Qishi quiz 1

Group 2

1. Problem 1

It is 3/4. In fact, Prob (x_1, x_2, x_3 will be on a semi-circle) = 3 Prob(fix x_1 as the north pole, x_2, x_3 are within the clockwise semicircle) = $3 \times \frac{1}{2} \times 12 = \frac{3}{4}$.

2. Problem 2

We have four different types of vertex:
Tier zero: the initial one;
Tier one: the ones that are one step away from the initial one;
Tier two: two steps away from the initial one
Tier three: the final one.
The expected steps from tier zero to the tier three should be one step more than the expected steps from the tier one to tier three:

$$E_0 = 1 + E_1.$$

The expected steps from tier one to tier three should be

$$E_1 = 1 + 1/3E_0 + 2/3E_2.$$

And the expected steps from tier two to tier three should be

$$E_2 = 1 + 2/3E_1.$$

Solve the set of equations above we can get

$$E_0 = 10.$$

3. Problem 3

Let $f(b, r)$ be the payoff when the starting state is there are b black cards and r red cards left on the table. Then if we stop at this stage, the payoff is $(26 - b) - (26 - r) = r - b$. If we keep on going the payoff is $\frac{b}{b+r} f(b-1, r) + \frac{r}{b+r} f(b, r-1)$. So we seek to solve the following dynamical programming problem:

$$E(f(b, r)) = \max(r - b, \frac{b}{b+r} E(f(b-1, r)) + \frac{r}{b+r} E(f(b, r-1))).$$

The boundary condition is giving by $f(0, r) = 26 - (26 - r) = r$ and $f(b, 0) = 0$ for $b, r = 0, 1, ...26$. The following program computes $f(b, r)$ recursively:

```
#include<iostream>
#include<algorithm>
using namespace std;

int main() {
    int N = 26;
```

```
    float card[N+1][N+1];
    card[0][0] = 0;
    for (int i =1; i < N+1; i++) {
       card[i][0] = 0.0;
       card[0][i] = (float) i;
    }
    for (int i = 2; i < 2*N+1; i++) {
        for (int j = 1; j < N+1; ++j) {
            float b = (float) j;
            float tmp = (float) i;
            float r = tmp - b;
            card[j][i-j] = max(r-b,(b * card[j-1][i-j])/(b+r) +
                              (r * card[j][i-j-1]) / (b+r));
        }
    }
    cout << "E(f(26,26)) is " << card[N][N] << endl;
    return 0;
}
```

The output is $E(f(26,26)) = 2.62448$. So the payoff of this game is 2.62448.

4. Problem 4

First consider the case that the first flip is head, Let $x =$ the probability of ever getting equal numberof heads and tails when the first flip is head, then

$$x = (1-p) + px^2.$$

Solving this give $x = (1-p)/p$ if $p \geq 1/2$ and $x = 1$ if $p \leq 1/2$. Similarly, if the first flip is tail, let $y =$ the probability of ever getting equal numberof heads and tails when the first flip is head. By symmetry, we get $y = p/(1-p)$ is $p \leq 1/2$ and $y = 1$ if $p \geq 1/2$.

In general the probability of ever getting equal numberof heads and tails after infinite flips is

$$p \times x + (1-p) \times y,$$

which is $p+(1-p)p/(1-p)$ if $p \leq 1/2$ and $p(1-p)/p+(1-p)$ if $p \geq 1/2$. So the probability is $2\min\{p, 1-p\}$.

5. Problem 5

They have exponential distribution whose pdf is $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$ and $f(x) = 0$ for $x < 0$. Then the mean is $\frac{1}{\lambda}$, so for the 100 hours bulbs $\lambda = 1/100$ and for the 200 hours bulbs $\lambda = 1/200$. Let the random varibles be $X_1, X_2, ...X_5, Y_1, Y_2, ..., Y_5$, where X_i Y_i represent the life age of 100 bulbs and 200 bulbs respectively. Then we are seeking to find

the distribution of $Z = \min\{X_1, X_2, ...X_5, Y_1, Y_2, ..., Y_5\}$. But

$$\begin{aligned} Prob(Z > x) &= Prob(X_1 > x, ...X_5 > x, Y_1 > x, ...Y_5 > x) \\ &= Prob(X_1 > x) \times \cdots Prob(X_5 > x) \times Prob(Y_1 > x) \times \cdots Prob(Y_5 > x) \\ &= (exp(-\frac{x}{100}))^5 \times (exp(-\frac{x}{200}))^5 \\ &= exp(-\frac{3}{40}x). \end{aligned}$$

So the pdf of Z is $f_Z(x) = \frac{3}{40}e^{-\frac{3}{40}x}$. The mean is $\frac{40}{3}$, which is just the expected number of hours before the first one burns out.

