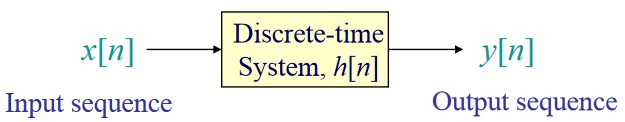
# Lab 4 – Arrays – Convolution

In this lab, each student is to write a program called **prog4.c** that convolves two discrete functions. The student should exhibit a proficiency in:

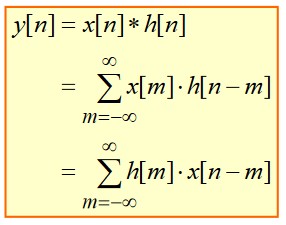
* **Array Storage and Access**
* **Dynamic Memory Allocation and Pointer Translation**
* **Basic File Operations**
* **Use of Command Line Arguments**

**Background**

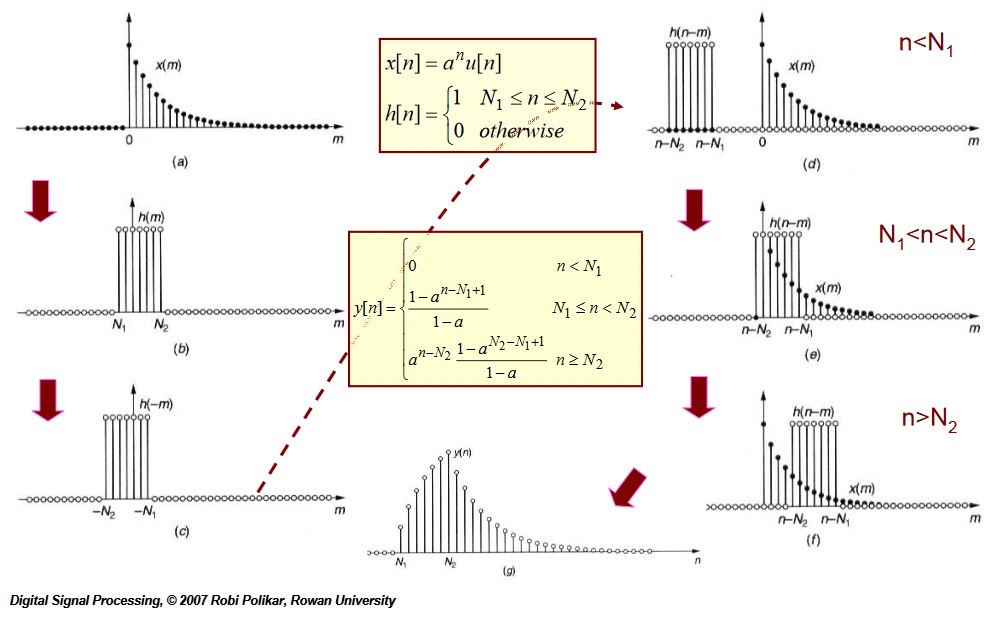
Convolution is a very important operation in electrical engineering and you will study it in **ECE 3300**. It defines the output for a discrete system with an input and a transfer function as shown:



The discrete operation is defined as either of the following summations, where ‘\*’ denotes convolution:



The diagrams below show what must be done to perform the operation. Finding the limits of the summation is the most involved part of the operation.



Luckily, your program doesn’t have to find the closed form equation for the result of the convolution operation. All it has to do is take in the data from two files, convolve them, and create another file to store the result.

**Input**

The program should accept three command line arguments with syntax show below,

**prog4 *input1.txt input2.txt output.txt***

where ***input1.txt*** and ***input2.txt*** are text files containing the data and ***output.txt*** is the file your program is to create and store the result in.

The input files will have data stored as follows:

**-3 1.000**

**-2 2.000**

**-1 3.000**

**0 4.000**

**1 3.000**

**2 2.000**

**3 1.000**

where all indices are sequential integers and the value of the functions are floating point numbers. *Each input file will contain no more than 256 data points.*

You program should open each input file, figure out how many points are contained in each, and then use dynamic memory allocation to create an array to store the points in.

Your data should use the following structure to store the input and output data:

**struct TData**

**{ int MinIndex, MaxIndex, Points;**

**float \*pValue;**

**} Input[2], Output;**

You need to use the method shown in the notes to create a vector which has indices from **MinIndex** to **MaxIndex**. For the above example of data,

**MinIndex** **= -3** and **MaxIndex** **= 3**,

and your function values should range from

**Input[0].pValue[-3] = 1.000** to

**Input[0].pValue[3]** **= 1.000**.

**Operation**

Convolution, **y[n] = x[n] \* h[n]**, is performed by reflecting and shifting one of the functions, e.g. **x[n-m]**, and then summing up the product of this function’s values with the untranslated function, e.g. **h[n]**.

