Q1:
3/4
Two points will always in a semicircle.
Nearest: If two points lie on the same, then the p=1
Farthest: If two points have 180 angle, then p=0.5
Along all the points, the probability are linearly decreasing, so the expected value shoud be (1+1/2)/2=3/4.

Q2:
10
Set the vertex to be A, and the opposite one to be B, set three vertex that directly connet to A to be A1, the rest to be B1.
Use E(A) as notation for starting from A, the expected steps taken to arrive at B.

E(A)=1+E(A1)
E(A1)=1/3*(1+E(A))+2/3(1+E(B1))
E(B1)=1/3+2/3*(E(A1)+1)

==> E(A1)=9 E(A)=10 E(B1)=7

Q3.
Assuming there are both 26 black and red cards, obviously we would play this game, because the worst outcome would be 0 given we can always choose to stop at the end.

To calculate the expected payoff of this game, we can treat it like American Option pricing and price the game backwards:

We first start off at only one card left. If the card left is Black, we will have keep playing for sure and the expected payoff is 0 since all the cards is revealed. If the card left is Red, we will stop and keep $1 as the payoff.

Now let's get to the position where there are two cards left. Using same logic we will keep playing if both left are black and get payoff 0, or stop and keep $2 as the payoff when both left are red cards. Now, if there is one each. then we need to consider two scenarios: 1. draw black, get $1 and stop. 2. draw red, get $-1 and keep playing until the end, which end up with $0 additional payoff. So the expected payoff is 0.5*1+0.5*0 = 0.5, you will keep play.

Using this recursive method we can calculate all the scenarios all the way back to the start. We finally get the expected payoff of the game: 2.62

Please see the excel table attached for the calculation details:

