Qishi Quiz 3

Q1.

We can look at each seat individually and calculate the expectation of boy-girl neighbors for each seat. Let X_i=1 if the person in chair i and his/her neighbor in the counterclockwise direction has different sex (boy-girl neighbor). By the linearity of expectation, we can get the expectation of total boy-girl neighbors E(X) = N*E(X_1) since all the E(X_i)s are the same by symmetry.

Now we can calculate E(X_1) = the chance of girl sitting * the chance of boy sitting counterclockwise + the chance of boy sitting * the chance of girl sitting counterclockwise = 7/16*9/15 + 9/16*7/15 = 21/40, thus E(X) = 16*(21/40) =42/5

Q2.

Independent variables must be uncorrelated. Uncorrelated can be dependent. So independent is stronger.

Assume X ~ N (0,1). Y = X^2. X and Y are dependent:

Cov(X,Y) = E(XY) - E(X)E(Y) = E(X^3) - 0*E(Y) = E(X^3) = 0

Q3.

Once: E=(1+2+3+4+5+6)*1/6=3.5

Twice: Since the last one's expected value is 3.5, so for the first time, if one achieves 4,5,6, one will stop. Otherwise, she will continue throwing. So the expected value is (4+5+6)/6+1/2*3.5=4.25

Three times: Since the last one's expected value is 4.25, so for the first time, if one achieves 5,6, one will stop. Otherwise, one will continue throwing. So the expected value is (5+6)/6+2/3*4.25=14/3

Q4.

Let X_i be the positions of the edges of the N pieces, where $0 \le i \le N$, so we have $X_0 = 0$, $X_N = 1$, and the other X_i's are where the cuts are made on the rod. Therefore, the length of the pieces $V_i = X_i - X_{i-1}$, where $1 \le i \le N$.

For each V_i, we know that that all the N−1 cuts were made in the interval of $(0, X_{i-1}) \cup (X_i, 1)$. So the probability that $V_i = X_i - X_{i-1} > x$ is the probability that all the n-1 cuts happens in the rest intervals of length 1-x. Since all the cuts are random, $P(V_i > x) = (1-x)^{n-1}$. Similarly, it can be proved with Order Statistics that $P(V_i > x \cap V_j > x) = (1-2x)^{n-1}$, etc.

Let V_{max} be the longest piece of the N pieces, so the probability that V_{max} bigger than x (CDF) is:

$$P(V_{max} > x) = P(V_1 > x \cup V_2 > x \cup \ldots \cup V_n > x)$$

Using principle of inclusion/exclusion:

$$P\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i) - \sum_{i<j} P(A_i \cap A_j) + \sum_{i<j<k} P(A_i \cap A_j \cap A_k) - \cdots + (-1)^{n-1} P(\bigcap_{i=1}^{n} A_i)$$

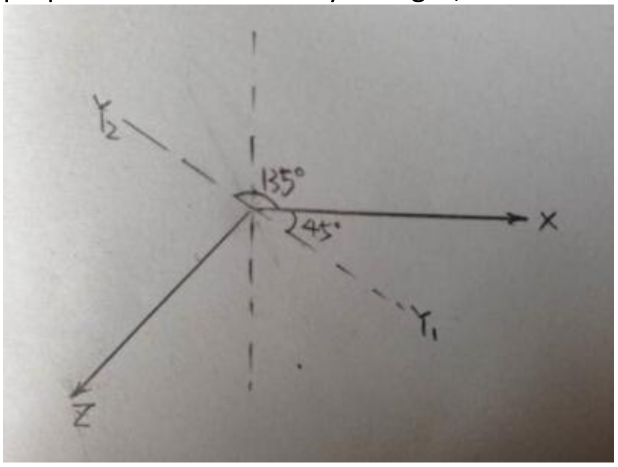
So $P(V_{max} > x) = P(V_1 > x \cup V_2 > x \cup \ldots \cup V_n > x)$

$$= \sum_{i=1}^{n} P(V_i > x) - \sum_{i<j} P(V_i > x \cap V_j > x) + \sum_{i<j<k} P(V_i > x \cap V_j > x \cap V_k > x) - \cdots$$

Until it is impossible to have k pieces bigger than x, i.e., kx > 1
So $P(V_{max} > x) = n(1-x)^{n-1} - \binom{n}{2}(1-2x)^{n-1} + \cdots + (-1)^{k-1}\binom{n}{k}(1-kx)^{n-1} + \ldots,$
where the sum is until kx > 1

And this is the Cumulative Distribution Function (CDF) of the longest piece.

