IPvX

1 Problem Statement

1.1 Background

IPv4 addresses are 32 bits long, typically represented by 4 integer numbers ranging 0-255, separated by periods. For instance, `155.246.89.22` is the human-readable representation of an IP address, in binary this is `10011011 11110110 01011001 00010110`. To create networks, IPv4 addresses are divided into two parts: the *network prefix* and the *host component*. Two IP addresses are on the same network if they have the same *network prefix*.

The *netmask* defines the bits that will be used to determine the network prefix portion of an IP address. In IPv4 this is represented by 32 bits where a `1` indicates that the bit is part of the network prefix and a `0` indicates it is part of the host component.

If the network prefix is 16 bits long, then the above IP address `155.246.89.22/16` is on the network `10011011 11110110 00000000 00000000`. This is found simply by performing a bitwise `AND` operation on the IP address and the netmask. The netmask, in this case, is `255.255.0.0` or `11111111 11111111 00000000 00000000` (16 leading `1`'s, hence the 16-bit network prefix).

1.2 Problem

The proposed IPvX specification takes a different approach for addressing. Whereas IPv4 may use the first (most significant) 16 bits of a 32-bit address for the network prefix, IPvX allows for addresses of length m, where $2 \leq m \leq 64$, which can use $1 \leq n < m$ bits for the network component. These n bits used for the network component (bits that are set to `1`) do not have to be the first n bits in the string, however the bits for the host component (indicated by `0`'s) must be contiguous.

`11111111 00000000 = 65,280`	valid (m = 16, n = 8)
`11111111 11111100 = 65,532`	valid (m = 16, n = 14)
`11110000 00001111 = 61,455`	valid (m = 16, n = 8)
`11110000 11110000 = 61,680`	invalid (m = 16, n = 8)

Given the specification, you must write a program to output all of the valid netmasks for a given IPvX configuration.

2 Input

The first line of input contains a single integer $\boldsymbol{P}$, ($\mathbf{1} \leq \boldsymbol{P} \leq \mathbf{1,000}$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set begins with a single line that contains $\boldsymbol{K}$, the data set number, followed by $\boldsymbol{M}$, ($\mathbf{2} \leq \boldsymbol{M} \leq \mathbf{64}$) which is the number of bits in each address and $\boldsymbol{N}$, ($\mathbf{1} \leq \boldsymbol{N} < \boldsymbol{M}$), the number of bits that make up the network prefix.

3 Output

For each data set there is a single line of output. The single line of output consists of the data set number ***K***, followed by a single space followed by, **in increasing order**, a space-separated list of the valid netmasks in their decimal representation (e.g for $\texttt{11110000}_2 = 240_{10}$, output `240`).

Note: It's okay to have leading or trailing spaces; these will simply be ignored.

4 Test Data

Input	Output
3 1 4 2 2 4 1 3 6 3	1 3 9 12 2 1 8 3 7 35 49 56

Test Case #1
In the first problem, a 2-bit network component is used in 4-bit addresses. Being as the host component must be contiguous, the valid netmasks are: $\texttt{0011} = 3_{10}$, $\texttt{1001} = 9_{10}$ and $\texttt{1100} = 12_{10}$

Test Case #2
In the second problem, a 1-bit network component is used in 4-bit addresses. Valid netmasks are: $\texttt{0001} = 1_{10}$ and $\texttt{1000} = 8_{10}$.

Test Case #3
In the third problem, a 3-bit network component is used in 6-bit addresses. Valid netmasks are: $\texttt{000111} = 7_{10}$, $\texttt{100011} = 35_{10}$, $\texttt{110001} = 49_{10}$ and $\texttt{111000} = 56_{10}$.