	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
26	2.624476	2.321697	2.060653	1.834201	1.636719	1.46364	1.311139	1.176023	1.055659	0.947876	0.850881	0.763169	0.683469	0.610699	0.543935	0.482393	0.425401	0.372372	0.32279	0.276192	0.232158	0.190306	0.150265	0.111658	0.074074	0.037037	0
25	2.927254	2.5727	2.269686	2.008897	1.78317	1.586815	1.415074	1.263982	1.130278	1.011297	0.904857	0.809149	0.722654	0.644081	0.572339	0.506505	0.445796	0.389535	0.337128	0.288047	0.241817	0.198006	0.156204	0.115995	0.076923	0.038462	0
24	3.295991	2.875714	2.520044	2.216565	1.955963	1.731078	1.535947	1.365597	1.215944	1.083676	0.96611	0.861047	0.766654	0.681386	0.603939	0.533217	0.468301	0.408402	0.352834	0.300991	0.252331	0.206366	0.162637	0.120684	0.08	0.04	0
23	3.75129	3.246204	2.823522	2.466309	2.162108	1.901816	1.677906	1.484072	1.315144	1.166952	1.036152	0.920042	0.816394	0.723337	0.639299	0.562971	0.493259	0.42924	0.370115	0.315184	0.263823	0.215476	0.16963	0.125769	0.083333	0.041667	0
22	4.322258	3.705641	3.196266	2.770511	2.411082	2.106256	1.846457	1.623621	1.431121	1.26363	1.116924	0.987639	0.873037	0.770837	0.679123	0.596311	0.521093	0.452375	0.389222	0.330815	0.276435	0.225443	0.177258	0.131304	0.086957	0.043478	0
21	5.050987	4.284476	3.660726	3.146278	2.715908	2.354246	2.049036	1.789871	1.568159	1.376994	1.21095	1.065776	0.938074	0.825031	0.724293	0.63392	0.552329	0.478211	0.410461	0.348118	0.290342	0.236394	0.185613	0.137352	0.090909	0.045455	0
20	6	5.027037	4.248666	3.617636	3.094791	2.659457	2.295859	1.990468	1.732015	1.511412	1.321558	1.156991	1.013436	0.887393	0.775934	0.676659	0.587623	0.507247	0.43421	0.367377	0.305755	0.248485	0.194805	0.143986	0.095238	0.047619	0
19	7	6	5.005902	4.218908	3.573657	3.041236	2.601249	2.235999	1.930557	1.672787	1.453211	1.264658	1.101666	0.959842	0.8355	0.725628	0.627811	0.540113	0.460943	0.388945	0.322936	0.261905	0.204969	0.151299	0.1	0.05	0
18	8	7	6	5	4.190145	3.527537	2.985723	2.54144	2.174761	1.86925	1.611993	1.393336	1.206168	1.044917	0.904892	0.782259	0.673968	0.577614	0.491259	0.413264	0.342209	0.276888	0.216268	0.159398	0.105263	0.052632	0
17	9	8	7	6	5	4.159653	3.479372	2.92849	2.480272	2.112214	1.806356	1.549309	1.331585	1.146047	0.986648	0.848437	0.727498	0.620792	0.525923	0.440896	0.363983	0.293727	0.228906	0.168421	0.111111	0.055556	0
16	10	9	8	7	6	5	4.127467	3.429441	2.869981	2.418072	2.048282	1.741419	1.484381	1.26794	1.084208	0.926692	0.790267	0.671017	0.565935	0.472567	0.388779	0.312792	0.243137	0.178535	0.117647	0.058824	0
15	11	10	9	8	7	6	5	4.093799	3.378378	2.810974	2.355145	1.982391	1.673767	1.417223	1.202345	1.020484	0.864798	0.730125	0.61262	0.509223	0.417275	0.334559	0.259288	0.189951	0.125	0.0625	0
14	12	11	10	9	8	7	6	5	4.059263	3.327579	2.75275	2.291015	1.913208	1.603451	1.347834	1.134653	0.95458	0.800629	0.667764	0.552133	0.450361	0.359649	0.277778	0.202941	0.133333	0.066667	0
13	13	12	11	10	9	8	7	6	5	4.025586	3.280446	2.695809	2.222965	1.840809	1.530561	1.276139	1.064544	0.886042	0.73384	0.603019	0.489238	0.388889	0.29916	0.217857	0.142857	0.071429	0
12	14	13	12	11	10	9	8	7	6	5	4	3.241398	2.634516	2.151058	1.765412	1.455181	1.201854	0.991413	0.814345	0.664285	0.535553	0.4234	0.324176	0.235165	0.153846	0.076923	0
11	15	14	13	12	11	10	9	8	7	6	5	4	3.198551	2.568841	2.075643	1.687397	1.377221	1.124214	0.914384	0.739379	0.591629	0.464744	0.353846	0.255495	0.166667	0.083333	0
10	16	15	14	13	12	11	10	9	8	7	6	5	4	3.151712	2.499183	1.997573	1.607228	1.295893	1.041661	0.833402	0.660839	0.515152	0.38961	0.27972	0.181818	0.090909	0
9	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3.101116	2.426953	1.918563	1.524702	1.208268	0.954196	0.748252	0.577922	0.433566	0.309091	0.2	0.1	0
8	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3.048318	2.356186	1.841135	1.43411	1.114375	0.861805	0.65812	0.488889	0.345455	0.222222	0.111111	0
7	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2.299038	1.753846	1.335373	1.014569	0.763889	0.560606	0.391667	0.25	0.125	0
6	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2.232101	1.656177	1.229437	0.909091	0.657143	0.452381	0.285714	0.142857	0
5	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2.15404	1.549784	1.119048	0.793651	0.535714	0.333333	0.166667	0
4	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2.066667	1.444444	1	0.657143	0.4	0.2	0
3	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1.342857	0.85	0.5	0.25	0
2	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1.2	0.666667	0.333333	0
1	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0.5	0
0	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Q4. 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?

Answer:
This is a standard random walk problem. Consider the starting point to be 0, getting a head to be moving one step to the right, and getting a tail to be moving one step to the left. Then the probably to the right is p, and the probability of moving to the left is 1-p, and the probability of the number of heads ever equals the number of tails is essentially the probability that moving back to the origin after the first step

(a) The direction of positive and negative is arbitrary here, so the probability of p and 1-p is symmetric. Let ‘s consider the case when p >= 0.5, and the situation when p < 0.5 would be similar.

Let’s call:

A = probability of ever moving one step to the right of where you are
B = probability of moving back to the current position from its left

For A, to ever move one step to the right, one can move one step to its right directly, or move to the left and then moving back, or even back and forth like this many times, so
$A = p + Bp + B^2p + B^3p + \ldots\ldots$
When we flip the coin for infinite times,
$A = p / (1-B)$

Note that A is a probability distribution, so $p/(1-B) <= 1$, so we know $B <= 1-p$ here

Also for B, the probability of moving back to the current position from the left is equal to the probability of moving one step to the left, times the probability of moving back to the current position from either direction is

$B = (1-p) * A$

Solve for B, and we have:

$B/(1-p) = p/(1-B)$, $B = p$ or $1-p$

When p > 0.5, B <= 1-p < 0.5, so B = p
When p =0.5, B = 0.5, also B = p

So if we start from origin, the probability of moving back to origin from its left is p. Because of the symmetry, we know that the probability of moving back to the origin from its right is also B = p. So the probability of moving back to the origin is 2p.

