**Seating Arrangement**

**Easy**

You are given an integer **n**, denoting the number of people who needs to be seated, and a list of **m** integers **seats**, where **0** represents a vacant seat and **1** represents an already occupied seat.

Find whether all **n** people can find a seat, provided that no two people can sit next to each other.

**Example 1:**

**Input:**

n = 2

m = 7

seats[] = {0, 0, 1, 0, 0, 0, 1}

**Output:**

Yes

**Explanation:**

The two people can sit at index 0 and 4.

**Example 2:**

**Input:**

n = 1

m = 3

seats[] = {0, 1, 0}

**Output:**

No

**Explanation:**

There is no way to get a seat for one person.

//{ Driver Code Starts

import java.io.\*;

import java.util.\*;

class CodingMaxima {

public static void main(String[] args) throws IOException {

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

PrintWriter out=new PrintWriter(System.out);

int test = Integer.parseInt(in.readLine().trim());

while(test-- > 0){

int n = Integer.parseInt(in.readLine().trim());

int m = Integer.parseInt(in.readLine().trim());

String s[]=in.readLine().trim().split(" ");

int [] seats = new int[m];

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

seats[i] = Integer.parseInt(s[i]);

}

Solution obj = new Solution();

boolean res = obj.is\_possible\_to\_get\_seats(n, m, seats);

String \_result\_val = (res) ? "Yes" : "No";

out.println(\_result\_val);

}

out.close();

}

}

// } Driver Code Ends

class Solution {

public static boolean is\_possible\_to\_get\_seats(int n, int m, int[] seats) {

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

if(seats[i]==1){

if(i-1>=0)

seats[i-1]=2;

if(i+1<m){

if(seats[i+1]==0)

seats[i+1]=2;

}

}

}

int count=0;

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

if(seats[i]==0){

count++;

if(i-1>=0)

seats[i-1]=2;

if(i+1<m){

seats[i+1]=2;

}

}

}

return count>=n;

}

}