Q5.
We can consider random variables x, y and z as vectors. Let θ be the angle between x and y, then we have $\cos\theta = \rho(X,Y) = \rho(Y,Z) = r$. Since $\rho(X,Z) = 0$, we can assume x and z are two perpendicular axis. With y changes, $\theta \in \left[\frac{\pi}{2}, \frac{3\pi}{2}\right], r \in \left[-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]$.



http://math.stackexchange.com/questions/284877/correlation-between-three-variables-question

Q6.
The expected steps s_x to hit either boundary starting at x satisfies the recurrence

$$s_x = \frac{1}{2}(s_{x-1} + 1) + \frac{1}{2}(s_{x+1} + 1)$$

The general solution of this first-order linear recurrence is $s_x = -x^2 + mx + n$. Given the boundary conditions which are the roots of the function $s_a = s_{-b} = 0$, we can get $s_x = (a - x)(x + b)$ so $s_0 = ab$

a. If both the right and left doors are open, i.e. a = 99 and b = -1, the expected number of steps he takes to go home is 99.

b. If the left door is locked and the right door is open, it is equivalent to the case we put left door at -101, i.e. a = 99 and b = -101, the expected number of steps he takes to go home is 99*101 = 9999.

Q7.

Ridge Regression: Search for β that minimizes

$$\sum_i (y_i - x_i^T\beta)\text{^}2 + \lambda\sum_{j=1}^{p}(\beta_j^2)$$

When λ →0, ridge estimate → OLS estimate ; λ →infinity, ridge estimate → 0

Ridge regression introduces penalty for bias and reduces the variance of the estimate.

One important reason that we choose ridge regression is that there always exists a ; λ such that MSE of ridge estimate is less than the MSE of OLS estimate (Existence Theory).

To choose a proper λ (Good fit model with parsimony): One way is to use AIC/BIC. One way is Cross Validation. Cross validation is usually better since it helps avoid over-fitting problem.

Q8.

The Least Absolute Shrinkage and Selection Operator (LASSO) solves

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\} \quad \|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{n} |\beta_j|$$

Lasso added a 1-norm penalizing term compared with regular linear regression, which ensures that coefficients will abruptly drop to zero as lambda increases – results in superior interpretability. Unlike ridge regression, a closed-from solution is not available for the Lasso. One advantage of Lasso over ridge regression is that the coefficients can abruptly drop to 0 as lambda increases, results in sparse models and can be viewed as a method for **feature selection.**

Similar to ridge regression, the value of lambda is usually selected via cross-validation.

Q9.

This question is the famous St. Petersburg paradox.

Normally, you would pay at most the expectation of the payoff of the game to play. And we can easily see that the payoff of the game, mathematically, is equal to $\mathrm{E(x)} = \sum_{k=1}^{\infty} 0.5^k 2^k = \infty$.

However, no one would pay 1 million to play this game. Why? First of all, we can argue that the utility money doesn't increase linearly and it is also related with the player's total wealth. A billionaire might be willing to pay 10,000 for the game while a student might only be wiling to pay 100. And the utility of 2^(200) dollars probably have very little difference with 2^(199) dollars to anyone. If we consider those factors, we can arrive a finite expectation of the game.

One of the other explanation is that the original game assumes that the counterparty has unlimited resources, but that is simply impossible. For example, even if the person who you are playing against is Bill Gates (who has 79 billion dollars networth in 2015), he can at most honor the rules less than 40 rounds. So if we assume that he would at most pay you his networth, the expectation of the game would be less than 40 dollars, which sounds reasonable for most people.

Q10.
In C++, a default constructor can be one of the following three forms:
1. The constructor is automatically generated by the compiler if a class has no explicitly defined constructors.

2. The constructor has no parameters.

3. The constructor has parameters that are provided default arguments.

The default constructor (eg. MyClass)is used in the following situations:
1. When an object value is declared with no argument list (e.g. MyClass x) or allocated dynamically with no argument list (e.g. new MyClass or new MyClass()).

