Quiz 1 Solution

Group A

1) What is the probability that three points on a circle will be on a semi-circle?

   Solution: Denote the three points as point1, point2 and point3. Event A_i is the following, two points lie in the semi-circle that starts from point$i$ clockwise. $P(A_i) = \frac{1}{2^2}$. Since point1, point2 and point3 are symmetric and A_i are mutually exclusive, 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 $n \times \frac{1}{2^{n-1}}$.

2) An ant walks randomly on the edges of a cube. It starts from a vertex, and each step it has equal probability to choose one of the three edges and walk to the other vertex of this edge. What is the expectation of the number of steps for the ant to walk from one vertex to the opposite vertex.

   Solution: Say the ant locates at $(0, 0, 0)$. Let E_i denote the expectation of steps needed to reach $(1, 1, 1)$, if the ant starts from a vertex with distance i to the $(1, 1, 1)$. Use condition expectation, we can get the following,

$$E_3 = 1 + E_2,$$
$$E_2 = \frac{1}{3}(1 + E_3) + \frac{2}{3}(1 + E_1),$$
$$E_1 = \frac{1}{3} + \frac{2}{3}(1 + E_2).$$

   Then we can get $E_3 = 10$.

3) From a deck of 52 cards, you can pick one card each time without replacement. If the card color is black, you win 1. If the card color is red, you lose 1. You can stop the game whenever you want. Questions: Will you play the game? If you want, how much would you pay to play this game?

   Solution: Let $E(r, b)$ denote the expectation of the optimal strategy if r red cards and b black cards are left. Then depend on the color of next card, we have the following.

$$E(r, b) = max(0, \frac{r}{r+b}(1 + E(r-1, b)) + \frac{b}{r+b}(-1 + E(r, b-1))).$$

   $E(26, 26)$ can be computed recursively and its value is 2.62.

4) Given a coin with probability p of landing on heads after a flip, what is the probability that the number of heads will ever equal the number of tails assuming an infinite number of flips?

   Solution: Let's only consider the case $0 < p < 1$, otherwise the answer is trival. Denote state i as the number of heads minus number of tails. Let state 0 be an absorbed state. Then the answer to the problem is equal to the probability of the absorbing probability starting from 0. Denote P_i as the absorbing probability starting from i. Then we have

$$P_1 = (1 - p) + pP_2,$$

$$P_2 = P_1^2,$$

the last equality comes from the fact that if we start from state 2, if it is absorbed by state 0, it must first go back to 1 before going back to 0, the probability of the first process is also P_1. We can get

$$P_1 = \frac{min(p, 1-p)}{p},$$

similarly we can get, $P_{-1} = \frac{min(p,1-p)}{(1-p)}$, since $P_0 = pP_1 + (1-p)P_{-1}$, then

$$P_0 = 2min(p, 1-p).$$

5) You have ten light bulbs. Five have an average life of 100 hours, and the other five have a average life of 200 hours. These light bulbs have a memoryless property in that their current age (measured in how long they have already been on) has no bearing on their future life expectancy. Assuming they are all already on what is the expected number of hours before the first one burns out?
Solution: The answer is $E(Y)$, where $Y = \min_i(X_1, X_2, \cdots, X_{10})$, X_1 to X_5 are i.i.d exponential distribution with parameter $1/100$, X_6 to X_{10} are i.i.d exponential distribution with parameter $1/200$. Then after computing the cdf of Y, we know Y is also exponential distribution with parameter $3/40$. Then $EY = 40/3$.

6) If a person tosses a coin once per second and he tosses 100 years,ask whether the following statement is correct or not: the probability of tossing 100 consecutive heads is less than 1 percent?
Solution: Yes.

$$\begin{aligned} & P(\text{there are 100 consecutive heads}) = P(\cup_i \text{there exists such a sequence starting from i}) \\ \leq & \sum_{i=1}^{N-100} P(\text{from i to i+99 the flips are all heads}) \leq N\frac{2^{N-100}}{2^N} = \frac{100 * 365 * 24 * 3600}{2^{100}} < 0.0001 \end{aligned}$$

7) Given a stick, if randomly cut into N pieces, what's the probability that the N pieces can form an N sided polygon?
Solution: The largest side being smaller than the sum of the other sides is necessary and sufficient for a given sides to form a polygon. Assume length of the stick is 1. Let x_i denote the i-th part length. The necessary and sufficient condition of making a polygon using positive x_i is, $x_i < \frac{1}{2}$ for all i.

$$P(x_i < 0.5 \text{ for all i}) = 1 - P(x_i > 0.5 \text{ for some i}) = 1 - NP(x_1 > 0.5).$$

The last equality comes from the fact that at most 1 piece can have length greater than 0.5. To have $x_1 > 0.5$ all $N-1$ cuts have to be on the right half of the stick, the probability is $1/2^{N-1}$. So the answer is $1-N/2^{N-1}$.

8) Suppose in a trading environment, to describe 20 mins prices movement, should we choose moving median or moving average? Why?
Solution: If we only consider the asymptotic relative efficiency (ARE) between mean and median, we can use order statistics and the underlying distribution to get the variance of sample median and compare it to the variance of sample mean. Generally speaking, for long tail underlying distribution, median is more efficient than sample mean (sample median's var is asymptotically smaller than sample mean). Thus median is more robust.