For example, consider the two functions to convolve:

**h[n] x[n]**

**-3 1.000 0 0.000**

**-2 2.000 1 1.000**

**-1 3.000 2 2.000**

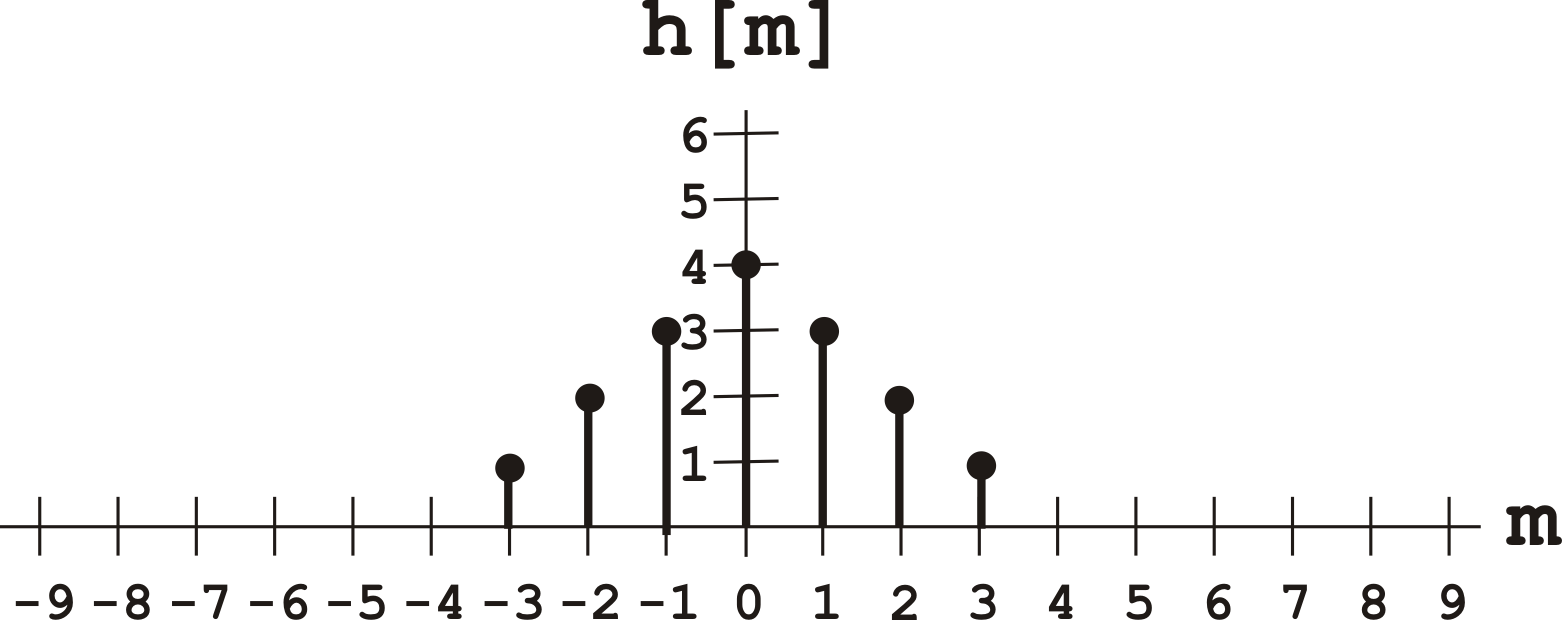
