Programming Assignment (Probability):

1.
Average for 100 times: 7.06.
Maximum for 100 times: 85.33.
Average for 10000 times: 16.761933.
Maximum for 10000 times: 30037.33
Average for 1000000 times: 20.25170533.
Maximum for 1000000 times: 873813.33.
How much money would you pay for the chance to play this game?: $20.

2.
Switch win probability for 1000 times: 0.673.
No switch win probability for 1000 times: 0.323.
How many games do you need to simulate to start to see definitively which strategy is better?: 10+ simulations.
What is the actual (unestimated/exact) percentage that each strategy will win? Please give an explanation/derivation for this answer.:

> Switch: wins 66.67% of the time. No switch: wins 33.33% of the time. When you initially make your choice, you have a 1/3 chance of picking the car, and a 2/3 chance of picking a goat. When the host reveals a goat behind one of the other doors, the probability distribution changes, favoring the strategy of switching doors.

3.
i.

Attacker dice:	Defender dice:	Attacker loss %:	Defender loss %:
1,	1,	0.586,	0.414
1,	2,	0.7421,	0.2579
2,	1,	0.4136,	0.5864
2,	2,	1.215,	0.785
3,	1,	0.3385,	0.6615
3,	2,	0.9275,	1.0725

Is it ever advantageous for a player to roll less than the most dice they are allowed by the rules?: No.
ii.

Attacker armies:	Defender armies:	Attacker win%:	Defender win %:
2	5	0.0016	0.9984
3	5	0.0463	0.9537
4	5	0.1986	0.8014
5	5	0.3559	0.6441
6	5	0.5009	0.4991
7	5	0.6448	0.3552
8	5	0.7421	0.2579
9	5	0.8181	0.1819
10	5	0.8707	0.1293
11	5	0.9211	0.0789
12	5	0.9421	0.0579
13	5	0.9608	0.0392
14	5	0.9759	0.0241
15	5	0.9817	0.0183
16	5	0.99	0.01
17	5	0.9938	0.0062
18	5	0.9963	0.0037
19	5	0.9981	0.0019
20	5	0.9983	0.0017

What is the minimum number of armies the attacker needs to guarantee a 50% chance of winning the territory? 7.
How about to guarantee an 80% chance of winning? 10.
iii.

Remaining attackers:	Remaining defenders:	Probability of happening:
2,	0,	0.0295
3,	0,	0.0624
4,	0,	0.0985
5,	0,	0.0917
6,	0,	0.0796
7,	0,	0.0573
8,	0,	0.0379
9,	0,	0.0193
10,	0,	0.0057
1,	1,	0.0416
1,	2,	0.079
1,	3,	0.0871
1,	4,	0.0755
1,	5,	0.0716
1,	6,	0.0617
1,	7,	0.0477
1,	8,	0.0298
1,	9,	0.0173
1,	10,	0.0068