(b) When p < 0.5, repeat the above process by replacing p with 1-p. So the probability of moving back to the origin is 2(1-p).

Combine the results of the two cases, and the number of heads will ever equal the total number of tails
p = 2min(p, (1-p))

Q5.
The probability density function of the shorter lived life bulbs is f1(x)=1/100 * e-x/100.
The probability density function of the longer lived life bulbs is f2(x)=1/200 * e-x/200.
The survival function is 1 minus the integral of the density function (CDF). S1(x)=e-x/100, and S2(x)=e-x/200.
We can get the survival function for the entire group: (e-x/100)5 * (e-x/200)5.
The density function for the group is the derivative of 1 minus the survival function (CDF) e-3x/40.
The expectation for the group to have first failure is the integral from 0 to infinity of x times the density function for the group = 40/3 hours.

Q6:
Correct.
100 year= 60*60*24*365*100 =>10^10 (seconds)
consecutive heads=(1/2)^100=(2^10)^10=1024^10 =>10^-30

P(tossing 100 consecutive heads)=P((100 consecutive heads start from 1st second) n (100 consecutive heads start from 2nd second) n ....)<=P (100 consecutive heads) * total time=>10^-30*10^10=10^-20<0.01%

So the statement is correct.

Q7.
Cutting a stick into N pieces with N-1 cut points is equivalent to cutting a same-length circle into N pieces with N cut points, since the first cut is arbitrary. To form a polygon, we need the length of any piece should be less than sum of other pieces. Assume the length of the stick and the circumference of the circle is 1, if one piece is more than 0.5, then they cannot form a polygon.
Define event Ei as from first cut at point i the other points are in the clockwise semicircle.

P(Ei) = $\left(\frac{1}{2}\right)^{N-1}$, for i = 1...N.

P(cannot form a polygon) = $\frac{N}{2^{N-1}}$

P(can form a polygon) = $1-\frac{N}{2^{N-1}}$
Ref: http://www.zhihu.com/question/25408010 and Math.pdf
Q8:
Moving average.
Reason:
1. Outliers, moving average catches outliers. Outliers are signal that worth tracing.
2. Sensitiveness. Moving average is more sensative to instant trading change, comparing to moving medium.

Q9.
We can calculate the expectation of sum of local maximas by calculating the sum of expectation of local maximas of each position.

It is not hard to see that the expectation of being maxima for the first position and last position is 0.5, since there is only one number next to it and the chance of either one been bigger is equal. We can also get that the expectation of being maxima for all the other positions is 1/3, since
there are two numbers next to it instead of one and the chance of each one been the biggest is equal.

So the answer is 0.5*2+⅓*(n-2)=1+(n-2)/3

Q10.
A number of power of 2 has form of 10...0 and the number minus 1 has form of 01...0. If we and them bit-wise, it will return 0. We also want to exclude 0. C++ Code:

```
bool isPower2(int x)
{
        return x && !(x & (x - 1));
}
```

Ref: http://stackoverflow.com/questions/3638431/determine-if-an-int-is-a-power-of-2-or-not-in-a-single-line