**0 4.000 3 3.000**

**1 3.000 4 4.000**

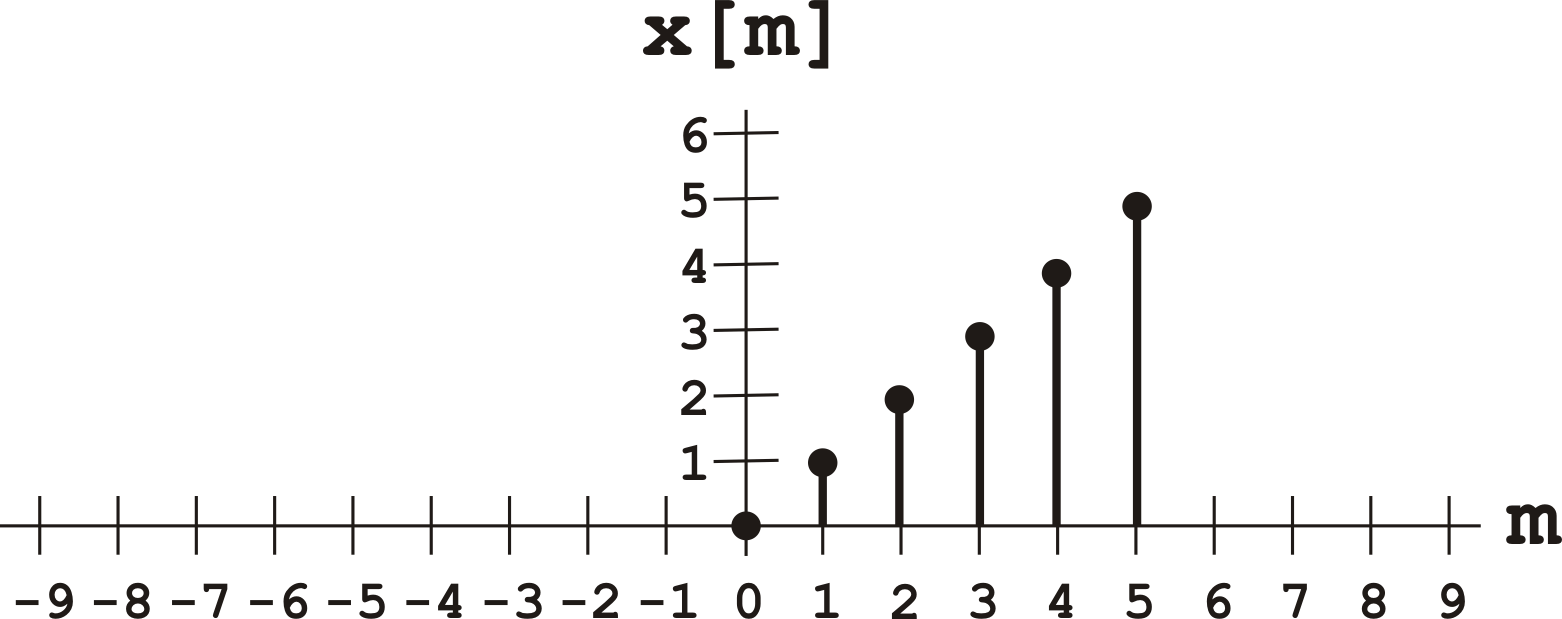
**2 2.000 5 5.000**

**3 1.000**

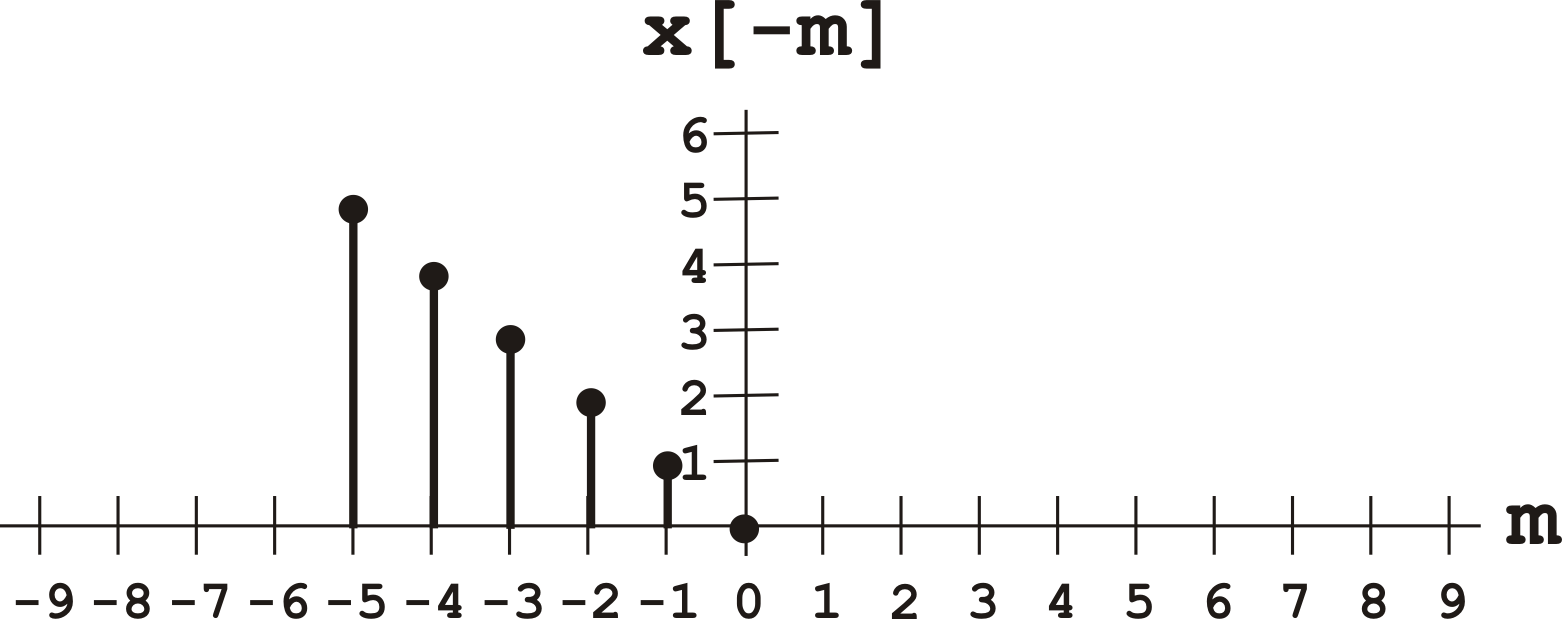
We’ll plot **h** as a function of **m**, **h[m]**



