```cpp
// g++ p.cpp -o p && ./p < 1.in
#include <iostream>
#include <string>
#include <algorithm>
#include <functional>
#include <cmath>
#include <array>
#include <vector>
#include <list>
#include <deque>
#include <queue>
#include <stack>
#include <map>
#include <set>
#include <unordered_map>
#include <bitset>
#include <climits>
#include <cfloat>

using namespace std;

#define SN scanf("\n")
#define SI(x) scanf("%d",&x)
#define SII(x,y) scanf("%d %d",&x,&y)
#define SL(x) scanf("%lld",&x)
#define SD(x) scanf("%lf",&x)
#define SC(x) scanf("%c",&x)
#define FOR(i, s, k) for(int i=s; i<k; i++)
#define REP(i, n) FOR(i, 0, n)
#define INF INT_MAX
#define EPS 1e-9
#define PI acos(-1)

typedef long long int lint;
typedef unsigned long long int ulint;
typedef pair<int, int> ii;
typedef pair<double, int> di;
typedef vector<int> vi;
typedef vector<lint> vl;
typedef vector<double> vd;
typedef vector<bool> vb;
typedef list<int> li;
typedef vector<vector<int>> vvi;
typedef vector<vector<double>> vvd;
typedef vector<pair<int, int>> vii;
typedef list<pair<int, int>> lii;
typedef vector<list<int>> vli;
typedef vector<list<pair<int, int>>> vlii;
typedef vector<list<pair<double,int>>> vldi;

/* Union Find Integer */
map<int, pair<int, unsigned int>> Sets;
void AddSet(int x){ Sets.insert(make_pair(x, make_pair(x, 1))); }
int Find(int x){
  if(Sets[x].first == x){ return x; }
  else{ return Sets[x].first = Find(Sets[x].first); }
}
void Union(int x, int y) {
  int parentX = Find(x), parentY = Find(y);
  int rankX = Sets[parentX].second, rankY = Sets[parentY].second;
  if (parentX == parentY){ return; }
  else if(rankX < rankY){
    Sets[parentX].first = parentY;
    Sets[parentY].second += Sets[parentX].second;
  }else{
    Sets[parentY].first = parentX;
    Sets[parentX].second += Sets[parentY].second;
  }
}
```

```
int Size(int x){ return Sets[Find(x)].second; }
void Reset(){ Sets.clear(); }

/* DFS */
void DFS(vlii &adj, vb &visited, int x){
  cout<<x;
  visited[x] = true;
  for(ii e : adj[x]){
    int y = e.second;
    int w = e.first;
    if(!visited[y]){
      DFS(adj, visited, y);
    }
  }
}

/* Topological sort */
vi topologicalSort(vlii adj){
  int n = adj.size();
  vi o(n,-1); vi pred(n,0);
  REP(i,n){
    for(ii e : adj[i]){
      int j = e.second;
      pred[j]++;
    }
  }
  stack<int> S; int i = 1;
  REP(u,n){
    if(pred[u]==0){
      if(o[u]==-1){ S.push(u); }
      while(!S.empty()){
        int v = S.top(); S.pop();
        o[v] = i; i++;
        for(ii e : adj[v]){
          int y = e.second;
          pred[y]--;
          if(pred[y]==0){ S.push(y); }
        }
      }
    }
  }
  return o;
}

/* Kruskal's Algorithm (Minimum spanning tree) */
typedef struct Edge{
  ii e;
  int w;
} Edge;
bool compareWeight(Edge a, Edge b){ return a.w < b.w; }
vector<Edge> Kruskal(vvi adj){
  int n = adj.size();
  vector<Edge> mst;
  vector<Edge> edges;
  REP(i,n){
    REP(j,n){
      if(adj[i][j]!=0){
        edges.push_back({make_pair(j,i),adj[i][j]});
      }
    }
  }
  sort(edges.begin(), edges.end(), compareWeight);
  REP(i,n){ AddSet(i); }
  for(Edge e : edges){
    if(Find(e.e.first)!=Find(e.e.second)){
      mst.push_back(e);
      Union(e.e.first,e.e.second);
    }
  }
```

```
  Reset();
  return mst;
}

/* Prim's Algorithm (Minimum spanning tree) */
int Prim(vlii adj){
  int n = adj.size();
  vb visited(n, false);
  priority_queue<ii, vii, greater<ii>> Q;
  int cost = 0;
  Q.push(make_pair(0,0));

