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  $x_1, \dots, x_N$ , then they can form a polyhedron if and only if

$$\sum_{j \neq i} x_j \ge x_i > 0 \ \forall i$$

$$\iff 0 < x_i < \frac{1}{2} \ \forall i$$

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

$$P(\exists i, x_i \ge \frac{1}{2}) = \sum_{i=1}^{N} P(x_i \ge \frac{1}{2})$$
$$= NP(x_1 \ge \frac{1}{2})$$
$$= \frac{N}{2^{N-1}}$$

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

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

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:
       ListNode* reverseBetween(ListNode* head, int m, int n) {
3
            ListNode* pointerM = head;
             if (m == 1) {
5
                  ListNode* pointerN = head;
6
                  for (int i = 1; i < n; i++) {
7
                      pointerN = pointerN->next;
8
9
                  while (head != pointerN) {
10
                      ListNode* temp = head;
11
                      head = head->next;
12
                      temp->next = pointerN->next;
13
                      pointerN->next = temp;
14
15
                  return head;
16
17
             for (int i = 1; i < m - 1; i++) {
18
                 pointerM = pointerM->next;
19
20
            ListNode* pointerN = pointerM;
21
            for (int i = 0; i \le n-m; i++) {
22
                  pointerN = pointerN->next;
23
24
             while (pointerM->next != pointerN) {
25
                 {\tt ListNode*\ temp\ =\ pointerM\ -\!> next}\,;
26
                 {\tt pointerM->next} \; = \; {\tt temp->next} \; ;
27
28
                 \texttt{temp-}\!\!>\!\!\texttt{next} \;=\; \texttt{pointerN-}\!\!>\!\!\texttt{next}\;;
                 {\tt pointerN-\!\!\!>\!\!next\ =\ temp\,;}
29
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
    // imput the data into a priority queue
4
    priority_queue<float> price_new;
    for (int i = 0; i < N; i++) {
      price_new.push(-price[i]);
8
9
    // calculate the average of the lowest M days closing price
10
    float sum = 0;
11
    for (int i = 0; i < M; i++) {
12
       sum += (-price_new.top());
13
      price_new.pop();
14
15
    float ave = sum / M;
16
17
    \verb"cout" << \verb"ave" << \verb"endl";
18
    // 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
    sort(price.begin(), price.end());
4
5
     // calculate the average of the lowest M days closing price
6
    float sum = 0;
7
    for (int i = 0; i < M; i++) {
8
      sum += (price[i]);
9
10
11
    float ave = sum / M;
12
    cout << ave << endl;</pre>
13
    // return true or false
14
    return (ave <= P);
15
16 }
```

• 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++) {</pre>
```

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

### Problem 17

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

# Code for testing problem 12

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

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

```
assert(M \le N \&\& M >= 1);
 74
      float P;
 75
      cout << "Insert a price P" << endl;;</pre>
 76
      cin >> P;
 77
      assert(P>0);
 78
      cout << "Now insert the closing prices for the last " << N << " \leftrightarrow
 79
          days" << endl;</pre>
80
      vector<float> price;
      float temp;
81
      for (int i = 0; i < N; i++) {
 82
        cin >> temp;
 83
         price.push_back(temp);
 84
 85
 86
      // check whether there exists M days average closing price less \leftarrow
87
           than P
      if (ave_lower_P_1(N,M,P,price)) {
88
           \texttt{cout} << \texttt{"Yes}, \texttt{ there exists "} << \texttt{M} << \texttt{" days"} << \texttt{endl};
 89
 90
 91
           cout << "No, there does not exist" << endl;</pre>
92
93
      if (ave_lower_P_2(N,M,P,price))  {
94
           cout << "Yes, there exists " << M << " days" << endl;</pre>
95
      } else {
96
           cout << "No, there does not exist" << endl;</pre>
97
 98
99
      if \ (ave\_lower\_P\_3(N,M,P,price)) \ \{\\
100
           \texttt{cout} << \texttt{"Yes}, \texttt{ there exists "} << \texttt{M} << \texttt{" days"} << \texttt{endl};
101
      } else {
102
           cout << "No, there does not exist" << endl;</pre>
103
104
      return 1;
106
107 }
```