6. Problem 6

In total the person has tossed $N = 100 * 365 * 24 * 60 * 60 = 3153600000$ times. Let X be the random variable of couting the numbers of the pattern of 100 consecutive heads apprears. Let I_j be the indicator function having value 1 if starting from toss j the pattern apprears. Then by linearity of expectation, we have

$$E(X) = E(\sum_j I_j) = \sum_j E(I_j) = \frac{N - 99}{2^{100}} \approx 2.4877517 \times 10^{-21}.$$

Then by Chebyshev's inequality

$$P(X \geq 1) \leq E(X) \approx 2.4877517 \times 10^{-21} << 0.01\%.$$

So the statement is correct, i.e. the probability of tossing 100 consecutive heads is less than 0.01%.

7. Problem 7

Prob(the N pieces can form an N sided polygon) = 1- Prob(the N pieces can NOT form an N sided polygon) = 1 - Prob(some piece has length greater or equal to one half of the length of the stick). Suppose the breaking points are $a_1, ...a_{n-1}$ and the two ends of the sticks are a_0 and a_n, then by symmetry, Prob = 1 - N $\times$ Prob(the lenght of $[a_0, a_1]$ is greater or equal to one half of the length of the stick) = 1 - N $\times$ Prob($a_1, ..., a_n$ fall into the second half of the stick) = $1 - \frac{N}{2^{N-1}}$.

8. Problem 8

It depends. If it is a trending following trading system, moving average is better at detecting large move than moving median does, hence provide better signal.

Moving median a robust measure of central tendency and is less sensitive to the outliers. During 20 mins prices movement, it is high likely that there are some outlier prices, so moving median maybe be better.

Moveover, for a sideways market (neither an uptrend nor a downtrend, moving average usually does not work well. In this case, moving median may give better signals.

On the other hand, median calculation more computer-intensive than moving average.

9. Problem 9

Let I_j be the indicator function taking value 1 if there is a local maxima at position j. Then there are two different cases: (I) $j = 1$ or n. In this case, $E(I_j) = P(\text{there is a local maxima at position n}) = P(a_n > a_{n-1}) = 1/2$. (II) $j = 2, ..., n-1$. In this case, we have $E(I_j) = P(\text{there is a local maxima at position j}) = P(a_j > a_{j-1} \&\& a_j > a_{j+1}) = 1/3$. So by linearity of expectation, we have $E(X) = E(\sum_j I_j) = \sum_j E(I_j) = 1/2 * 2 + 1/3 * (n-2) = 1 + \frac{n-2}{3} = \frac{n+1}{3}$.

10. Problem 10

```
#include<stdio.h>
#define bool int

bool isPowerTwo (int n) {
    return n && (!(n & (n-1)));
}

int main() {
    isPowerTwo(16)? printf("yes\n"): printf("no\n");
    isPowerTwo(5)? printf("yes\n"): printf("no\n");
    return 0;
}
```

11. Problem 11

```
// template class for smart pointer objects
template<typename T> class smartPtr {
public:
    smartPtr(T* realPtr) : pointee(realPtr) {}

    // destructor
    ~smartPtr() {
        delete pointee;
    }

    // dereference a smart pointer to get
    // a member of what it points to
    T* operator->() const {
        return pointee;
    }
```

```
    // deference a smart pointer
    T& operator*() const {
        return *pointee;
    }

private:
     // what the smart pointer points to
     T *pointee;

};
```

12. Problem 12

```
//Definition for singly-linked list.

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
 };


ListNode *reverse(ListNode *head, int m, int n) {
    if (head == NULL || m >= n) {
        return head;
    }

    ListNode* newHead = new ListNode(0);

    newHead->next = head;
    head = newHead;

    for (int i = 1; i < m; i++) {
        if (head == NULL) {
            return NULL;
        }
        head = head->next;
    }

    ListNode* prev = head->next;
```

```
    ListNode* curr = pre->next;

    for(int i = 0 ; i < n-m ; i++){
        ListNode * tmp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = tmp;
     }
     head->next->next = curr;
     head->next = prev;
     head = newHead->next
     delete newHead;
     return head;
}
```