2. When an array of objects is declared (e.g. MyClass x[10]) or allocated dynamically (e.g. new MyClass [10]).
3. When a derived class constructor does not explicitly call the base class constructor in its initializer list, the default constructor for the base class is called.
4. Some containers, like vector, "fill in" values using the default constructor when the value is not given explicitly. E.g. vector<MyClass>(10) initializes the vector with ten default constructed <MyClass> objects.
5. When a class constructor does not explicitly call the constructor of one of its object-valued fields in its initializer list, the default constructor for the field's class is called.

If we want to initialize an object with default arguments, we can define our own default constructors, whose parameters are all provided default arguments. In C++11, we can also define our own default constructors like this:
MyClass() = delete; //Inhibiting the automatic generation of a default constructor by the compiler.
MyClass() = default ; // Explicitly forcing the automatic generation of a default constructor by the compiler.
https://en.wikipedia.org/wiki/Default_constructor
http://en.cppreference.com/w/cpp/language/default_constructor

Q11.

```
// Compile with C++ 11
#include <iostream>
#include <memory>
#include <mutex>
#include <thread>
#include <vector>
#include <algorithm>

class safeSingleton
{
	static std::shared_ptr< safeSingleton > sp;
	static std::once_flag flag;

	safeSingleton(int id) {
		std::cout << "safeSingleton::Singleton() " << id << std::endl;
	}

	safeSingleton(const safeSingleton& rs) {
		sp = rs.sp;
	}

	safeSingleton& operator = (const safeSingleton& rs)
	{
		if (this != &rs) {
			sp = rs.sp;
		}
		return *this;
	}

public:
	~safeSingleton() {
		std::cout << "Singleton::~Singleton" << std::endl;
	}

	static safeSingleton & getInstance(int id)
	{
		std::call_once(safeSingleton::flag,
			[](int idx)
		{
			safeSingleton::sp.reset(new safeSingleton(idx));

			std::cout << "safeSingleton::create_singleton_() | thread id " << idx <<
std::endl;
		}
```

```
                , id);
                return *safeSingleton::sp;
        }
        void demo(int id) { std::cout << "demo stuff from thread id " << id << std::endl; }
};
std::once_flag                              safeSingleton::flag;
std::shared_ptr< safeSingleton >       safeSingleton::sp = nullptr;

int main()
{
        std::vector< std::thread > v;
        int num = 20;

        for (int n = 0; n < num; ++n) {
                v.push_back(std::thread([](int id)
                {
                        safeSingleton::getInstance(id).demo(id);
                }
                , n));
        }
        std::for_each(v.begin(), v.end(), std::mem_fn(&std::thread::join));
        return 0;
}
```

http://silviuardelean.ro/2012/06/05/few-singleton-approaches/
http://en.cppreference.com/w/cpp/memory/shared_ptr
http://en.cppreference.com/w/cpp/thread/call_once
http://www.cplusplus.com/reference/mutex/call_once/

Q12.
Firstly, if the given $k \geq n/2$, where n is the total number of days, then we can make any necessary transactions to get the maximized profit.

Otherwise, this problem can be solved by dynamic programming. Define x[i, j] to be the max profit from day 1 to day j using at most i transactions, then immediately we know all x[0, j] = x[i, 0] = 0, and x[i, j] can be computed in the following way:
x[i, j] = max(x[i, j-1], prices[j] - prices[m] + x[i-1, m])
        = max(x[i, j-1], prices[j] + max(x[i-1, m] - prices[m]))
(where $0 \leq m \leq j-1$)

Python code:

```
import numpy as np

def make_money(price, k):
    n = len(price)
    if k >=  n/2:
        profit = 0
        for i in range(1, n):
            if price[i] > price[i-1]:
                profit = profit + price[i] - price[i-1];
        return float(profit)
    else:
        x = np.zeros(shape=(k+1, n))
        for i in range(1, k+1) :
            premax = x[i-1, 0] - price[0]
            for j in range(1, n):
                x[i, j] = max(x[i, j-1], price[j] + premax)
                premax = max(premax, x[i-1, j] - price[j])
        return x[k,n-1]

>>> price = [5,7,4,3,8,9,2,4,6]
>>> make_money(price, 1)
6.0
>>> make_money(price, 2)
10.0
>>> make_money(price, 3)
12.0
>>> make_money(price, 4)
12.0
>>> make_money(price, 5)
12.0
```

Q13.