However, whether median is better depends on the length of the interval of interest. Short period trend(SPT) and long period trend(LPT) are different. The trade-off between these two is as follow: SPT should be more sensitive and should capture signal "outliers". SPT is vulnerable to error but it can detect the change faster; LPT should be more robust to outliers and reflect the overall trend during the time interval. 20 minutes may be regarded as a LPT, so we prefer median. Note that in some situations, 20 minutes might be considered as a SPT.

9) What is the average number of local maxima of a permutation of 1, ..., n, over all permutations? Maxima at ends also count.

   Solution: Let X_i denote the value at location i. I_i is the indicator function of location i being a maxima, then $E\sum_i I_i = \sum_i EI_i$. $EI_1 = 0.5$ and $EI_n = 0.5$ since the maxima at ends count and $I_0 = 1$ if $X_1 > X_2$ which has probability $1/2$ due to symmetry. For any middle location m, it is a maxima iff X_m is the maximum in (X_{m-1}, X_m, X_{m+1}) which has probability $1/3$ due to the symmetry. So $E\sum_i I_i = 1/2 + \frac{1}{3}(n-2) + 1/2 = (n+1)/3$.

10) Give a one-line C expression to test whether a number is a power of 2.

    Solution: $x\&(x-1) == 0$.

11) Implement a smartpointer in C++.

```
// SharedPtr.h
<template T>

class SharedPtr {
public:
  // constructor
  SharedPtr(const T &obj):
    p(new T(obj)), refCount(new std::size_t(1)) {}
  // copy constructor
  SharedPtr(const SharedPtr &p_):
    p(p_), refCount(p_.refCount) { ++*refCount }
  // copy assignment operator
  SharedPtr& operator= (const SharedPtr&);
  // Destructor
  ~SharedPtr();
private:
  T *p;
  std::size_t *refCount; // reference count
}

// Destructor definition
SharedPtr::~SharedPtr() {
  // decrement the current reference count
  // if it is 0, free the allocated resource
  // if not, do nothing
  if(--*refCount == 0) {
```

```
    delete p;
    delete refCount;
  }
}

// copy assignment operator definition
SharedPtr& SharedPtr::operator=(const SharedPtr &rhs) {
  // increment the rhs pointer's reference count
  ++*rhs.refCount;
  // decrement the lhs pointer's reference coount
  // and check whether it is 0 or not
  // if it is 0, free the allocated resource
  if( --*refCount == 0) {
    delete p;
    delete refCount;
  }
  // copy rhs pointer and reference count to lhs
  p = rhs.p;
  refCount = rhs.refCount;
  return *this;
}
```

12) Reverse a linked list from position m to n. Do it in-place and in one-pass.

```
class Solution {
public:
   ListNode *reverseBetween(ListNode *head, int m, int n) {
       if(m>=n || head==0 || head->next ==0 ) return head;
       ListNode dummy=ListNode(0);
       dummy.next=head;
       ListNode *mMinus=&dummy, *post;
       for(int i=m; --i>0;)
          mMinus=mMinus->next;
       ListNode * pre=mMinus->next, *p=pre->next;
       while(n-m++>0){
          post=p->next;
          p->next=pre;
          pre=p;
          p=post;
       }
       mMinus->next->next=p;
       mMinus->next=pre;
       return dummy.next;
   }
};
```

13) Implement a program to find out whether there exist M days within the last N(N>=M) trading days that

the average closing price of these M days is at most P. Assume we have collected the history of the closing prices of the last N trading days for a stock. Requirements: Inputs are positive integer M and N, M<= N; An array of N float elements containing the closing prices of the last N trading days; And a float P. Please design and implement the program in C, C++, Java or Python to produce the answer in most time and space efficient way.

Solution: The program returns true iff MP is greater than the sum of the lowest M prices. Among many methods min-heap method has the best performance, it has $O(MN)$ worst case time complexity and $O(1)$ space complexity. For comparison, methods of using priority queue and std::sort are also included.

```
// Solution 1: Use priority queue.
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++.
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++:
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);
}
```

14) Implement a string indexOf method that returns index of matching string.

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

class strMatch {
private:
   vector<int> next;
   void GetNext(const string& str);
public:
   strMatch(){};
   ~strMatch(){};

   bool strStr(const string& haystack, const string& needle);
   bool strStrKMP(const string& haystack, const string& needle);
};

// Brute force: time O(m*n), space O(1)
bool strMatch::strStr(const string& haystack, const string& needle) {
   if (needle.empty()) {
      return true;
   }
```