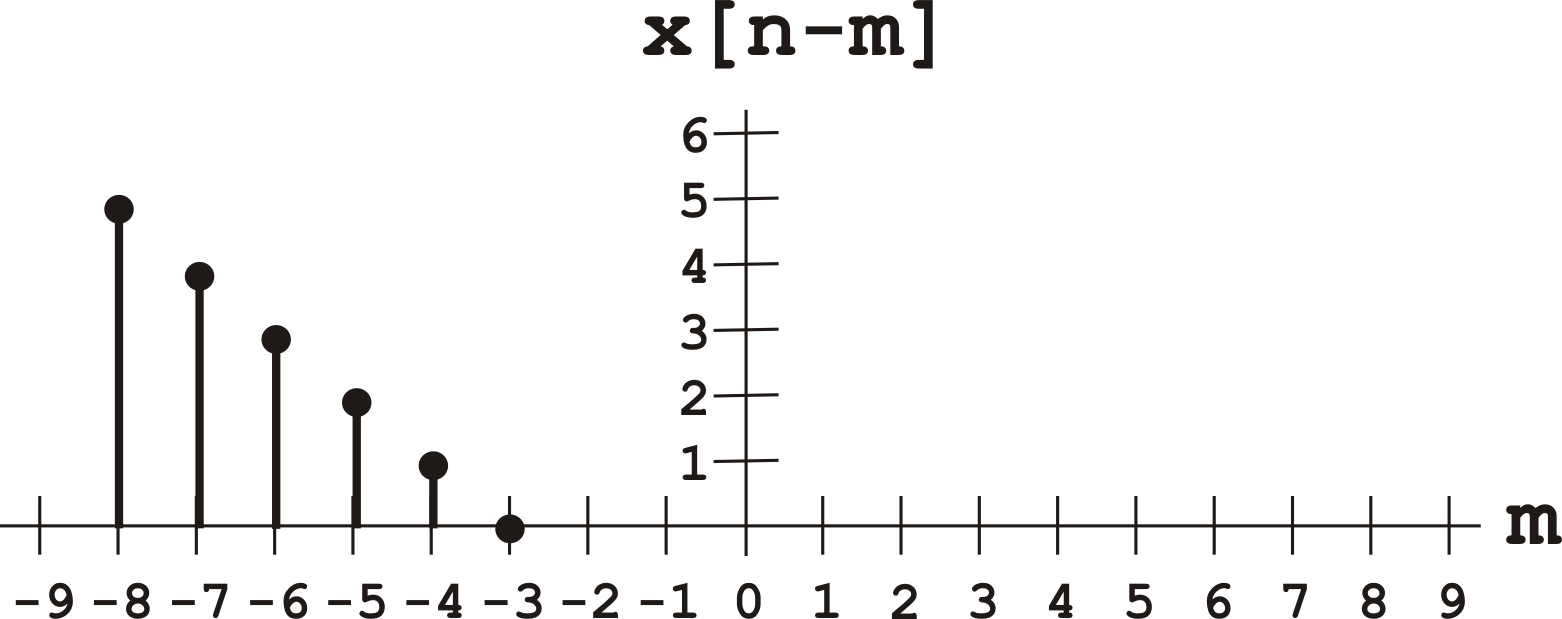
and **x** as a function of **m**, **x[m]**



