**Assignment 01:**

Ramu owns a Pan shop and he manages it in his own way. While in a normal shop, a customer is served by following the first-come, first-served rule, Ramu simply minimizes the average waiting time of his customers. So he gets to decide who is served first, regardless of how sooner or later a person comes.

Different kinds of Pans take different amounts of time to cook. Also, once he starts cooking a Pan, he cannot cook another Pan until the first Pan is completely cooked. Let's say we have three customers who come at time t=0, t=1, & t=2 respectively, and the time needed to cook their Pans is 3, 9, & 6 respectively. If Ramu applies first-come, first-served rule, then the waiting time of three customers is 3, 11, & 16 respectively. The average waiting time in this case is (3 + 11 + 16) / 3 = 10. This is not an optimized solution. After serving the first customer at time t=3, Ramu can choose to serve the third customer. In that case, the waiting time will be 3, 7, & 17 respectively. Hence the average waiting time is (3 + 7 + 17) / 3 = 9.

Help Ramu achieve the minimum average waiting time. For the sake of simplicity, just find the integer part of the minimum average waiting time.

**Input Format**

* The first line contains an integer N, which is the number of customers.
* In the next N lines, the ith line contains two space separated numbers Ti and Li. Ti is the time when ith customer order a pan and Li is the time required to cook that pan.
* The  ith customer is not the customer arriving at the  ith arrival time.

**Output Format**

* Display the integer part of the minimum average waiting time.

**Note**

* The waiting time is calculated as the difference between the time a customer orders pan (the time at which they enter the shop) and the time she is served.
* Cook does not know about the future orders.

**Code:**

import java.io.\*;

import java.util.\*;

import java.math.\*;

public class Solution {

static BufferedReader in;

static PrintWriter out;

static StringTokenizer tok;

static void solve() throws Exception {

int n = nextInt();

Customer[] a = new Customer[n];

for (int i = 0; i < n; ++i) {

a[i] = new Customer(nextLong(), nextLong());

}

Arrays.sort(a);

long res = 0;

long time = 0;

PriorityQueue<Long> q = new PriorityQueue<Long>();

int i = 0;

while (!q.isEmpty() || i < n) {

if (q.isEmpty() && i < n) {

time = Math.max(a[i].t, time);

}

while (i < n && a[i].t <= time) {

res += time - a[i].t;

q.add(a[i].l);

++i;

}

long minV = q.poll();

res += (q.size() + 1) \* minV;

time += minV;

}

out.println(res / n);

}

static class Customer implements Comparable<Customer> {

long t, l;

Customer(long t, long l) {

this.t = t;

this.l = l;

}

public int compareTo(Customer other) {

return Long.compare(this.t, other.t);

}

}

static int sqr(int x) {

return x\*x;

}

static int nextInt() throws IOException {

return Integer.parseInt(next());

}

static long nextLong() throws IOException {

return Long.parseLong(next());

}

static double nextDouble() throws IOException {

return Double.parseDouble(next());

}

static BigInteger nextBigInteger() throws IOException {

return new BigInteger(next());

}

static String next() throws IOException {

while (tok == null || !tok.hasMoreTokens()) {

tok = new StringTokenizer(in.readLine());

}

return tok.nextToken();

}

static String nextLine() throws IOException {

tok = new StringTokenizer("");

return in.readLine();

}

static boolean hasNext() throws IOException {

while (tok == null || !tok.hasMoreTokens()) {

String s = in.readLine();

if (s == null) {

return false;

}

tok = new StringTokenizer(s);

}

return true;

}

public static void main(String args[]) {

try {

in = new BufferedReader(new InputStreamReader(System.in));

out = new PrintWriter(new OutputStreamWriter(System.out));

//in = new BufferedReader(new FileReader("input.in"));

//out = new PrintWriter(new FileWriter("output.out"));

solve();

in.close();

out.close();

} catch (Throwable e) {

e.printStackTrace();

java.lang.System.exit(1);

}

}

}