Q11.

```
#include <iostream>

using namespace std;

class referencecount
{
  public:
  int count;                                    //reference count

  referencecount() : count(0) {}
  ~referencecount() {}
  void increase() { count++;}
  int decrease() { return --count;}
};
```

```
template < class T >
class smartpointer
{
   private:
   T* p;                                //pointer
   referencecount* reference;   //reference count

   public:
   //overloading constructors
   smartpointer()
   {
          reference = new referencecount();
          reference->increase();                    //reference count = 1 when construct
          cout << "Pointer is being created: " << "reference count = " << reference->count<< endl;
   }

   smartpointer(T* p): p(p)
   {
          reference = new referencecount();
          reference->increase();
          cout << "Pointer is being created: " << "reference count = " << reference->count<< endl;
   }

   smartpointer(const smartpointer<T>& copyfrom): p(copyfrom.p), reference(copyfrom.reference)
   {
          reference->increase();
          cout << "Pointer is being created: " << "reference count = " << reference->count<< endl;
   }

   //destructor
   ~smartpointer()                                  //decrease the reference count, if reference become
zero delete the pointer
   {
      if(reference->decrease() == 0)
      {
             cout << "Pointer is being deleted " << "reference count = " << reference->count<<  endl;
         delete p;
         delete reference;
      }
          else cout << "Pointer is being deleted " << "reference count = " << reference->count<<  endl;
   }

   //overloading operator =
   smartpointer<T>& operator = (const smartpointer<T>& copyfrom)
   {
      if (this != &copyfrom)                        //avoid copy from itself
      {
         //copy the T pointer  and reference pointer and increase the reference count
```

```
        p = copyfrom.p;
        reference = copyfrom.reference;
        reference->increase();
            cout << "copy pointers: reference count = " << reference->count<< endl;
      }

        return *this;
   }
};

class test{

        public:
        int a;
        test(int a) : a(a) {}
        ~test() {}
};

int main()
{
 cout<< "Testing the smart pointer" << endl;
 for(int i=0; i < 1; i++){
        smartpointer<test> first(new test(0));

        for(int j=0; j < 1; j++){
                smartpointer<test> second;
                second =  first;
        }
 }
 return 0;

}
```

Q12.

```
//Reverse a linked list from position m to n.
#include<iostream>
#include<new>
#include<climits>

using namespace std;

struct ListNode {                                       //definition for singly-linked list.
public:
 int val;
 ListNode *next;
 ListNode(int x) : val(x), next(NULL) {}                //constructor
};
```

```
void Append(ListNode* head, int x){                                    //add one node after the last node
        ListNode *last = new ListNode(x);
        head->next = last;
}

ListNode* ReverseBetween(ListNode* head, int m, int n) {        //this algorithm is in-place and one-pass
   n-=m;
   ListNode prehead(0);
   prehead.next=head;
   ListNode* mpre=&prehead;
   while(--m)mpre=mpre->next;
   ListNode* pstart=mpre->next;                        //pstart point to the mth node, mpre point to the
node before it
   while(n--)
   {
      ListNode *p=pstart->next;                              //p point to the node after pstart
      pstart->next=p->next;                                  //delete this node
      p->next=mpre->next;
      mpre->next=p;                                          //move it after mpre
   }
   return prehead.next;

 }

int main()
{
  double x;
  int i;
  cout << "Enter a list of numbers to construct a singly-linked list \n"
     << "(Quit with a letter):" << endl;

  ListNode prehead(0);
  ListNode* head = &prehead;
  for( i = 0; cin >> x; ++i, head = head->next)
        Append(head,x);

  cin.clear();
  cin.ignore(INT_MAX,'\n');

  head = prehead.next;
  cout << "The given singly-linked list:" << endl;
  while( head) {cout << head->val << "  "; head = head->next;}
  cout << endl;

  int m, n;
  cout << "Reverse between nodes:"<<endl;
  cin >> m >> n;
```

```
  head = ReverseBetween(prehead.next, m, n);
  cout << "The reversed numbers" << endl;
  while( head) {cout << head->val << "  "; head = head->next;}
  cout << endl;
        return 0;
}
```

Q13.
Python Code:

```
import numpy as np
import heapq
def mnp(m, n, prices, p):
        return np.mean(heapq.nlargest(m, prices[-n:])) <= p
```

Some test case:

```
>>> np.random.seed(123)
>>> prices = 5 * np.random.randn(100) + 100
>>> mnp(5, 50, prices, 100)
False
>>> mnp(5, 50, prices, 105)
False
>>> mnp(5, 50, prices, 110)
True
>>> mnp(25, 50, prices, 105)
True
```

Better version:
Sometimes it is better to test if the input satisfy requirements first, i.e. if n < len(prices), m <= n, and if p and prices are numeric, m and n are integer, and then provide proper error message when the input is not valid. Here is an example:

```
import numpy as np
import heapq
def mnp(m, n, prices, p):
        try:
                m, n, p = int(m), int(n), float(p)
                prices = [float(x) for x in prices]
                if m > n or n > len(prices):
                        raise NotImplementedError('m should be no more than n, n should be no more
than the length of prices')
                else:
                        return np.mean(heapq.nlargest(m, prices[-n:])) <= p
        except ValueError:
                print ('Error: m, n should be integers, prices and p should be numeric')
```

