The Chef has come to a forest. He sees **N** trees in a line. He wants to tie a rope from the top of one tree to the top of another. To successfully tie this rope, he needs to cut the trees that get in the way of the rope. However, he does not like this kind of deforestation and wants your help to choose the best pair of trees between which to tie the rope.

Formally, you are given **N** trees on the X-axis. Tree $\mathbf{T_i}$ has height $\mathbf{H_i}$ and is planted at $\mathbf{x = x_i}$.

You need to choose two trees $\mathbf{T_i}$ and $\mathbf{T_j}$ (where $\mathbf{x_i < x_j}$ and $\mathbf{H_i < H_j}$) and connect the **tops** of these trees with a rope such that the following conditions are satisfied:

- The angle between the rope and the X-axis is equal to **45** degrees.
- The rope **does not pass** through any other trees.

In order to satisfy the second condition, you are allowed to reduce the height of any trees except $\mathbf{T_i}$ and $\mathbf{T_j}$. Formally, you should choose numbers $\mathbf{C_1}$, $\mathbf{C_2}$, ..., $\mathbf{C_N}$, such that $\mathbf{0 \le C_k \le H_k}$ for each $\mathbf{1 \le k \le N}$ and $\mathbf{C_i = C_j = 0}$. Then decrease $\mathbf{H_k}$ by $\mathbf{C_k}$ for each $\mathbf{1 \le k \le N}$.

Find the maximum value of **(length of the rope between the two tops) - (sum of $\mathbf{C_k}$ for each tree $\mathbf{T_k}$)**.

Note that the rope can touch the top of some intermediate tree. Look at the examples given for such a scenario.

Input

- The first line of the input contains a single integer **T** denoting the number of test cases. The description of **T** test cases follows.
- The first line of each test case contains a single integer **N** denoting the number of trees.
- Each of the following **N** lines contains two space-separated integers denoting $\mathbf{x_i}$ and $\mathbf{H_i}$.

Output

For each test case, print a single line with the maximum value of the given expression. Your answer will be considered correct if the **absolute or relative error $\le 10^{-6}$**. If it's impossible to find any valid pair of trees to tie the rope between, print **-1** instead.

Constraints

- $1 \le \mathbf{T} \le 100$
- $1 \le \mathbf{x_i}, \mathbf{H_i} \le 10^9$
- $1 \le \mathbf{N} \le 2.5 \cdot 10^5$
- all $\mathbf{x_i}$ will be distinct
- sum of **N** over all test cases $\le \mathbf{5 \cdot 10^5}$

Example

Input:

```
2
3
1 1
2 2
3 3
3
1 1
2 5
3 3
```

Output:

```
2.82842712
-0.17157287
```

Explanation

Example case 1: Tie the rope from tree 1 (1,1) to tree 3 (3,3). You don't need to cut down any trees. Note that this rope would touch the top of tree 2, but that is fine. Answer = length of rope = 2 · sqrt(2) = 2.82842.

Example case 2: Tie the rope from tree 1 (1,1) to tree 3 (3,3). You need to cut down tree 2 from height 5 to height 2. Answer = length of rope - (height reduction of tree 2) = 2 · sqrt(2) - 3 = -0.17157287.

PROBLEM

You are given N trees on the X-axis. Tree T_i has height H_i and is planted at $x = x_i$. You need to choose two trees T_i and T_j (where $x_i < x_j$ and $H_i < H_j$) and connect the tops of these trees with a rope such that the following conditions are satisfied:

- The angle between the rope and the X-axis is equal to 45 degrees.
- The rope does not pass through any other trees.

In order to satisfy the second condition, you are allowed to reduce the height of any trees except T_i and T_j. Formally, you should choose numbers $C_1, C_2, \ldots, C_N$, such that $0 \le C_k \le H_k$ for each $1 \le k \le N$ and $C_i = C_j = 0$, then decrease H_k by C_k for each $1 \le k \le N$.

Find the maximum value of $(\text{Length of the rope between the two tops} - \sum_{i=1}^{N} C_i)$
Note that the rope can touch the top of some intermediate tree.

Constraints: $N \le 2.5 \cdot 10^5$. All x_i are distinct.

PREREQUISITES

Knowledge of Fenwick Tree or Segment Tree.

EXPLANATION

Partitioning into groups

First, we analyze the constraint given to connect two trees. Note that the two trees we choose cannot have any height reductions. Suppose the pair (i, j) is a valid candidate for the answer. Then we must have $H_j - H_i = X_j - X_i$. Rearranging this equation, we get that $H_i - X_i = H_j - X_j$. This gives us a partitioning of all the trees into groups based on their $H_i - X_i$ value. We denote this value as G_i. We can connect two trees if and only if they belong to the same group.

Solving each group

Consider two trees i and j belonging to the same group i.e. $G_i = G_j$. Let's analyze the cost of rope between T_i and T_j. We denote this as $\text{cost}(i, j)$. If we need to cut the tree at (X_k, H_k) then $G_k > G_i$ and the cost is precisely $G_k - G_i$. Thus, the total cost is the summation of $G_k - G_i$ where $G_k > G_i$ and $X_i < X_k < X_j$. Note that if we process the groups in decreasing order of G_i value, then the condition $G_k > G_i$ will implicitly be taken care of, and we would have to consider only the $X_i < X_k < X_j$ constraint, which can be done by using Fenwick Trees indexed by X_i values.