13. Problem 13

The code inplemented in C++ is using the idea of Max heap. The algorithm is as follows:
1. store stock prices in a max heap: this takes O(N) time 2. extract M times to get the largest M closing prices: this takes time O(M*logN) 3. check whether the average of the largest M closing prices is less or equal to P

The total runtime is O(N+M*logN)

```
#include <iostream>
#include <algorithm>
#include <vector>

class stock {
public:
        bool closing(int M, int N, std::vector<float> &price, float P);
};

bool stock::closing(int M, int N, std::vector<float> &price, float P) {
        // store stock prices in a max heap
        std::make_heap (price.begin(),price.end());
        // use res to compute the average of largest M closing prices
        float res = 0.0;
        for (int i =0; i < M; ++i) {
                // extract the max elememt and add it to res
                res = res + (float)price.front();
                std::pop_heap (price.begin(),price.end());
                price.pop_back();
```

```
        }
        res = res / (float)M;
        // compare average of the largest M closing price and P
        if (res <= P) return true;
        else return false;
}

int main () {
        stock s;
        int M = 2, N = 5;
        float P = 10.0;
        float num1[] = {1.0,2.0,3.0,5.0,15.0};
        float num2[] = {1.0,2.0,3.0,7.0,15.0};
    std::vector<float> price1(num1,num1+5);
    std::vector<float> price2(num1,num2+5);
    bool x = s.closing(M, N, price1, P);
    bool y = s.closing(M, N, price2, P);
    std::cout << "The first stock is: " << x << '\n';
    std::cout << "The second stock is: " << y << '\n';
    return 0;
}
```

On the other hand, there is another algorithm based on quickselect which takes time O(N). If M is very larget (for example M = N/2), this algorithm is faster than the above one using max heap.

The algorithm as follows: 1. Divide the array into groups of 5 2. Find median of each group in linear time. 3. Use Select() recursively to find the median x of the [n/5] medians. 4. Partition the array around x, i.e. use set x as pivot and use quicksort partition algorithm. 5. Let i = rank(x): if (M == i) then return x. 6. If ($i > M$), then recursively call Select() to find the kth largest element in the left partition of x. 7. if ($i < M$), then recursively call Select() to find (M-i)th smallest element in right partition of x. 8. After obtaining the Mth largest elment, use quicksort partition algorithm to find largest M elements.

14. Problem 14

The KMP algorithm (time complexity is $O(N + M)$, where N is the length of the text and M is the length of the pattern):

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

```
vector<int> indexOf(string s, string pattern) {
    vector<int> matchTable;
    vector<int> res;
    res.push_back(-1);

    if (pattern.size() == 0  || s.size() < pattern.size()) {
        return res;
    }

    // build the partical match table for pattern which contains:
    // the length of the longest proper prefix in the (sub)pattern that
    // matches a proper suffix in the same (sub)pattern.
    for(int i = 0; i < pattern.size(); i++) {
        string subpattern = pattern.substr(0,i+1);
        int count = 0;
        string::iterator pbegin = subpattern.begin();
        string:: iterator pend = subpattern.end();
        while (pbegin != pend) {
           if (*pbegin == *pend) {
              count++;
           }
           else {
               break;
           }
           pbegin++;
           pend++;
        }
        matchTable.push_back(count);
    }

    // use the partial match table to find pattern in string
    int i = 0;
    while(i < s.size()) {
        if ((s.size()-i) < pattern.size()) {
            return res;
        }
        int j = i;
        int partial_match_length = 0;
        string::iterator pbegin = pattern.begin();
        while (pbegin != pattern.end() && *pbegin == s[j]) {
            pbegin++;
            j++;
```

```
            partial_match_length++;
        }
        if (partial_match_length == pattern.size()) {
            res.push_back(i);
        }
        if (partial_match_length == 0) {
            i++;
        }
        else {
            i += partial_match_length - matchTable[partial_match_length-1];
        }
    }
    return res;
}

int main() {
    // no match
    string s = "baacdfaabcdab";
    string pattern = "abababca";
    vector<int> res;
    res = indexOf(s,pattern);
    for (vector<int>::iterator it = res.begin(); it != res.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;

    // there are two matches at places 0 and 3
    string s1 = "abaaba";
    string pattern1 = "aba";
    vector<int> res1;
    res1 = indexOf(s1,pattern1);
    for (vector<int>::iterator it = res1.begin(); it != res1.end(); ++it) {
        cout << *it << " ";
    }

    return 0;
}
```