Some test case:

```
>>> np.random.seed(123)
```

```
>>> prices = 5 * np.random.randn(100) + 100
>>> mnp(5, 50, prices, 100)
False

>>> mnp(150, 50, prices, 100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in mnp
NotImplementedError: m should be no more than n, n should be no more than the length of prices

>>> mnp(5, 150, prices, 100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in mnp
NotImplementedError: m should be no more than n, n should be no more than the length of prices

>>> mnp('5.3', 50, prices, 100)
Error: m, n should be integers, prices and p should be numeric
```

Q14.
We use the straightforward way to compare the substring with the whole string. If the first character of substring appears, it starts to count until finished the comparison. If the comparison is interrupted, we will move forward to later comparison. At last, it returns the index list of all matched positions. C++ Code:

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

vector<int> indexOf(string& str, string& substr)
{
	vector<int> ivec; // use vector to contain match index
	int len = str.size();
	int sublen = substr.size();
	int limit = len - sublen + 1;

	for (int i = 0; i < limit; i++)
	{
		int count = 0;
	   for (int j = 0; j < sublen; j++)
	   {
	      if (str[j+i] != substr[j])
	      {
		   i += j; // increase i by j if not match
		  break;
	      }
	      count++;
	   }
```

```
            if (count == sublen)
            { ivec.push_back(i);}
        }
        return ivec;
}

int main()
{
        string s = "qishibuysidequantshixinhang";
        string p = "shi";
        vector<int> ivec = indexOf(s,p); // match at 2 and 17
        for (int i = 0; i < ivec.size(); i++)
        {
                cout << ivec[i] << ' ';
        }
        return 0;
}
```

Q15.

We use Taylor expansions to expand the exponential function. Since the expansion terms will be smaller and smaller, we stop the approximation with the term is smaller than the value we defined. C++ Code:

```
#include <iostream>
using namespace std;
// use Taylor expansion to approximate the e^x
double expo(double x)
{
        double ret = 0;
        double t = 1;
// stop approximation if the item is small enough
        for (int i = 1;t > 1.0E-20;i++)
        {
                ret += t;
                t = t * x / i;
        }
        return ret;
}
int main()
{
        double j = expo(20);
        cout << j << endl;
        return 0;
}
```

Q16.

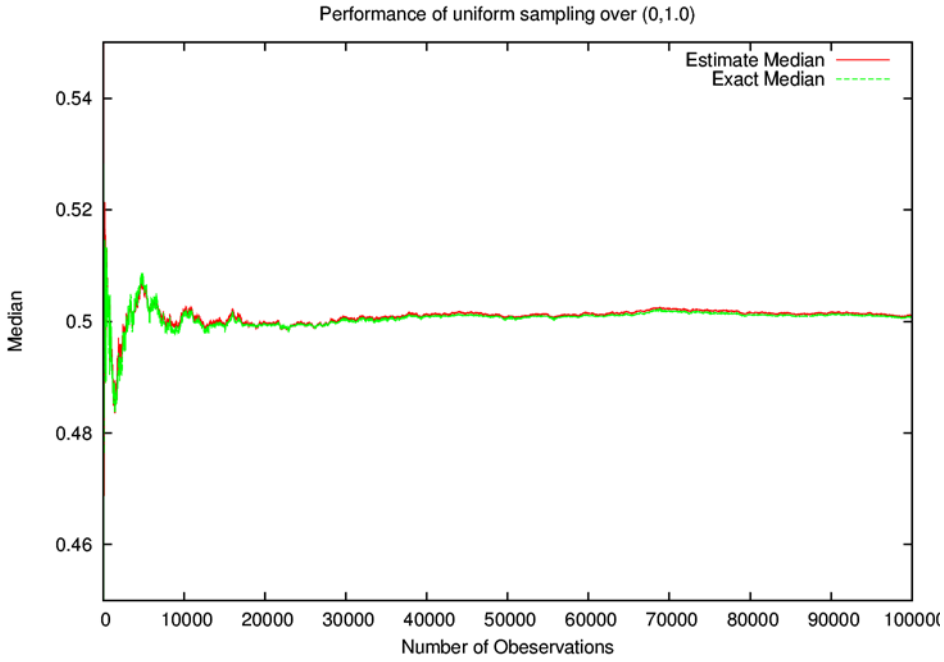
```
//Estimate the median of streaming data, need  gcc 4.6.3 or up
#include <iostream>
#include <fstream>
#include <iomanip>
```

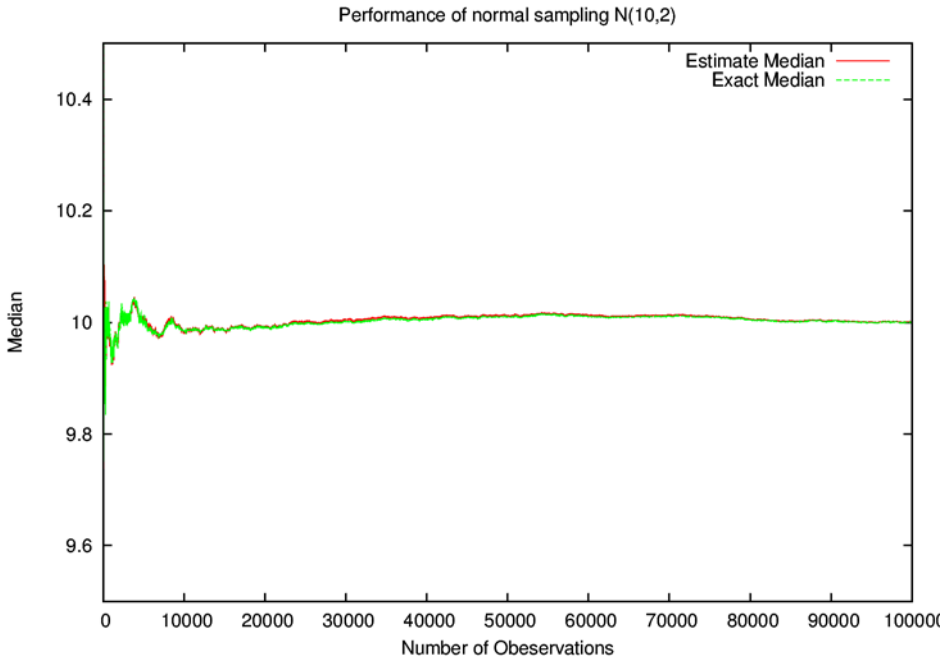
```
#include <algorithm>
#include <vector>
#include <cmath>
#include <random>