  while(!Q.empty()){
    ii p = Q.top(); Q.pop();
    int v = p.second;
    int w = p.first;

    if(!visited[v]){
      cost += w;
      visited[v]=true;
      for(ii nei : adj[v]){
        if(!visited[nei.second]){
          Q.push(nei);
        }
      }
    }
  }
  return cost;
}

/* Dijkstra's Algorithm (Shortest paths from source) */
vi Dijkstra(vlii adj, int src){
  int n = adj.size();
  priority_queue<ii, vii, greater<ii>> PQ;
  vi dist(n,INF);
  vi parent(n, -1);
  dist[src] = 0;
  PQ.push(make_pair(0,src));
  while(!PQ.empty()){
    int u = PQ.top().second; PQ.pop();
    for(ii p : adj[u]){
      int v = p.second; int w = p.first;
      if(dist[u]+w<dist[v]){
        dist[v] = dist[u]+w;
        parent[v] = u;
        PQ.push(make_pair(dist[v],v));
      }
    }
  }
  return dist;
}

/* Bellman-Ford's Algorithm (Shortest paths from source and negative weight) */
pair<bool,vd> BellmanFordCycle(vldi adj, int src){
  int n = adj.size();
  deque<int> Q, Qp;
  vd dist(n,DBL_MAX);
  vi parent(n, -1);
  dist[src] = 0;
  Q.push_back(src);
  REP(i,n){
    while(!Q.empty()){
      int v;
      v = Q.front(); Q.pop_front();

      for(di p : adj[v]){
        int w = p.second; double c = p.first;
        if(dist[v]+c<dist[w]){
          dist[w] = dist[v]+c;
```

```
          parent[w] = v;
          if(find(Qp.begin(),Qp.end(),w)==Qp.end()){
            Qp.push_back(w);
          }
        }
      }
    }
    swap(Q,Qp);
  }
  return make_pair(!Q.empty(),dist);
}


/* Ford-Fulkerson's Algorithm (Maximum Flow) */
bool FordFulkersonBFS(vvi residualAdj, int s, int t, vi &parent){
  int n = residualAdj.size();
  vb visited(n, false);
  queue<int> q;
  q.push(s);
  visited[s] = true;
  parent[s] = -1;

  while(!q.empty()){
    int u = q.front(); q.pop();
    REP(v,n){
      if(!visited[v] && residualAdj[u][v]>0) {
        q.push(v);
        parent[v] = u;
        visited[v] = true;
      }
    }
  }
  return visited[t];
}
int FordFulkerson(vvi adj, int s, int t) {
  int n = adj.size();
  vvi residualAdj(n, vi(n));
  REP(i,n){ REP(j,n){ residualAdj[i][j] = adj[i][j]; } }
  vi parent(n);
  int maxFlow = 0;

  while(FordFulkersonBFS(residualAdj, s, t, parent)){
    int pathFlow = INF;
    int u;
    int v = t;
    while(v != s){
      u = parent[v];
      pathFlow = min(pathFlow, residualAdj[u][v]);
      v = u;
    }

    v = t;
    while(v != s){
      u = parent[v];
      residualAdj[u][v] -= pathFlow;
      residualAdj[v][u] += pathFlow;
      v = u;
    }
    maxFlow += pathFlow;
  }
  return maxFlow;
}

/* Longest common subsequence non recursive */
// "abcd","wabwd" => 3
int lcs(string a, string b){
  int m=a.size();
  int n=b.size();
  vvi memo(m+1,vi(n+1));
```

```

  REP(i,m+1){
    REP(j,n+1){
      if(i==0 || j==0){
        memo[i][j]=0;
      }else if(a[i-1] == b[j-1]){
        memo[i][j]=memo[i-1][j-1]+1;
      }else{
        memo[i][j]=max(memo[i-1][j],memo[i][j-1]);
      }
    }
  }
  return memo[m][n];
}

/* Longest incresing subsequence O(n^2) */
int lis1(vi nums) {
  int n = nums.size();
  vi d(n,1);
  REP(i,n){
    REP(j,i){
      if(nums[j]<nums[i]) d[i]=max(d[i],d[j]+1);
    }
  }

  int ans = d[0];
  FOR(i,1,n) ans = max(ans, d[i]);
  return ans;
}

/* Longest incresing subsequence O(n.log(n)) */
int lis2(vi nums) {
  int n = nums.size();
  vi d(n+1,INF); d[0]=-INF;
  REP(i,n){
    int j = upper_bound(d.begin(),d.end(),nums[i])-d.begin();