Use the XOR operator:

Python code:

```
def find_single(a):
        x = a[0]
        for  i in range(1, len(a)):
                x = x ^ a[i]
        return x

>>> a = [1, 3, 2, 4, 6, 3, 4, 2, 1]
>>> find_single(a)
6
```

Q15.
1) There are two principal methods used to generate random numbers.

The first method measures some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. This type of random number generator is often called a true random number generator.

The second method uses computational algorithms that can produce long sequences of apparently random results, which are in fact completely determined by a shorter initial value, known as a seed value or key. This type of random number generator is often called a pseudorandom number generator.

For ture random number generators, example sources include measuring atmospheric noise, thermal noise, and other external electromagnetic and quantum phenomena. For pseudorandom number generator, examples include simple but poor quality methods like midsquare method, linear congruential generator and better quality method like the Mersenne Twister, which is the default random number generator in many languages.

These two approaches each has its pros and cons. True random number generator has good randomness but poor efficiency. It is aperiodic and nondeterministic. These characteristics make true random number generators suitable for roughly the set of applications that pseudorandom number generators are unsuitable for, such as data encryption, games and gambling. Conversely, the poor efficiency and nondeterministic nature of true random number generators make them less suitable for simulation and modeling applications, which often require more data than it's feasible to generate with a true random number generator .
2) Here we refer to the pseudorandom number generators using computational algorithms.
A good random-number generator should satisfy the following properties:

• Uniformity:	The numbers generated appear to be distributed uniformly;
• Independence:	The numbers generated show no correlation with each other;
• Replication:	The numbers should be replicable;
• Cycle length:	It should take long before numbers start to repeat;
• Speed:	The generator should be fast;
• Memory usage:	The generator should not require a lot of storage.

The evaluation mainly including the uniformity(goodness of fit) and independence properties. It can be tested using Pearson's chi-squared test:

Divide the range of random numbers X into k adjacent intervals
$(a_0, a_1], (a_1, a_2], \ldots, (a_{k-1}, a_k]$
Let N_j = number of X_j in $(a_{j-1},a_j]$, and let p_j be the probability of an outcome in $(a_{j-1},a_j]$
Then the test statistic is $\chi^2 = \sum_{j=1}^{k} \frac{(N_J - np_j)^2}{np_j}$

If H_0 is true, then np_j is the expected number of the n X_i's that fall in the j-th interval, and so we expect $\chi^2$to be small.
If H_0 is true, then the distribution of χ^2 converges to a chi-square distribution with k - 1 degrees of freedom as $n \to \infty$.
The chi-square distribution with k - 1 degrees of freedom is the same as the Gamma distribution with parameters (k - 1)/2 and 2.
Hence, we reject H_0 if

$$\chi^2 > \chi^2_{1-\alpha,k-1}$$

where $\chi^2_{1-\alpha,k-1}$ is the 1 − α quantile of the chi-square distribution with k − 1 degrees of freedom.

Q16.
A straight forward way is to use rejection sampling method: flip the coin 3 times and create a 3-bit binary number in [0,7]. If the number is in [1,6], take it as a die roll. Otherwise, if the result is 0 or 7, repeat the flips.
A more efficient method is to flip the coin N times to get a number below 2^N. Then we find the largest integer M so that $6^M<2^N$ and get M dice rolls. If we get a number greater or equal to 6^M, we need to reject the value and repeat all N flips. For example, let flip the coin 8 times, the largest M is 3 so that $6^3 = 216 < 256 = 2^8$.
The result is $10111100_2 = 2^7+2^5+2^4+2^3+2^2 = 188 < 216$ and $188 = 5*6^2 + 1*6^1 + 2*6^0 = 512_6$, which gives us three dice rolls 5, 1 and 2.
The rejection rate $R = 1 - \frac{6^M}{2^N}, 6^M < 2^N$.
N=3, M=1, R=25%; N=8, M=3, R=16%; N=13, M=5, R=5%. So we can find the rejection rate decreases and efficiency increases for larger N, which make 6^M near 2^N.
http://cs.stackexchange.com/questions/29204/how-to-simulate-a-die-given-a-fair-coin

Q17.
the program is like:

```
while (! at-zero()) {
Go right;
}
while(true) {
Go right;
Go right;
}
```

It excute on both of the robots. Then at first, both robots move one step right each loop. But after the left robot reaches 0, the left robots will move two step right each loop while the right robot still move one step. The left robots will finally meet the right one.