1.

2. independent is stronger, e.g. X is standard normal, Y=X\*X, X and Y are uncorrelated but not independent

3. 14/3

4. L/N(1/1+1/2+…+1/N)

5. r square less than or equal to .5

6.

a) 99

b) 9800

Markov process with transition matrix

Use expected time to absorption formula

7. L2 penalty

8. L1 penalty

9.

10. A default constructor will only be automatically generated by the compiler if no other constructors are defined

11. Problem only happens at instantiation

std::mutex mtx;

MySingleton \* GetInstance()

{

if (m\_pOnlyOneInstance == NULL)

{

mtx.lock();

if (\_instance == NULL)

{

\_instance = new MySingleton();

}

Mtx.unlock();

}

return \_instance;

}

12. reference from leetcode

class Solution {

public:

int maxProfit(int k, vector<int> &prices) {

int n = (int)prices.size(), ret = 0, v, p = 0;

priority\_queue<int> profits;

stack<pair<int, int> > vp\_pairs;

while (p < n) {

// find next valley/peak pair

for (v = p; v < n - 1 && prices[v] >= prices[v+1]; v++);

for (p = v + 1; p < n && prices[p] >= prices[p-1]; p++);

// save profit of 1 transaction at last v/p pair, if current v is lower than last v

while (!vp\_pairs.empty() && prices[v] < prices[vp\_pairs.top().first]) {

profits.push(prices[vp\_pairs.top().second-1] - prices[vp\_pairs.top().first]);

vp\_pairs.pop();

}

// save profit difference between 1 transaction (last v and current p) and 2 transactions (last v/p + current v/p),

// if current v is higher than last v and current p is higher than last p

while (!vp\_pairs.empty() && prices[p-1] >= prices[vp\_pairs.top().second-1]) {

profits.push(prices[vp\_pairs.top().second-1] - prices[v]);

v = vp\_pairs.top().first;

vp\_pairs.pop();

}

vp\_pairs.push(pair<int, int>(v, p));

}

// save profits of the rest v/p pairs

while (!vp\_pairs.empty()) {

profits.push(prices[vp\_pairs.top().second-1] - prices[vp\_pairs.top().first]);

vp\_pairs.pop();

}

// sum up first k highest profits

for (int i = 0; i < k && !profits.empty(); i++) {

ret += profits.top();

profits.pop();

}

return ret;

}

};

13. xor

14. okay

15. srand(time(0));

random\_number = (rand() % max) + 1;

<http://www.drdobbs.com/testing-random-number-generators/184403185>

16. simulate one dice:

flip the coin three times and record the results:

HHH->1, HHT->2, HTH->3, HTT->4, THH->5, THT->6, TTH and TTT, do it again.

17. idea: first, both robots move slowly to the right (right, left, right) until left robot hit 0 (right robot never hit zero), then left robot speed up (goright() only works for left robot) and finally they will meet

while(!at\_zero()){

goright();

goleft();

goright();

}

goright();