1. ¾

2. 10

3. 2.62

4. 2\*min(p,1-p)

5.

6. Correct

7. 1-N/2^(N-1)

8. moving median, since moving average works better when normal distributed but it’s highly unlikely that market move is normal distributed.

9. (n+1)/3 (Reference from https://math152.wordpress.com/2008/10/31/stat-problem/)

10. a & (a-1) == 0

11. Smart Pointer:

#include <iostream>

#include <string>

using namespace std;

template<class T>

class ptcounter{

template<class U> friend class SmartPointer;

ptcounter(T\* pt):\_pt(pt),\_use(1){

cout << "ptcounter constructor called !" << endl;

}

~ptcounter(){

delete \_pt;

cout << "ptcounter destructor called !" << endl;

}

ptcounter(ptcounter<T> &rhs){

this->\_pt = rhs.\_pt;

this->\_use= rhs.\_use;

}

ptcounter<T>& operator=(ptcounter<T>& rhs){

this->\_pt = rhs.\_pt;

this->\_use= rhs.\_use;

}

int \_use;

T\* \_pt;

};

template<class T>

class SmartPointer{

public:

SmartPointer(T \*t):\_ptc(new ptcounter<T> (t)) {

cout<<"SmartPointer::SmartPointer() invoked use is: "<<\_ptc->\_use<<endl;

}

SmartPointer(SmartPointer<T> &rhs){

this->\_ptc = rhs.\_ptc;

this->\_ptc->\_use++;

cout<<"SmartPointer copy invoked use is: "<<\_ptc->\_use<<endl;

}

// 利用SmartPointer类自身的析构函数来对程序中未释放的指针进行释放

~SmartPointer(){

if(\_ptc){

delete \_ptc;

}

cout<<"SmartPointer::~SmartPointer() invoded"<<endl;

}

SmartPointer<T>& operator=(SmartPointer<T>& rhs){

if(rhs.\_ptc == this->\_ptc){

return \*this;

}

// 将最初指向的对象的指针引用-1

this->\_ptc->\_use--;

if(this->\_ptc->\_use == 0) {

delete \_ptc;

}

this->\_ptc = rhs.\_ptc;

this->\_ptc->\_use++;

cout<<"SmartPointer::operator=() invoked use is: "<<\_ptc->\_use<<endl;

return \*this;

}

private:

ptcounter<T> \*\_ptc;

};

class HasPtr{

public:

HasPtr(int val):value(val),p(new int(3)){

cout<<"HasPtr::HasPtr() invoked"<<endl;

}

~HasPtr(){ delete p; cout<<"HasPtr::~HasPtr() invoded"<<endl;}

private:

int \*p;

int value;

};

int main(void) {

HasPtr \*php = new HasPtr(3);

HasPtr \*php2 = new HasPtr(4);

SmartPointer<HasPtr> psp(php);

SmartPointer<HasPtr> npsp(php2);

psp = npsp;

//SmartPointer<HasPtr> npsp(psp);

SmartPointer<HasPtr> nnpsp = npsp;

nnpsp = npsp;

return 0;

}

12.

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

if(head){

ListNode\* later = head;

ListNode\* first = head;

head = head->next;

while(head){u

ListNode\* tmp = head->next;

head->next = later;

later = head;

head = tmp;

}

first->next = NULL;

return later;

}

return head;

}

};

13. if M days have to be continuous, use moving window, O(N) time and O(1) space; if M days can be disconnected, we can sort and check the smallest M elements’ average, O(NlogN) time and O(1) space.

14. KMP algorithm

15. Taylor expansion(first n terms)

double exponential(int n, double x){

double sum = 1.0;

    for (int i = n - 1; i > 0; i-- )

        sum = 1 + x \* sum / i;

    return sum;

}

16. Idea: Successive Bins/reservoir sampling/two heap

Sampling method:

http://www.tks.informatik.uni-frankfurt.de/data/events/deis10/downloads/10452.ZelkeMariano.Slides.pdf

17.

class Solution {

public:

int maxProfit(vector<int> &prices) {

if(prices.size()<=1){

return 0;

}

else{

int i = 1;

while(prices[i]<prices[i-1] && i<prices.size()){

i++;

}

if(i==prices.size()){

return 0;

}

else{

int buy = prices[i-1];

int profit = 0;

int flag = 1;//hold

for(int j = i; j<prices.size();j++){

if(prices[j]>prices[j-1] && flag==1){

continue;

}

else if(prices[j]>prices[j-1] && flag==0){

buy = prices[j-1];

flag = 1;

}

else if(prices[j]<prices[j-1] && flag==1){

flag = 0;

profit += prices[j-1]-buy;

}

else{

continue;

}

}

if(flag == 1){

profit += prices[prices.size()-1] - buy;

}

return profit;

}

}

}

};