```
//Dynamic Programming
//Subset Sum Problem
//Complexity O(n^2)
#include<bits/stdc++.h>
using namespace std;
int main()
{
    long t,n,value; //t is number of
    testcases, n is number of coins, v is
    value to be formed
    cin>>t;
    while(t--)
    {
        cin>>n>>value;
        long input[n]; //we will store the
        available coin values in an array;
        for(int i=0;i<n;i++)
        {
            cin>>input[i];
        }
        sort(input,input+n); //then we sort
        the array because in the
        memoization matrix coin values are
        like {0,1,3,4,6,1,9}
        bool matrix[n+1][value+1]; //we
        create a matrix, n+1 and value+1
        because 0 is also included
        for(int i=0;i<value+1;i++) //we
        have to fill the first row with
        false and (0,0) as true, because
        only zero can be formed with 0
        {
            matrix[0][i]=false;
        }
        matrix[0][0]=true;

        long newvalue; //newvalue takes the
        value of the "new element to be
        considered(available coins)
        for(int i=1;i<n+1;i++)
        {
            newvalue=input[i-1]; //since i
            starts from 1, and available
            values are from 0, hence i-1
            for(int j=0;j<value+1;j++)
            {
                if(matrix[i-1][j]==true)
                //if above row cell is T ,
                answer will always be T,
                regardless of newvalue
                {
                    matrix[i][j]=true;
                }
                else if(j==newvalue) //if
                newvalue is equals to
                coloumn_value(current
                required sum), it can
                directly be formed, hence
                true
                {
                    matrix[i][j]=true;
                }
                else if(newvalue>j) //if
                newvalue is greater than
                coloumn_value, it wont'
                make any effect on the
                result, hence we copy the above cell
                {
                    matrix[i][j]=matrix[i-1][j];
                }
                else if(newvalue<j) //if new_value(suppose 5) is less than coloum_value(suppose 6)
                {                  //the problem boils down to: can 6-5(i.e 1) be formed excluding the new_value
                    if(matrix[i-1][j-newvalue]==true) //if so true is assigned
                    {
                        matrix[i][j]=true;
                    }
                    else //else false
                    {
                        matrix[i][j]=false;
                    }
                }
            }
        }
        //printing the matrix
        /*for(int i=0;i<n+1;i++)
        {
            for(int j=0;j<value+1;j++)
```

```
                {
                    cout<<matrix[i][j]<<" ";
                }
                cout<<endl;
            }*/


            //finally we check if
            matrix[n][value] is true, if it is
            then sum can be fromed
            if(matrix[n][value]==true)
            {
                cout<<"1"<<endl;
            }
            else //else not
            {
                cout<<"0"<<endl;
            }
        }
        return 0;
    }
```