Then the fun part comes when we reflect **x** to get **x[-m]**

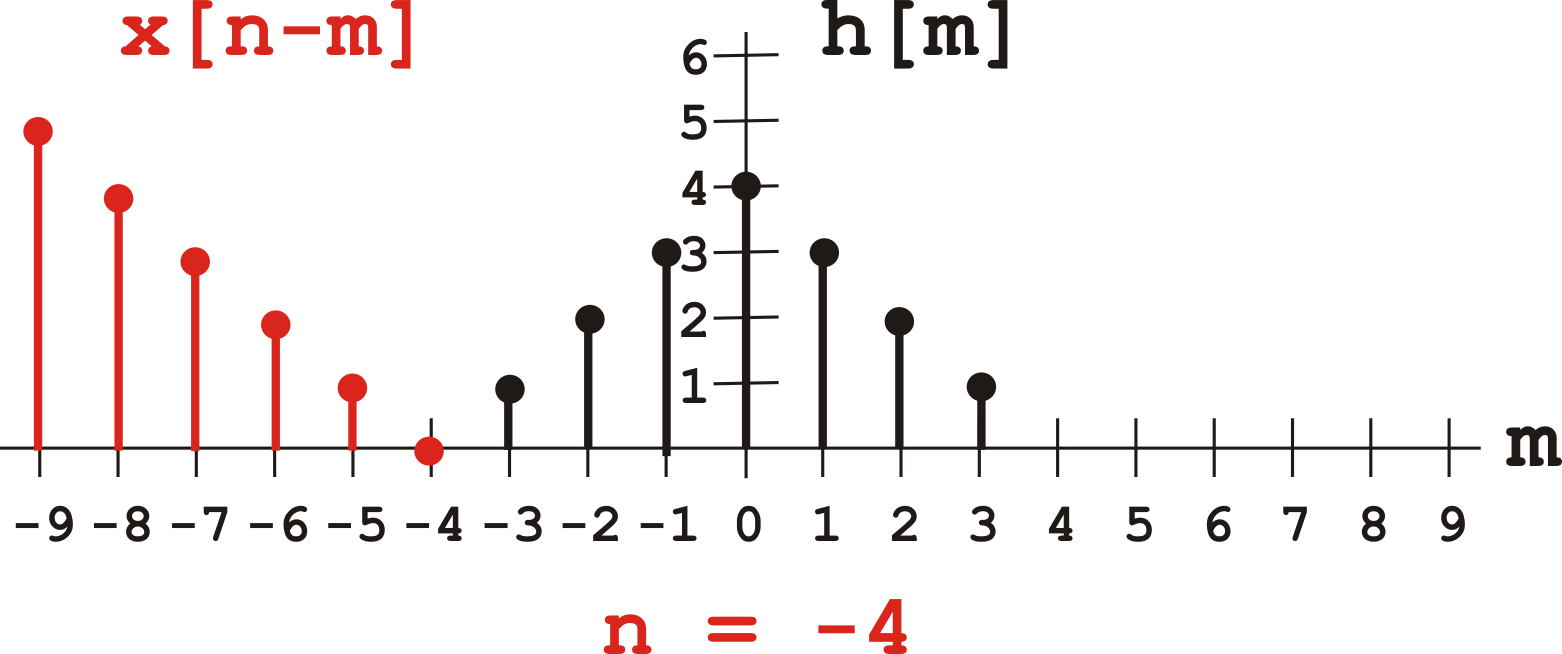
****

and then translate **x** by a constant **n** to get **x[n-m]**



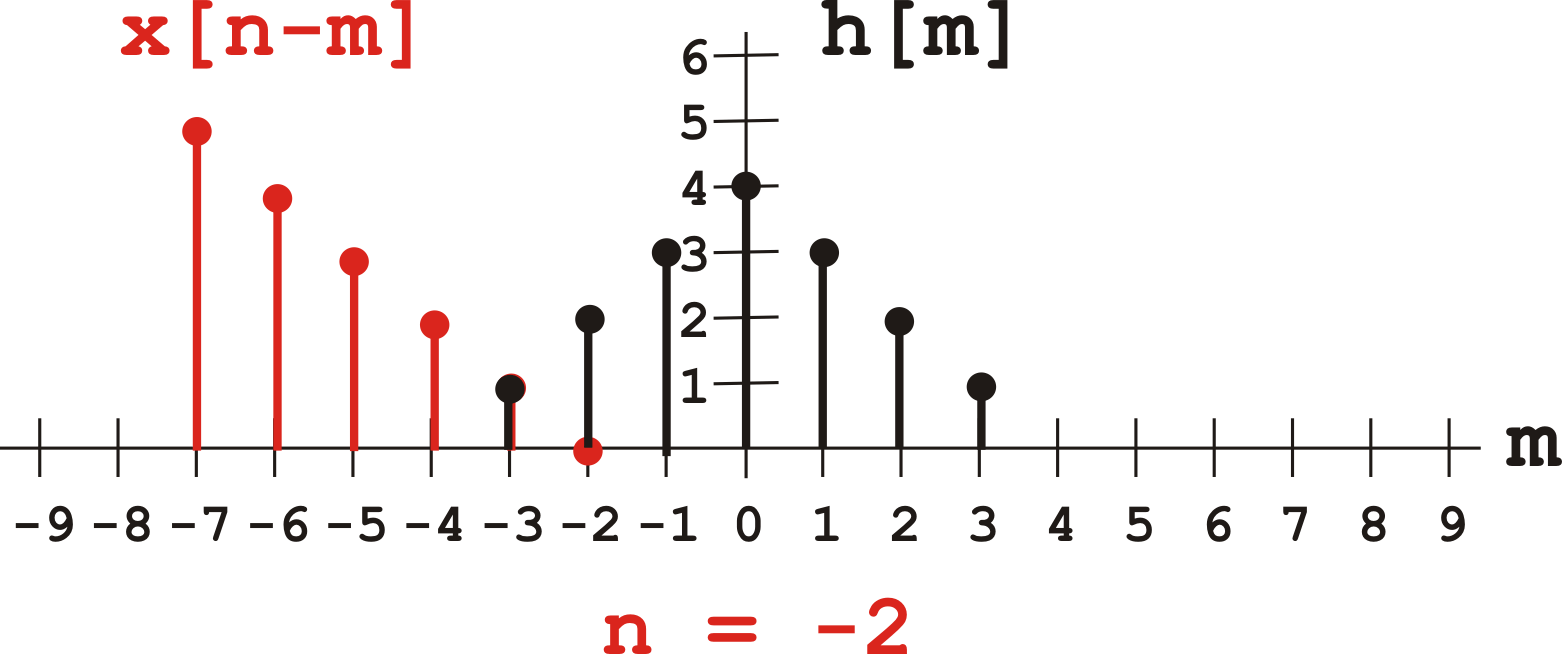
Here, **n = -3**, so **x[-3-(-3)] = x[0] = 0**, and **x[-3-(-8)] = x[5] = 5**.