int N = 100000;              // total number of datas
double AVERAGE = 10;    // parameter for sampling
double STDERROR =2;
/*
---------------------------------------------------------------------------------------------------
The algorithm is from paper:

Raj Jain, Imrich Chlamtac, The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations,
Communications of the ACM, v.28 n.10, p.1076-1085, Oct. 1985  [doi>10.1145/4372.4378].

Here we set p = 0.5, so we get 50th percentile(median) at the center marker position.

---------------------------------------------------------------------------------------------------
*/

using namespace std;

class MarkerPoint{

  public:

    int n;              //current marker postion(x), count the number of datas less than or equal to q
    double q;                   //current estimate value at the Marker position(y)
    double n0;                  //desired marker location

    MarkerPoint(int n, double q, double n0): n(n), q(q), n0(n0) {}    // constructor
};

class Median_Est{

  public:
    vector<MarkerPoint> marker;             //5 marker points
    Median_Est (vector<double>);            //constructor
    void update (double);             //update the location of markers
    void adjust ();             //update the location of markers
};

Median_Est::Median_Est(vector<double> initial_five){

          for(int i =0; i < 5; i++)
                    marker.push_back(MarkerPoint(i+1, initial_five[i], i+1));
}
```

```
void Median_Est::update(double x){

	int k = 0;

	if(x<marker[0].q) {marker[0].q = x; k = 1;} //Find cell k such that qk<= X< qk+i and adjust
min/max (q0/q4) if necessary
	else{
		if (marker[0].q<=x && x<marker[1].q) k = 1;
		else{
			if (marker[1].q<=x && x<marker[2].q) k = 2;
			else{
				if (marker[2].q<=x && x<marker[3].q) k = 3;
				else{
					if (marker[3].q<=x && x<=marker[4].q) k = 4;
					else{
						marker[4].q = x; k = 4;
					}
				}
			}
		}
	}

	for( int i = 0; i < 5; i++)
	{
		marker[i].n0 += ((double)i/4.0); //update the desired marker locations if the total
number of datas have a increament 1
		if(i >= k) marker[i].n++;
	}