    if (d[j-1]<nums[i] && nums[i]<d[j]) d[j] = nums[i];
  }

  int ans = 0;
  REP(i,n+1){
    if (d[i]<INF) ans = i;
  }
  return ans;
}

/* Compute positive modulo */
lint modulo(lint a,lint b){ return (a%b+b)%b; }

/* Compute GCD */
lint gcd(lint a, lint b){
  if(b==0){
    return a;
  }else{
    return gcd(b, a%b);
  }
}

/* Compute LCM */
lint lcm(lint a, lint b){
  return a*b/gcd(a,b);
}

/* Find Bezeout relation */
pair<lint,pair<lint,lint>> bezout(lint a, lint b){
  lint s = 0; lint sp = 1;
  lint t = 1; lint tp = 0;
  lint r = b; lint rp = a;
  lint q, temp;
```

```
  while(r!=0){
    q = rp/r;
    temp=rp; rp=r; r=temp-q*rp;
    temp=sp; sp=s; s=temp-q*sp;
    temp=tp; tp=t; t=temp-q*tp;
  }
  return make_pair(rp,make_pair(sp,tp));
}

/* Compute inverse modulo m of a O(n.log(n)) */
lint modularInverse(lint a, lint m){
  lint m0 = m, t, q;
  lint x0 = 0, x1 = 1;
  if(m==1){ return 0; }

  //extended Euclid algorithm
  while(a>1){
    q = a/m;
    t = m;
    m = a%m;
    a = t;
    t = x0;
    x0 = x1-q*x0;
    x1 = t;
  }

  if(x1<0){ x1 += m0; } //make x1 positive
  return x1;
}

/* Chinese Remainder Theorem O(n.log(n)) */
//returns the smallest x s.t.
//x = a[i] (mod r[i]) for all i between 0 and n-1
//assumption: a[i]s are pairwise coprime
lint chineseRemainder(vl a, vl r){
  int n = a.size();
  ulint prod=1; REP(i,n){ prod*=a[i]; }

  lint result = 0;
  REP(i,n){
    lint pp = prod/a[i];
    result += r[i]*modularInverse(pp, a[i])*pp;
  }
  return result%prod;
}

/* Generate all permutations O(n!.n) */
void permutations(vi currentConfig){
  int m = currentConfig.size();

  //things to do for current perm
  REP(i,m){cout<<currentConfig[i];}cout<<endl;

  vi nextConfig = currentConfig;
  //Find largest k s.t. a[k]<a[k+1]
  int k=m-1-1; while(k>=0 && nextConfig[k]>=nextConfig[k+1]){ k--; }
  //If k does not exist then this is the last permutation
  if(k<0){ return; }
  //Find largest l s.t. a[k]<a[l]
  int l=m-1; while(l>=0 && currentConfig[k]>=currentConfig[l]){ l--; }
  //Swap value a[k] and a[l]
  swap(nextConfig[k],nextConfig[l]);
  //Reverse sequence from a[k+1] to a[m]
  reverse(nextConfig.begin()+k+1, nextConfig.end());

  permutations(nextConfig);
}

/* Geometric structs */
```

```
typedef struct Point{
  double x;
  double y;
} Point;

/* Return the angle defined by ABC (degree) */
double angle(Point a, Point b, Point c){
  double aa = pow(b.x-a.x,2) + pow(b.y-a.y,2);
  double bb = pow(b.x-c.x,2) + pow(b.y-c.y,2);
  double cc = pow(c.x-a.x,2) + pow(c.y-a.y,2);
  return acos((aa+bb-cc)/sqrt(4*aa*bb))*180/PI;
}

/* Return the centroid of a polygon */
Point centroid(vector<Point> points){
  int n = points.size();
  Point centroid = {0, 0};
  double area = 0;
  double x0=0; double y0=0;
  double x1=0; double y1=0;
  double partialArea = 0.0;

  REP(i,n){
    x0 = points[i].x;
    y0 = points[i].y;
    x1 = points[(i+1)%n].x;
    y1 = points[(i+1)%n].y;
    partialArea = x0*y1 - x1*y0;
    area += partialArea;
    centroid.x += (x0+x1)*partialArea;
    centroid.y += (y0+y1)*partialArea;
  }

  area *= 0.5;
  centroid.x /= (6.0*area);
  centroid.y /= (6.0*area);

  return centroid;
}

/* Geometry algorithms from geeksforgeeks.org */
//given collinear points p, q, r check if q lies on pr
bool onSegment(Point p, Point q, Point r){
  if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
      q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
    return true;
  return false;