Code:

```
import random

def run (times):
  totalmoney = 0
  maxmoney = 0
  for i in range(times):
    tails = False
    count = 0
    while tails == False:
      count += 1
      result = random.randint(0, 1)
      if result == 1:
        tails = True
        money = 2**count
        totalmoney += money
        if money > maxmoney:
          maxmoney = money
  averagemoney = totalmoney/times

  return averagemoney, maxmoney


averagemoney1main = 0
maxmoney1main = 0
averagemoney2main = 0
maxmoney2main = 0
averagemoney3main = 0
maxmoney3main = 0

for i in range(3):
  averagemoney1, maxmoney1 = run(100)
  averagemoney1main += averagemoney1
  maxmoney1main += maxmoney1
  averagemoney2, maxmoney2 = run(10000)
  averagemoney2main += averagemoney2
  maxmoney2main += maxmoney2
  averagemoney3, maxmoney3 = run(1000000)
  averagemoney3main += averagemoney3
  maxmoney3main += maxmoney3

print(averagemoney1main/3)
print(maxmoney1main/3)
print(averagemoney2main/3)
print(maxmoney2main/3)
print(averagemoney3main/3)
print(maxmoney3main/3)

print('$20.')


import random

def play (times, switch):
  wins = 0
  for i in range(times):
    doors = [0, 1, 2]
    car = random.randint(0, 2)
    choose = random.randint(0, 2)
    doors.remove(car)
    if car != choose:
      doors.remove(choose)
    reveal = random.choice(doors)
    doors.remove(reveal)
    doors.append(car)
    if car != choose:
      doors.append(choose)
    if switch:
      doors.remove(choose)
      choose = doors[0]
    if car == choose:
      wins += 1
  return wins/times


switchyes = play(1000, True)
switchno = play(1000, False)

print(switchyes)
print(switchno)
print('10+ simulations.')
print('Switch: wins 66.67% of the time. No switch: wins 33.33% of the time. When you initially make your choice, you have a 1/3 chance of picking the car, and a 2/3 chance of picking a goat. When the host reveals a goat behind one of the other doors, the probability distribution changes, favoring the strategy of switching doors.')


import random
import bisect

def round(attackArmies, defendArmies, attackDice, defendDice):
  attackRolls = []
  defendRolls = []
  for j in range(attackDice):
    roll = random.randint(1, 6)
    index = bisect.bisect_left(attackRolls, roll)
    attackRolls.insert(index, roll)
  for j in range(defendDice):
    roll = random.randint(1, 6)
    index = bisect.bisect_left(defendRolls, roll)
    defendRolls.insert(index, roll)

  attackRolls = attackRolls[::-1]
  defendRolls = defendRolls[::-1]

  for i in range(min(attackDice, defendDice)):
    if attackRolls[i] > defendRolls[i]:
      defendArmies -= 1
    else:
      attackArmies -= 1

  return attackArmies, defendArmies

def battle(attackArmies, defendArmies):
  while attackArmies > 1 and defendArmies > 0:
    attackDice = min(3, attackArmies-1)
    defendDice = min(2, defendArmies)
    attackArmies, defendArmies = round(attackArmies, defendArmies, attackDice, defendDice)

  return attackArmies, defendArmies

def game1 (attackDice, defendDice, runs):
  totalAttackLosses = 0
  totalDefendLosses = 0
  for i in range(runs):
    attackArmies, defendArmies = round(0, 0, attackDice, defendDice)
    totalAttackLosses -= attackArmies
    totalDefendLosses -= defendArmies
  return attackDice, defendDice, totalAttackLosses/runs, totalDefendLosses/runs

def game2 (attackArmies0, defendArmies0, runs):
  attackWins = 0
  defendWins = 0

  for i in range(runs):
    attackArmies, defendArmies = battle(attackArmies0, defendArmies0)

    if (attackArmies == 1 and defendArmies > 0):
      defendWins += 1
    elif (defendArmies == 0 and attackArmies > 1):
      attackWins += 1

  return attackArmies0, defendArmies0, attackWins/runs, defendWins/runs

def game3 (attackArmies0, defendArmies0, runs):
  outcomes = []
  for i in range(2, attackArmies0+1):
    outcomes.append([i, 0, 0])
  for i in range(1, defendArmies0+1):
    outcomes.append([1, i, 0])

  for i in range(runs):
    attackArmies, defendArmies = battle(attackArmies0, defendArmies0)

    for j in range(len(outcomes)):
      if attackArmies == outcomes[j][0]:
        if defendArmies == outcomes[j][1]:
          outcomes[j][2] += 1

  for i in range(len(outcomes)):
    outcomes[i][2] = outcomes[i][2]/runs

  return outcomes


for i in range(1,4):
  for j in range(1,3):
    print(game1(i, j, 10000))
print('No.')


for i in range(2,21):
  print(game2(i, 5, 10000))
print('7.')
print('10.')


for element in game3(10, 10, 10000):
  print(element)
```