15. Problem 15

Use the formula:

$$e^x = 1 + (x/1)(1 + (x/2)(1 + (x/3)(...))).$$

```
#include <stdlib.h>
#include <iostream>

using namespace std;

double expx(double x)
{
        int i=1;
        int power=1;
        while (x > 1||x<-1)
        {
                x = x / 10;
                power*=10;
        }
        double result = 0;
        double nplus1 = 1;

        while (((nplus1 > numeric_limits<double>::min()) &&
              (nplus1<numeric_limits<double>::max())) ||
                   ((nplus1 < -numeric_limits<double>::min()) &&
                    (nplus1>-numeric_limits<double>::max())))
        {
                nplus1 = nplus1*x / i;
                result += nplus1;
                i++;
        }
        return pow(result + 1, power);
}

int main()
{
        double x = -500;
        char wait;
        cout.precision(40);
        cout << expx(x) << endl;
        cin >> wait;
}
```

16. Problem 16

The idea is to bin the data and only record histogram. The number of bin can be chosen so that it guarantees to cover the whole range of data and the step of the bin can be chosen so that it meets the requirement on resolution.

```
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <string>
using namespace std;

void updateBin(vector<int>* bins, float data, float* endoBin,float step)
{
        int i=0;//aditional bins needed to be added.
        if (data > *endoBin)
        {
                i = (data - *endoBin) / step;
                *endoBin = i>0?data:*endoBin;
                while (i > 0)
                {
                        (*bins).push_back(0);
                        i--;
                }
                (*bins).back()++;
        }
        else
        {
                int mid, upper, lower;
                lower = 0;
                upper = (*bins).size();
                mid = (upper + lower) / 2;

                while (upper - lower > 1)
                {
                        lower = data >= mid*step ? mid : lower;
                        upper = data > mid*step ? upper : mid;
                        mid = (upper + lower) / 2;
                }
                (*bins)[lower]++;
        }
```

```
}
float binMedian(float data, vector<int>* bins,
                float step,float* endoBin,int n)
{
        int i=0;
        int count = 0;
        updateBin(bins, data, endoBin, step);
        if (n == 1)
        {
                cout << "The approximate median of first " << n
                     << " item is " << data << endl;
                return 0;
        }

        while (count < n / 2 )
        {
                count += (*bins)[i];
                i++;
        }

        if (n % 2 == 0)
        {
                int j = i;
                while (count < n / 2 + 1)
                {
                        count += (*bins)[j];
                        j++;
                }
                cout << "The approximate median of first " << n
                    << " item is "  << (i+j)*step*0.5-step << endl;
        }
        else
        {
                while (count < n / 2 + 1)
                {
                        count += (*bins)[i];
                        i++;
                }
                cout << "The approximate median of first " << n
                    << " item is " << (i - 1)*step << endl;
        }
        return 0;
```

```
}

int main()
{
        vector<int> bins;
        bins.push_back(0);
        float endoBin = 0;
        float* streams;
        float step = 1;
        int N = 10;
        streams = new float[N];
        int i;
        streams[0] = 0;

        for (i = 1; i < N; i++)
        {
                streams[i] = streams[i-1]+step;
        }
        i = 0;
        while (i < N)
        {
                binMedian(streams[i], &bins, step,&endoBin,i + 1);
                i++;
        }
        char wait;
        cout << "Done" << endl;
        cin >> wait;
}
```

17. Problem 17

Greeedy algorithm: if the price of next day is higher than the price of current, then add the difference to the profit.

```
#include<iostream>
using namespace std;

float maxProfit(float prices[], int size) {
    if (size <= 1) return 0.0;
    float profit = 0.0;
    for (int i = 1; i < size; i++) {
```

```
        float diff = prices[i] - prices[i-1];
        if (diff > 0.0) {
            profit += diff;
        }
    }
}

int main() {
    float prices[4] = {1.2,3.2,2.5,4.6};
    int size = 4;
    float profit = maxProfit(prices, 4);
    cout << "profit is " << profit << endl;
    return 0;
}
```