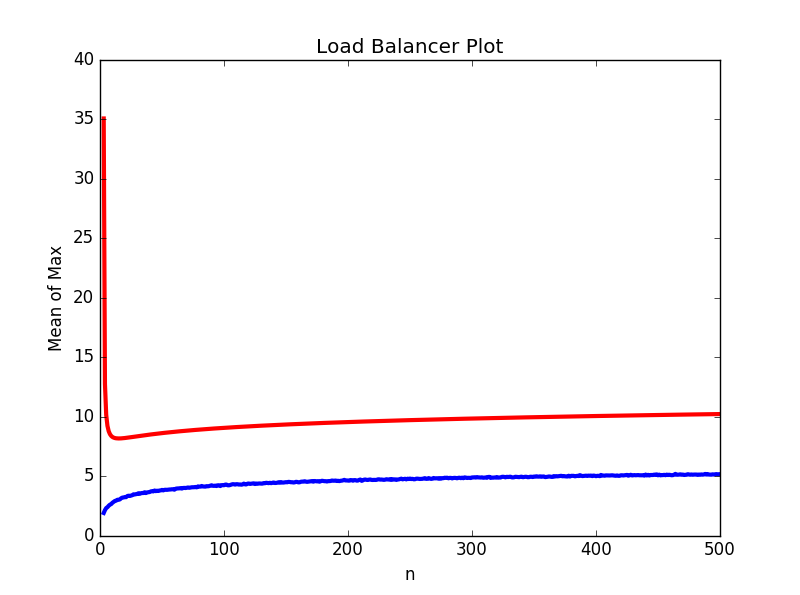
plt.plot(range(3,501), mean\_list)

plt.xlabel("n")

plt.ylabel("Mean of Max")

plt.title("Load Balancer Plot")

plt.show()



The empirical data never gets higher than the theoretical bound.

8. def loadBalancer(n, d):

array = [0]\*n

for x in range(n):

number = []

for y in range(d):

idx = rnd.randint(0,n-1)

number += [array[idx]]

array[array.index(min(number))] += 1

return max(array)

9. Theoretically, if d and n were the same, we would get the same runtime as the deterministicLoadBalancer(n) function and the max should be 1 every time.

10. def loadBalancerPlot2():

mean\_list = []

y\_coord = []

for i in range(2, 11):

max\_list = [0]\*1000

for j in range(1000):

max\_array = loadBalancer(1000, i)

max\_list[j] = max\_array

mean\_list += [np.mean(max\_list)]

y\_coord += [((math.log(math.log(1000)))/(math.log(i)))+2]

plt.figure(1)

plt.plot(range(2,11), mean\_list, linewidth = 3)

plt.draw()

x\_coord = range(2, 11)

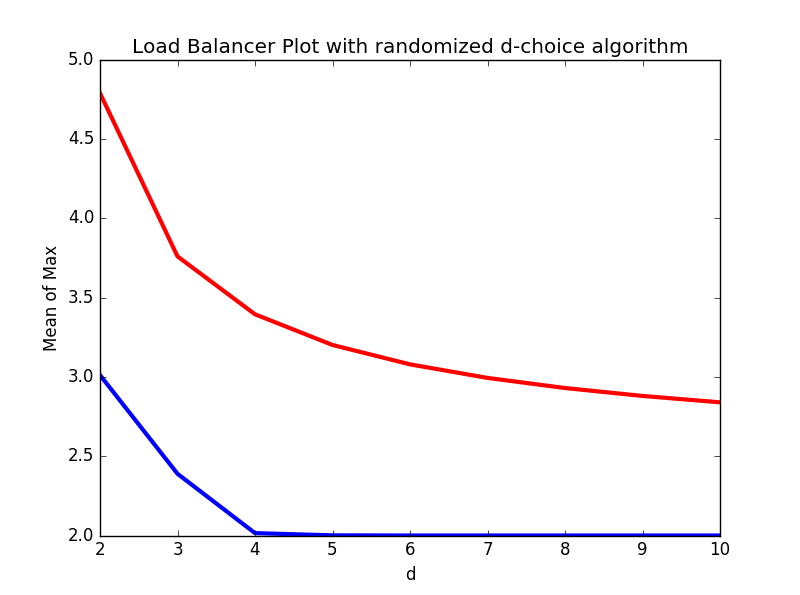
plt.plot(x\_coord, y\_coord, color = 'r', linewidth = 3)

plt.xlabel("d")

plt.ylabel("Mean of Max")

plt.title("Load Balancer Plot with randomized d-choice algorithm")

plt.show()



11. The empirical data line flattens out quickly because as d increases, we are checking more lines and picking the smallest of more lines. This increases our chances of getting a shorter line.

12. vd = 3

13. F2 = 18