}

//find orientation of ordered triplet (p, q, r)
//p,q,r collinear => 0
//clockwise => 1
//counterclockwise =>2
int orientation(Point p, Point q, Point r){
  int val = (q.y-p.y)*(r.x-q.x)-(q.x-p.x)*(r.y-q.y);
  if (val == 0) return 0;
  return (val > 0)? 1: 2;
}

//check if line segment p1q1 and p2q2 intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2) {
  int o1 = orientation(p1, q1, p2);
  int o2 = orientation(p1, q1, q2);
  int o3 = orientation(p2, q2, p1);
  int o4 = orientation(p2, q2, q1);

  if (o1 != o2 && o3 != o4) return true;

  if (o1 == 0 && onSegment(p1, p2, q1)) return true;
```

```
  if (o2 == 0 && onSegment(p1, q2, q1)) return true;
  if (o3 == 0 && onSegment(p2, p1, q2)) return true;

  if (o4 == 0 && onSegment(p2, q1, q2)) return true;
  return false;
}

//check if p lies inside the polygon
bool isInside(vector<Point> polygon, Point p){
  int n = polygon.size();
  if (n < 3) return false;

  Point extreme = {10000, p.y+1001};
  int count = 0, i = 0;
  do{
    int next = (i+1)%n;
    if (doIntersect(polygon[i], polygon[next], p, extreme)){
      if (orientation(polygon[i], p, polygon[next]) == 0)
      return onSegment(polygon[i], p, polygon[next]);
      count++;
    }
    i = next;
  } while (i != 0);

  return count&1; //same as (count%2 == 1)
}

/* Grahamâ\200\231s scan (convex hull) O(n.log(n)) from geeksforgeeks.org */
Point p0;
//utility function to find next to top in a stack
Point nextToTop(stack<Point> &S){
  Point p = S.top();
  S.pop();
  Point res = S.top();
  S.push(p);
  return res;
}
//utility function to swap two points
void swap(Point &p1, Point &p2){
  Point temp = p1;
  p1 = p2;
  p2 = temp;
}
//utility function for distance between p1 and p2
int distSq(Point p1, Point p2) {
  return (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
}
//utility function to sort an array of points
int compare(const void *vp1, const void *vp2){
  Point *p1 = (Point *)vp1;
  Point *p2 = (Point *)vp2;

  int o = orientation(p0, *p1, *p2);
  if(o==0) return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

  return (o == 2)? -1: 1;
}
//returns convex hull of a set of n points.
vector<Point> convexHull(vector<Point> points){
  int n = points.size();
  int ymin = points[0].y, min = 0;
  for(int i = 1; i < n; i++){
    int y = points[i].y;
    if((y < ymin) || (ymin == y && points[i].x < points[min].x)){
      ymin = points[i].y, min = i;
    }
  }
  swap(points[0], points[min]);

```

```
  p0 = points[0];
  qsort(&points[1], n-1, sizeof(Point), compare);

  int m = 1;
  for (int i=1; i<n; i++){
    while(i<n-1 && orientation(p0,points[i],points[i+1])==0) i++;
    points[m] = points[i];
    m++;
  }

  stack<Point> S;
  S.push(points[0]);
  S.push(points[1]);
  S.push(points[2]);

  for (int i = 3; i < m; i++){
    while(S.size()>1 && orientation(nextToTop(S),S.top(),points[i])!=2) S.pop();
    S.push(points[i]);
  }

  vector<Point> hull;
  while (!S.empty()){
    Point p = S.top();
    hull.push_back(p);
    S.pop();
  }
  return hull;
}

int main(){

  int t; SI(t); FOR(testcase, 1, t+1){

    double a; string b; SD(a); SN; cin>>b;

    vii c = {{1,3},{5,0},{1,2},{2,3}};
    sort(c.begin(), c.end(), [](const ii a, const ii b){
      if(a.second==b.second) return a.first>=b.first;
      else return a.second>=b.second;
    }); //(2;3)(1;3)(1;2)(5;0)

    printf("%.4f\n", 2.436729092);

    printf("Case #%d: \n", testcase);
  }

  return 0;
}
```