Now convolution is performed by putting these functions on the same axis and translating **x[n-m]** by different values of **n** to perform a sum of the products of each function at each index:



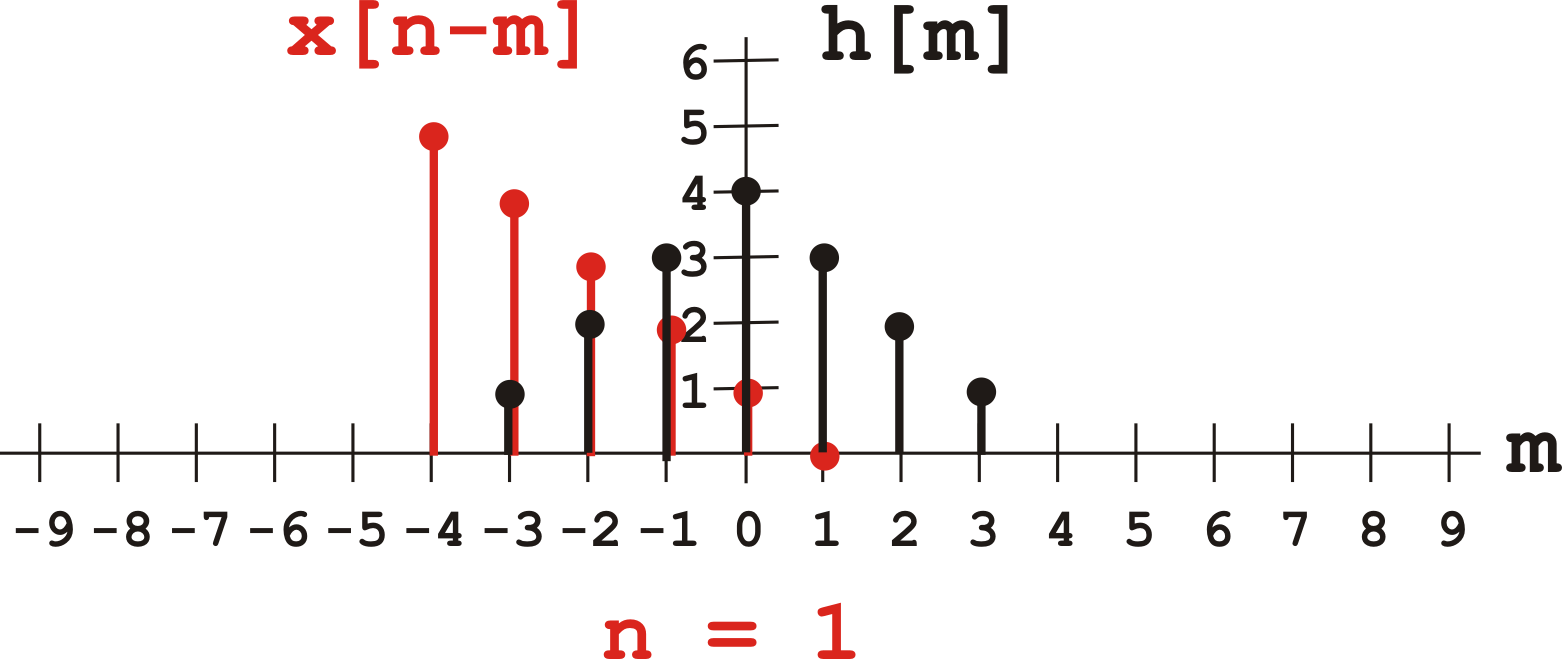
Here above, **y[n] = y[-4] = 0** since there is no overlap of the functions.