```
24
25    for (int i = 0; i < haystack.size(); i++) {
26       if (haystack[i] == needle[0]) {
27          bool match = true;
28          for (int j = 0; j < needle.size(); j++) {
29             if (haystack[i+j] != needle[j]) {
30                match = false;
31                break;
32             }
33          }
34
35          if (match) {
36             return true;
37          }
38       }
39    }
40
41    return false;
42 }
43
44 // KMP: time O(m + n), space O(n)
45 void strMatch::GetNext(const string& str) {
46       next.push_back(-1);
47       int i = -1;
48       int j = 0;
49
50       while (j < str.size() - 1) {
51             //str[i] - prefixstr[j] - suffix
52             if (i == -1 || str[j] == str[i]) {
53                i++;
54                j++;
55
56                if (str[j] != str[i]) {
57                      next.push_back(i);
58                   } else {
59                      next.push_back(next[i]);
60                   }
61
62             } else {
63                i = next[i];
64             }
65       }
66
67       return;
68 }
69
70 bool strMatch::strStrKMP(const string& haystack, const string& needle) {
71    GetNext(needle);
72
```

```
73    int i, j;
74       int haystackLen = haystack.size();
75       int needleLen = needle.size();
76
77       for (i = 0, j = 0; i < haystackLen && j < needleLen; ) {
78       // currently, match!
79             if (j == -1 || haystack[i] == needle[j]) {
80                 i++;
81                 j++;
82             } else {
83                 // currently, NOT match..
84                    j = next[j];
85          }
86       }
87
88    if (j == needle.size()) {
89       return true;
90    } else {
91       return false;
92    }
93 }
94
95 int main(int argc, char const *argv[]) {
96    strMatch soln;
97
98    string haystack;
99    cout << "Input haystack: ";
100   getline(cin, haystack);
101
102   string needle;
103   cout << "Input needle: ";
104   getline(cin, needle);
105
106   clock_t now = clock();
107   cout << "Brute force: " << soln.strStr(haystack, needle) << endl;
108   clock_t after = clock();
109   cout << "Brute force run-time: " << (after - now) /
110   (double)(CLOCKS_PER_SEC / 1000) << " ms" << endl;
111
112   now = clock();
113   cout << "KMP: " << soln.strStrKMP(haystack, needle) << endl;
114   after = clock();
115   cout << "KMP run-time: " << (after - now) /
116   (double)(CLOCKS_PER_SEC / 1000) << " ms" << endl;
117
118   return 0;
119 }
```

15) Write a function to calculate exp(x).

```cpp
#include <iostream>
#include <limits>
#include <cmath>
using namespace std;

class expFunction {
private:
   double my_power(double x, int n);
public:
   expFunction(){};
   ~expFunction(){};

   double my_exp(double x);
};

double expFunction::my_power(double x, int n) {
   if (n == 0) {
      return 1.0;
   }

   if (n%2 == 0) {
      return my_power(x, n/2) * my_power(x, n/2);
   } else {
      return x * my_power(x, n/2) * my_power(x, n/2);
   }
}

double expFunction::my_exp(double x)
{
   if (x < 0) {
      return 1.0/my_exp(-x);
   }

   // Round up x when x is large so that
   // e^x = 1 + x + ... + x^n/n! + O(x^n) converges faster.
   int roundup = ceil(x);
   double x_modified = x/roundup;

   double result = 1.0;
   double TaylorExpansionTerm = x_modified;
   int n = 1;
   while (TaylorExpansionTerm > numeric_limits<double>::min()) {
      result += TaylorExpansionTerm;
      TaylorExpansionTerm *= (x_modified/++n);
   }

```

```
47    return my_power(result, roundup);
48 }
49
50
51 int main(int argc, char const *argv[]) {
52    expFunction soln;
53
54    double power;
55    cout << "Input the power: " << endl;
56    cin >> power;
57
58    cout << "e^" << power << " = " << soln.my_exp(power) << endl;
59
60    return 0;
61 }
```

16) Given streaming data, design an algorithm to get approximate median of all previous data, use constant memory.

Solution: Ideas: 1.Successive Bins/2.Reservoir sampling/3.Max and Min heap

Successive Bins: http://www.stat.cmu.edu/ ryantibs/papers/median.pdf (paper by Robert Tibshirani's son...)

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

Max and Min heap: Hold two heaps, one max heap for values less than current median, one min heap for values large than current median. The size of the two heaps diff at most 1 by constantly change the current median.

Suppose we already have a heap structure in C++, max and min heap method is as follows.

```
1  int get_median(int new_number, Heap& min_heap, Heap& max_heap) {
2    if (max_heap.size() && new_number < max_heap.top()) {
3      max_heap.enqueue(new_number);
4      if (max_heap.size() > min_heap.size()) {
5        min_heap.enqueue(max_heap.dequeue());
6      }
7    } else {
8      min_heap.enqueue(new_number);
9      if (min_heap.size() > max_heap.size()+1) {
10       max_heap.enqueue(min_heap.dequeue());
11     }
12   }
13
14   if (max_heap.size()==min_heap.size()) {
15     return (max_heap.top() + min_heap.top())/2;
16   } else {
17     return  min_heap.top();
18   }
19 }
```

17) Say you have an array for which the i-th element is the price of a given stock on day i. Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e. buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (i.e. you must sell the stock before you buy again).

```
int maxProfit(vector<int> &prices) {
   int profit = 0;
      for (int i = 1; i < prices.size(); i++)
         profit += max(prices[i] - prices[i - 1],0);
   return profit;
}
```