Talent Battle 100 Days Coding Series

There is a hallway of length $N-1$ and you have M workers to clean the floor. Each worker is responsible for segment [Li,Ri], i.e., the segment starting at Li and ending at Ri. The segments might overlap.

Every unit of length of the floor should be cleaned by at least one worker. A worker can clean 1 unit of length of the floor in 1 unit of time and can start from any position within their segment. A worker can also choose to move in any direction. However, the flow of each worker should be continuous, i.e, they can't skip any portion and jump to the next one, though they can change their direction. What's the minimum amount of time required to clean the floor, if the workers work simultaneously?

Input:

- First line will contain T, number of testcases. Then the testcases follow.
- Each testcase contains of M+1 lines of input.
- First line contains 22 space separated integers N, M, length of the hallway and number of workers.
- Each of the next M lines contain 2 space separated integers Li, Ri, endpoints of the segment under *ith* worker.

Output: For each testcase, output in a single line minimum time required to clean the hallway or −1 if it's not possible to clean the entire floor.

Sample Input

3

10 3

1 10

1 5

6 10

10 1

2 10

10 2

5 10

1 5

Sample Output

3

```
-1

5
```

C++

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

int main() {
        int t;
        cin >> t;
        while(t--){
          long long n,m;
          cin >> n >> m;
          vector<pair<long long, long long> > v;
          long long li, ri;
          for(int i=0; i<m; i++){
            cin >> li >> ri;
            v.push_back(make_pair(li,ri));
          }
          sort(v.begin(),v.end());
          long long minTime = 1;
          long long maxTime = n-1;

          long long mid;
          long long ans = -1;

          while(minTime <= maxTime){
            mid = minTime + ((maxTime - minTime) / 2);
```

```
long long cur = 1, i = 0;
multiset<long long> e;

while(cur < n){

   while(i<m && v[i].first <= cur){
      e.insert(v[i].second);
      i++;
   }

   while(!e.empty() && *e.begin() <= cur){
      e.erase(e.begin());
   }

   if(e.empty()){
      break;
   }

   long long x = *e.begin();
   e.erase(e.begin());
   cur = min(cur+mid,x);

}

if(cur == n){
   maxTime = mid - 1;
   ans = mid;
} else {
   minTime = mid + 1;
```

```
                }
            }

            cout << ans <<"\n";

        }
        return 0;
}
```

Java

```
import java.io.*;
import java.lang.reflect.Array;
import java.util.*;
import java.lang.*;

class Main {
    int n,m;
    public boolean check(Segment[]ss,int x){
        int curLoc = 1;
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        int i=0;
        while ( curLoc < n ){
            while ( i < m && ss[i].l <= curLoc ){
                pq.add(ss[i].r);
                i++;
            }
            int cur = curLoc;
            while ( cur == curLoc && !pq.isEmpty() ){
                int r = pq.poll();
            curLoc = Math.max(curLoc,Math.min(r,curLoc+x));
        }
        if( cur == curLoc)break;
    }
    return curLoc==n;
}
public void solve(){
    FastScanner fs = new FastScanner();
    PrintWriter out = new PrintWriter(System.out);
    int tc = fs.nextInt();
    while(tc-- >0 ){
        n = fs.nextInt(); m = fs.nextInt();
        Segment[]ss = new Segment[m];
        for(int i=0;i<m;i++){
            int l = fs.nextInt(), r = fs.nextInt();
            ss[i] = new Segment(l,r);
        }
        Arrays.sort(ss,Comparator.comparingInt(s -> s.l));
        int ans= -1 ,l=1,r= (int)1e9;
        while ( l <= r ){
            int mid = (l+r)/2;
            if( check(ss,mid)){
                ans = mid;
                r = mid-1;
            }else{
                l = mid+1;
            }
        }
        out.println(ans);
    }
    out.flush();
}
class Segment{
    int l,r;
    Segment(int l,int r){
        this.l = l;
        this.r = r;
    }
}
public static void main(String[]args){
    try{
        new Codechef().solve();
    } catch (Exception e){
        e.printStackTrace();
    }
}
class FastScanner {
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    StringTokenizer st=new StringTokenizer("");
    String next() {
        while (!st.hasMoreTokens())
            try {
                st=new StringTokenizer(br.readLine());
            } catch (IOException e) {
                e.printStackTrace();
            }
        return st.nextToken();
    }
```

```
        String nextLine()
        {
            String str = "";
            try
            {
                str = br.readLine();
            }catch (IOException e)
            {
                e.printStackTrace();
            }
            return str;
        }
        int nextInt() {return Integer.parseInt(next()); }
        long nextLong() {return Long.parseLong(next());}
    }
}
```

Python

```
for _ in range(int(input())):
    n,m = map(int,input().split())
    v = []
    for _m in range(m):
        x,y = map(int,input().split())
        v.append([y,x])
    v.sort()
    l = 0
    r = n - 1
    cnt = -1
    while(l <= r):
```

```
    mid = (l+r)//2
    seg = 1
    for i in range(m):
        if(seg >= v[i][0] or v[i][1] > seg):
            continue
        else:
            seg = min(seg + mid,v[i][0])
    if(seg == n):
        cnt = mid
        r = mid - 1
    else:
        l = mid + 1
print(cnt)
```