Now let’s look at **x[n-m] = x[-2-m]**:



Now we have **y[n] = y[-2] = x[-2-(-3)]\*h[-3] + x[-2-(-2)]\*h[-2] = 1\*1 + 0\*2 = 1**.

For **x[1-m]** we have



So **y[n] = y[1]**

**= x[1-(-3)]\*h[-3]**

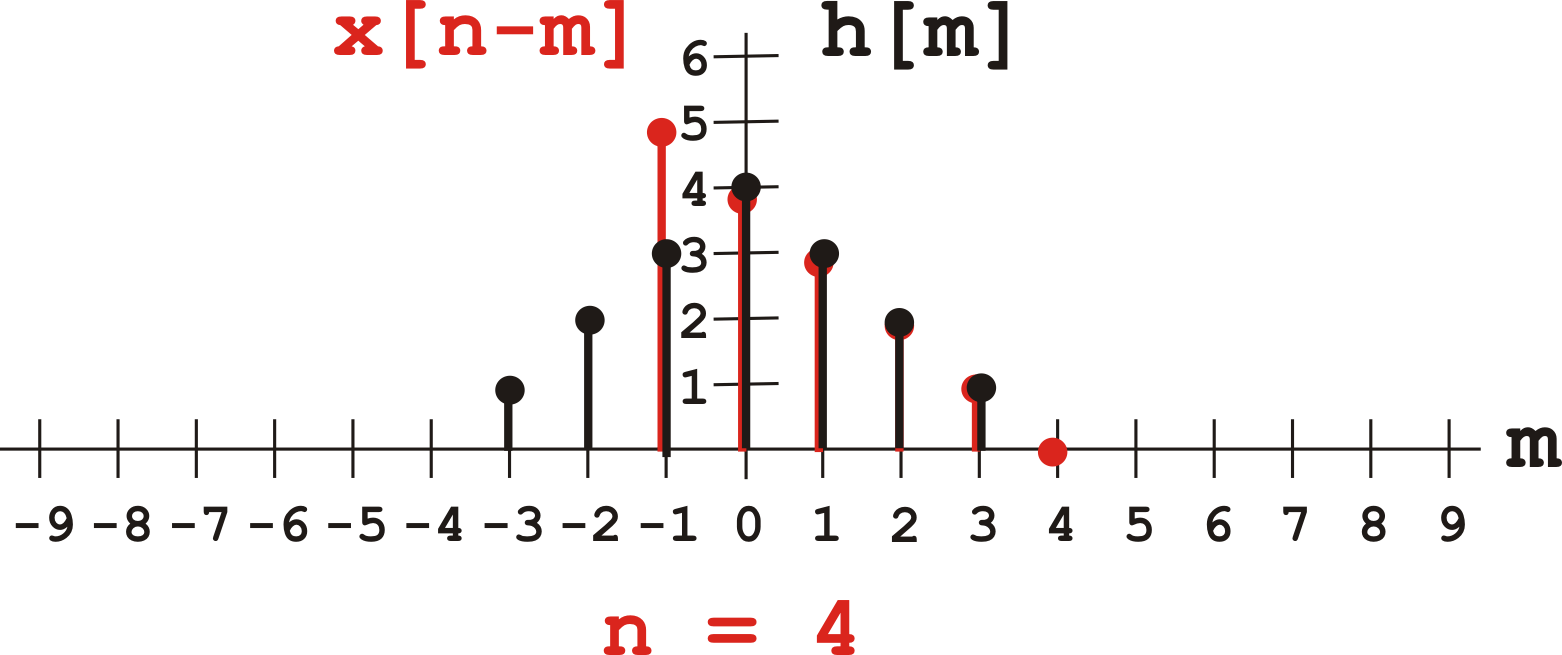
**+ x[1-(-2)]\*h[-2]**

**+ x[1-(-1)]\*h[-1]**