}

void Median_Est::adjust(){

	for(int i = 1; i <= 3; i++){
		double d = marker[i].n0 -marker[i].n;
		if( (d>=1 && (marker[i].n+1)<marker[i+1].n)||(d<=-1 && (marker[i].n-1)>marker[i-
1].n)){ // constraint 1: n(i+1) > n(i)
			d = d/abs(d);
			//p2 method: quadratic interpolation using adjacent marker points
			double dq = (d/(marker[i+1].n - marker[i-1].n))*((marker[i].n-marker[i-
1].n+d)*(marker[i+1].q-marker[i].q)/(marker[i+1].n-marker[i].n)+(marker[i+1].n-marker[i].n-
d)*(marker[i].q-marker[i-1].q)/(marker[i].n-marker[i-1].n));
			if( marker[i].q + dq < marker[i+1].q && marker[i].q + dq > marker[i-1].q)
marker[i].q += dq; //constraint 2: q(i+1) > q(i)
			//otherwise, use linear interpolation
			else marker[i].q += (d * (marker[i+d].q - marker[i].q)/(marker[i+d].n -
marker[i].n));
```

```
                marker[i].n += d;
            }
        }

}


int main()
{
  default_random_engine generator;
  uniform_real_distribution<double> distribution(0, 1.0);
  //normal_distribution<double> distribution(AVERAGE, STDERROR);

  ofstream write("./performance", ios::out);
  write << "count" << setw(15) << "est median" << setw(15) << "exact median" << setw(15) <<
"theoretical median" << endl;

  vector<double> all_datas;

  // first five obeservations
  for(unsigned int i = 0; i < 5; i++) all_datas.push_back(distribution(generator));
  sort(all_datas.begin(), all_datas.end());
  Median_Est median(all_datas);
  write << "5" << setw(15) << median.marker[2].q << setw(15) << all_datas[2] << setw(15) << 0.5 << endl;

  // start from 6th obeservation
  for(unsigned int i = 5; i < N; i++){

        double tmp = distribution(generator);
        all_datas.push_back(tmp);

        // sort datas to get the exact median for comparison
        sort(all_datas.begin(), all_datas.end());
        double exact_median;
        if(i%2 == 0) exact_median = all_datas[(int)(i/2)];
        else exact_median = (all_datas[(int)(i/2)] + all_datas[(int)(i/2)+1])/2;

        // main routine for estimate median
        median.update(tmp);
        median.adjust();

        // write result to file
        write << all_datas.size() << setw(15) << median.marker[2].q << setw(15) << exact_median <<
setw(15) << AVERAGE << endl;
  }

  write.close();
```

```
  return 0;
}
```


Performance of uniform sampling over (0,1.0)
Estimate Median
Exact Median
Median
0.54
0.52
0.5
0.48
0.46
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Number of Obeservations


Performance of normal sampling N(10,2)
Estimate Median
Exact Median
Median
10.4
10.2
10
9.8
9.6
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Number of Obeservations

Q17.
If we can both long and short stocks, the long short profit (lsp) will be the sum of absolute difference of any two consecutive numbers. If we can only long stocks, the long profit (lp) will be the sum of positive difference of any two consecutive numbers. We also attach the code to find the buy and sell day numbers when only long is permitted.

Python Code to find max profit:

```
plist = [10,11,12,13,14,12,12,11,12,14,13,11]
lp = 0 # max profit for only long
lsp = 0 # max profit for both long and short
for i in range(len(plist)-1):
   if plist[i] < plist[i+1]:
      lp += plist[i+1] - plist[i];
      lsp += plist[i+1] - plist[i];
   else:
      lsp += plist[i] - plist[i+1]
print lp,lsp
```

Python Code to find buy and sell point:

```
plist = [10,11,12,13,14,12,12,11,12,14,13,11]
blist = []
slist = []
pos = False
if plist[0] < plist[1]:
   pos = True
   blist.append(1)
for i in range(1,len(plist)-1):
   if plist[i-1] <= plist[i] and plist[i] > plist[i+1] and pos:
      slist.append(i+1)
      pos = False
   if plist[i-1] >= plist[i] and plist[i] < plist[i+1] and not pos:
      blist.append(i+1)
      pos = True
if pos:
   slist.append(len(plist)+1)

print 'Buy at ',blist,'\nSell at ',slist
```