

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, \dots, x_N , then they can form a polyhedron if and only if

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

Based on the above analysis, we know N pieces can't form 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, \dots, x_n lies in $[\frac{1}{2}, 1]$. Thus,

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

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

```
1 class Solution {
2 public:
3     ListNode* reverseBetween(ListNode* head, int m, int n) {
4         ListNode* pointerM = head;
5         if (m == 1) {
6             ListNode* pointerN = head;
7             for (int i = 1; i < n; i++) {
8                 pointerN = pointerN->next;
9             }
10            while (head != pointerN) {
11                ListNode* temp = head;
12                head = head->next;
13                temp->next = pointerN->next;
14                pointerN->next = temp;
15            }
16            return head;
17        }
18        for (int i = 1; i < m - 1; i++) {
19            pointerM = pointerM->next;
20        }
21        ListNode* pointerN = pointerM;
22        for (int i = 0; i <= n - m; i++) {
23            pointerN = pointerN->next;
24        }
25        while (pointerM->next != pointerN) {
26            ListNode* temp = pointerM->next;
27            pointerM->next = temp->next;
28            temp->next = pointerN->next;
29            pointerN->next = temp;
30        }
31        return head;
32    }
33};
```

Problem 13

- Solution 1: Use priority queue:

```

1 // Solution 1:
2 bool ave_lower_P_1(int N, int M, float P, vector<float> price)↵
3 {
4     // imput the data into a priority queue
5     priority_queue<float> price_new;
6     for (int i = 0; i < N; i++) {
7         price_new.push(-price[i]);
8     }
9
10    // calculate the average of the lowest M days closing price
11    float sum = 0;
12    for (int i = 0; i < M; i++) {
13        sum += (-price_new.top());
14        price_new.pop();
15    }
16    float ave = sum / M;
17
18    cout << ave << endl;
19    // return true or false
20    return (ave <= P);
21 }

```

- Solution 2: Use sort in c++:

```

1 // Solution 2:
2 bool ave_lower_P_2(int N, int M, float P, vector<float> price)↵
3 {
4     sort(price.begin(), price.end());
5
6     // calculate the average of the lowest M days closing price
7     float sum = 0;
8     for (int i = 0; i < M; i++) {
9         sum += (price[i]);
10    }
11    float ave = sum / M;
12
13    cout << ave << endl;
14    // return true or false
15    return (ave <= P);
16 }

```

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

```

1 // Solution 3:
2 bool ave_lower_P_3(int N, int M, float P, vector<float> price)↵
3 {
4     make_heap(price.begin(), price.end(), std::greater<int>());
5
6     float sum = 0;
7     for (int i = 0; i < M; i++) {

```

```

8     sum += price.front();
9     cout << price.front();
10    pop_heap(price.begin(), price.end() - 1 - i, std::greater<
        int>());
11    price.pop_back();
12 }
13
14 float ave = sum / M;
15
16 cout << ave << endl;
17 // return true or false
18 return (ave <= P);
19 }

```

Problem 17

```

1 class Solution {
2 public:
3     int maxProfit(vector<int> &prices) {
4         int profit = 0;
5         for (int i = 1; i < prices.size(); i++) {
6             if (prices[i] > prices[i-1]) {
7                 profit += prices[i] - prices[i - 1];
8             }
9         }
10        return profit;
11    }
12 };

```

Code for testing problem 12

```

1 #include <algorithm>
2 #include <assert.h>
3 #include <iostream>
4 #include <queue>
5 #include <vector>
6
7 using namespace std;
8
9
10 // Solution 1:
11 bool ave_lower_P_1(int N, int M, float P, vector<float> price) {
12
13     // imput the data into a priority queue
14     priority_queue<float> price_new;
15     for (int i = 0; i < N; i++) {
16         price_new.push(-price[i]);
17     }
18

```

```

19 // calculate the average of the lowest M days closing price
20 float sum = 0;
21 for (int i = 0; i < M; i++) {
22     sum += (-price_new.top());
23     price_new.pop();
24 }
25 float ave = sum / M;
26
27 cout << ave << endl;
28 // return true or false
29 return (ave <= P);
30 }
31 // Solution 2:
32 bool ave_lower_P_2(int N, int M, float P, vector<float> price) {
33
34     sort(price.begin(), price.end());
35
36     // calculate the average of the lowest M days closing price
37     float sum = 0;
38     for (int i = 0; i < M; i++) {
39         sum += (price[i]);
40     }
41     float ave = sum / M;
42
43     cout << ave << endl;
44     // return true or false
45     return (ave <= P);
46 }
47
48 // Solution 3:
49 bool ave_lower_P_3(int N, int M, float P, vector<float> price) {
50
51     make_heap(price.begin(), price.end(), std::greater<int>());
52
53     float sum = 0;
54     for (int i = 0; i < M; i++) {
55         sum += price.front();
56         cout << price.front();
57         pop_heap(price.begin(), price.end() - 1 - i, std::greater<int>());
58         price.pop_back();
59     }
60
61     float ave = sum / M;
62
63     cout << ave << endl;
64     // return true or false
65     return (ave <= P);
66 }
67 int main() {
68     // input the data
69     int N,M;
70     cout << "Input the number of trading days N" << endl;
71     cin >> N;
72     cout << "Input M (integer)" << endl;
73     cin >> M;

```

```

74  assert(M <= N && M >= 1);
75  float P;
76  cout << "Insert a price P" << endl;;
77  cin >> P;
78  assert( P > 0);
79  cout << "Now insert the closing prices for the last " << N << " ↵
      days" << endl;
80  vector<float> price;
81  float temp;
82  for (int i = 0; i < N; i++) {
83      cin >> temp;
84      price.push_back(temp);
85  }
86
87  // check whether there exists M days average closing price less ↵
      than P
88  if (ave_lower_P_1(N,M,P,price)) {
89      cout << "Yes, there exists " << M << " days" << endl;
90  } else {
91      cout << "No, there does not exist" << endl;
92  }
93
94  if (ave_lower_P_2(N,M,P,price)) {
95      cout << "Yes, there exists " << M << " days" << endl;
96  } else {
97      cout << "No, there does not exist" << endl;
98  }
99
100 if (ave_lower_P_3(N,M,P,price)) {
101     cout << "Yes, there exists " << M << " days" << endl;
102 } else {
103     cout << "No, there does not exist" << endl;
104 }
105
106 return 1;
107 }

```