QISHI QUIZ1 **Due on 11:59PM SEP 29, 2015**
Name: Bangrui Chen

I didn't write down answers for all the problems since for some of them, my solution is the same with other people's.

Math

Problem 1

Solution: Denote the three points as point 1, point2 and point3. Consider for point1, the probability for point2 and point3 lies in the semi-circle that starts from point1 clockwise is $\frac{1}{2^2}$. Since point1, point2 and point3 are symmetric and we have three points in total, thus the final probability is $3 \times \frac{1}{2^2}$.
In general, if there are n points are the probability they all lie in a semi-circle is $n \times \frac{1}{2^{n-1}}$.

Problem 7

Without loss of generality, we assume the total length of N pieces is 1. An easy observation is that N pieces can form a polyhedron if and only if the total length of any $N-1$ pieces is greater than the length of the other one. Or equivalently, let we denote the length of the N pieces from the beginning to the end as $x_1, \cdots, x_N$, then they can form a polyhedron if and only if

$$\begin{aligned} &\sum_{j\neq i} x_j \geq x_i > 0 \; \forall i \\ \Longleftrightarrow \; & 0 < x_i < \frac{1}{2} \; \forall i \end{aligned}$$

Based on the above analysis, we know N pieces can't forma a polyhedron if and only if there is one piece $x_i \geq \frac{1}{2}$. Since

$$\begin{aligned} P(\exists i, x_i \geq \frac{1}{2}) &= \sum_{i=1}^{N} P(x_i \geq \frac{1}{2}) \\ &= NP(x_1 \geq \frac{1}{2}) \\ &= \frac{N}{2^{N-1}} \end{aligned}$$

where the last equation is due to all the $x_2, \cdots, x_n$ lies in $[\frac{1}{2}, 1]$. Thus,

$$P(0 < x_i < \frac{1}{2} \; \forall i) = 1 - P(\exists i, x_i \geq \frac{1}{2}) = 1 - \frac{N}{2^{N-1}}.$$

Thus, the probability that N pieces will form a polyhedron is $1 - \frac{N}{2^{N-1}}$.

Problem 8

Here is a very interesting experiment: `http://www.automated-trading-system.com/moving-median-better-indicator-than-moving-average/`.

The results of the experiments suggest that: the moving median will not increase robustness and the performance will also drop.

Programming

Problem 12

```cpp
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        ListNode* pointerM = head;
        if (m == 1) {
            ListNode* pointerN = head;
            for (int i = 1; i < n; i++) {
                pointerN = pointerN->next;
            }
            while (head != pointerN) {
                ListNode* temp = head;
                head = head->next;
                temp->next = pointerN->next;
                pointerN->next = temp;
            }
            return head;
        }
        for (int i = 1; i < m - 1; i++) {
            pointerM = pointerM->next;
        }
        ListNode* pointerN = pointerM;
        for (int i = 0; i <= n-m; i++) {
            pointerN = pointerN->next;
        }
        while (pointerM->next != pointerN) {
            ListNode* temp = pointerM ->next;
            pointerM->next = temp->next;
            temp->next = pointerN->next;
            pointerN->next = temp;
        }
        return head;
    }
};
```

Problem 13

- Solution 1: Use priority queue:

```
// Solution 1:
bool ave_lower_P_1(int N, int M, float P, vector<float> price){

  // imput the data into a priority queue
  priority_queue<float> price_new;
  for (int i = 0; i < N; i++) {
    price_new.push(-price[i]);
  }

  // calculate the average of the lowest M days closing price
  float sum = 0;
  for (int i = 0; i < M; i++) {
    sum += (-price_new.top());
    price_new.pop();
  }
  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}
```

- Solution 2: Use sort in c++:

```
// Solution 2:
bool ave_lower_P_2(int N, int M, float P, vector<float> price){

  sort(price.begin(), price.end());

  // calculate the average of the lowest M days closing price
  float sum = 0;
  for (int i = 0; i < M; i++) {
    sum += (price[i]);
  }
  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}
```

- Solution 3: Use min heap in c++:

```
// Solution 3:
bool ave_lower_P_3(int N, int M, float P, vector<float> price){

  make_heap(price.begin(), price.end(), std::greater<int>());

  float sum = 0;
  for (int i = 0; i < M; i++) {
```

```cpp
    sum += price.front();
    cout << price.front();
    pop_heap(price.begin(), price.end() - 1 - i, std::greater<int>());
    price.pop_back();
  }

  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}
```

Problem 17

```cpp
class Solution {
public:
    int maxProfit(vector<int> &prices) {
        int profit = 0;
        for (int i = 1; i < prices.size(); i++) {
            if (prices[i] > prices[i-1]) {
                profit += prices[i] - prices[i - 1];
            }
        }
        return profit;
    }
};
```

Code for testing problem 12

```cpp
#include <algorithm>
#include <assert.h>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;


// Solution 1:
bool ave_lower_P_1(int N, int M, float P, vector<float> price) {

  // imput the data into a priority queue
  priority_queue<float> price_new;
  for (int i = 0; i < N; i++) {
    price_new.push(-price[i]);
  }

  // calculate the average of the lowest M days closing price
  float sum = 0;
  for (int i = 0; i < M; i++) {
    sum += (-price_new.top());
    price_new.pop();
  }
  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}
// Solution 2:
bool ave_lower_P_2(int N, int M, float P, vector<float> price) {

  sort(price.begin(), price.end());

  // calculate the average of the lowest M days closing price
  float sum = 0;
  for (int i = 0; i < M; i++) {
    sum += (price[i]);
  }
  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}

// Solution 3:
bool ave_lower_P_3(int N, int M, float P, vector<float> price) {

  make_heap(price.begin(), price.end(), std::greater<int>());

  float sum = 0;
  for (int i = 0; i < M; i++) {
    sum += price.front();
    cout << price.front();
    pop_heap(price.begin(), price.end() - 1 - i, std::greater<int>());
    price.pop_back();
  }

  float ave = sum / M;

  cout << ave << endl;
  // return true or false
  return (ave <= P);
}
int main() {
  // input the data
  int N,M;
  cout << "Input the number of trading days N" << endl;
  cin >> N;
  cout << "Input M (integer)" << endl;
  cin >> M;
```

```cpp
  assert(M <= N && M >= 1);
  float P;
  cout << "Insert a price P" << endl;;
  cin >> P;
  assert( P > 0);
  cout << "Now insert the closing prices for the last " << N << " days" << endl;
  vector<float> price;
  float temp;
  for (int i = 0; i < N; i++) {
    cin >> temp;
    price.push_back(temp);
  }

  // check whether there exists M days average closing price less than P
  if (ave_lower_P_1(N,M,P,price)) {
      cout << "Yes, there exists " << M << " days" << endl;
  } else {
      cout << "No, there does not exist" << endl;
  }

  if (ave_lower_P_2(N,M,P,price)) {
      cout << "Yes, there exists " << M << " days" << endl;
  } else {
      cout << "No, there does not exist" << endl;
  }

  if (ave_lower_P_3(N,M,P,price)) {
      cout << "Yes, there exists " << M << " days" << endl;
  } else {
      cout << "No, there does not exist" << endl;
  }

  return 1;
}
```