So now we can compute the cost of each candidate pair in $O(\log n)$, but we're still not done. Why? Because we can potentially have $O(n^2)$ candidates since a group can have $O(n)$ elements. We're almost there!

Reduction to Maximum Subarray Sum

We now present another observation to optimize the previous step. Sort the trees in each group G in increasing order of their X_i values. Now consider three trees (T_i, T_j, T_k) in this sorted group (i.e. $X_i < X_j < X_k$). Then, $\text{cost}(i, k) = \text{cost}(i, j) + \text{cost}(j, k)$. Similarly, let's denote $\text{len}(p, q)$ as the length of rope between a valid candidate pair (p, q) and $\text{profit}(p, q)$ as $\text{len}(p, q) - \text{cost}(p, q)$. Then, $\text{len}(i, k) = \text{len}(i, j) + \text{len}(j, k)$ and $\text{profit}(i, k) = \text{profit}(i, j) + \text{profit}(j, k)$.

This implies that we need to consider only adjacent trees in a group, and then the profit of *any pair* can be represented as a sum of a subarray of the profit list we computed. Thus, we only need to consider $O(n)$ adjacent pairs to build this profit list, and then find the maximum sum subarray in this list. This is a very well known problem and can be solved efficiently in $O(n \log n)$.

That's it! Note that the complexity is $O(n \log n)$ since we use a Fenwick Tree or Segment Tree to efficiently compute the cost and profit of candidate pairs. Note that the fact that we process the groups in decreasing order of G_i values is important too; We would require some more complicated data structures if we didn't do that.

```cpp
#include<bits/stdc++.h>
using namespace std;

#define ll long long
const int N = 250005;
struct Data{ int x,h,hx; }a[N];
struct Node{ ll sum; int cnt; }tree[4*N];
set<int>ss;
map<int,int>mp;
map<int,int>sp;
vector<int>vec;
vector<int>G[N];
vector<double>profit;

bool cmp(Data xx, Data yy){
    return xx.x < yy.x;
}

void CLEAR_ALL(int n){
    memset(tree,0,sizeof(tree));
    ss.clear();
    mp.clear();
    sp.clear();
    vec.clear();
    for(int i=0; i<=n; i++)G[i].clear();
}

double get_dist(double x,double h){
    return sqrt((x*x)+(h*h));
}

void update(int node,int b,int e,int x,int v){
    if(b==x && e==x){
        tree[node].sum += v;
        tree[node].cnt+=1;
        return;
    }
    int lson=(node<<1), rson=lson+1, m=(b+e)>>1;
    if(x<=m)update(lson,b,m,x,v);
    else update(rson,m+1,e,x,v);
    tree[node].sum = tree[lson].sum + tree[rson].sum;
    tree[node].cnt = tree[lson].cnt + tree[rson].cnt;
}

Node query(int node,int b,int e,int x,int y){
    if(b==x && e==y)return tree[node];
    int lson=(node<<1), rson=lson+1, m=(b+e)>>1;
    if(y<=m)return query(lson,b,m,x,y);
    else if(x>m)return query(rson,m+1,e,x,y);
    else{
        Node f1 = query(lson,b,m,x,m);
        Node f2 = query(rson,m+1,e,m+1,y);
        Node f;
        f.sum = f1.sum + f2.sum;
        f.cnt = f1.cnt + f2.cnt;
        return f;
    }
}

int main(){
    int tt; scanf("%d",&tt);
    while(tt--){
        int n; scanf("%d",&n);
        CLEAR_ALL(n);
        for(int i=1; i<=n; i++){
            scanf("%d%d",&a[i].x,&a[i].h);
        }
        sort(a+1,a+n+1,cmp);
        for(int i=1; i<=n; i++){
            ss.insert(a[i].h - a[i].x);
            a[i].hx = a[i].h - a[i].x;
        }
```

```cpp
        int k = 0;
        for(auto v : ss){
            mp[v] = ++k;
            sp[k] = v;
            vec.push_back(v);
        }
        for(int i=1; i<=n; i++){
            G[mp[a[i].hx]].push_back(i);
        }

        double ans = -1e+40;
        int flag = 0;
        for(int id=k; id>0; id--){
            int sz = (int)G[id].size();
            if(sz==0)continue;
            if(sz==1){
                int ii = G[id][0];
                update(1,1,n,ii,a[ii].hx);
                continue;
            }

            flag = 1;
            profit.clear();
            for(int i=0; i<sz-1; i++){
                int ii = G[id][i];
                int jj = G[id][i+1];
                Node val = query(1,1,n,ii,jj);
                double dist = get_dist(a[jj].x-a[ii].x+0.0, a[jj].h-a[ii].h+0.0);
                double prof = dist - (val.sum - (val.cnt*sp[id]));
                profit.push_back(prof);
            }

            /// Maximum Subarray Sum
            double have = 0.0;
            for(int i=0; i<profit.size(); i++){
                have += profit[i];
                ans = max(ans,have);
                have = max(have,0.0);
            }

            for(int i=0; i<sz; i++){
                int ii = G[id][i];
                update(1,1,n,ii,a[ii].hx);
            }
        }

        if(flag==0){
            printf("-1\n");
        }else{
            printf("%.9f\n",ans);
        }
    }
    return 0;
}
```