**+ x[1-(0)]\*h[0]**

**+ x[1-(1)]\*h[1] = 4\*1 + 3\*2 + 2\*3 + 1\*4 + 0\*3 = 20**.

Now for **x[4-m]** we have



So **y[n] = y[4]**

**= x[4-(-1)]\*h[-1]**

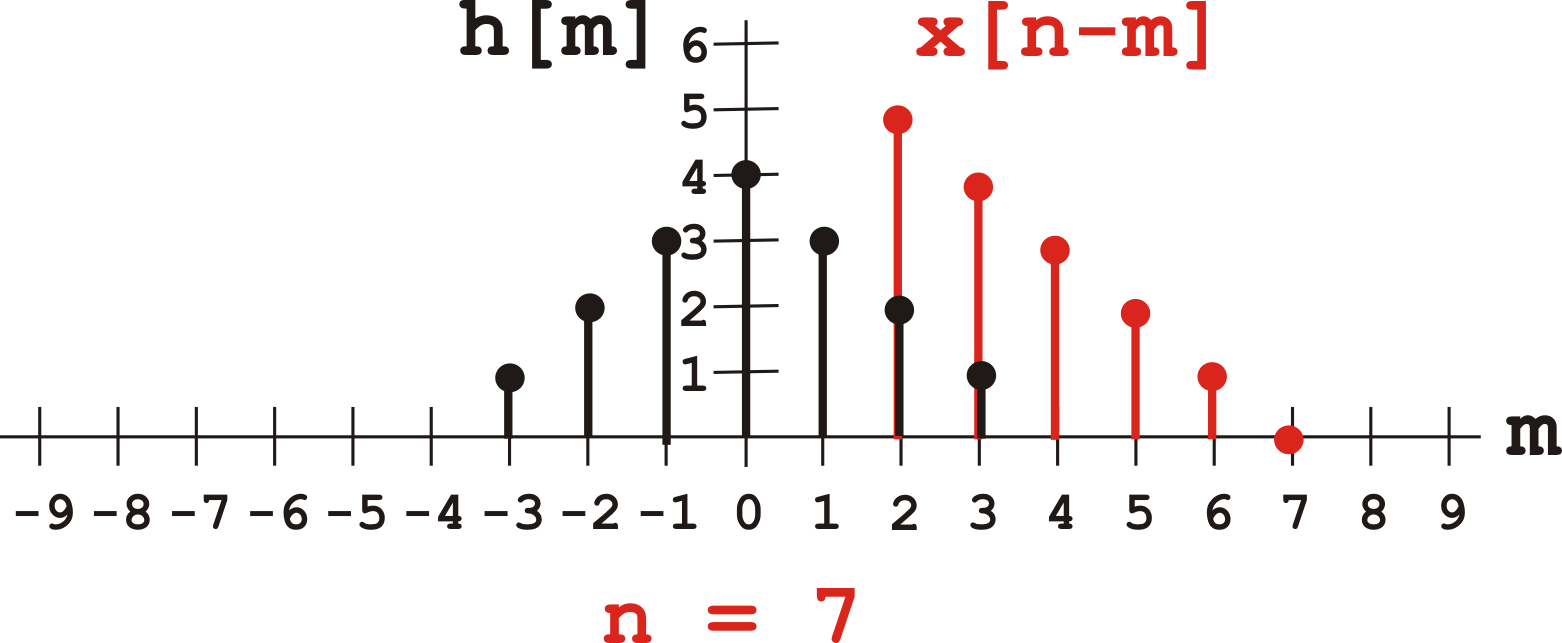
**+ x[4-(0)]\*h[0]**

**+ x[4-(1)]\*h[1]**

**+ x[4-(2)]\*h[2]**

**+ x[4-(3)]\*h[3] = 5\*3 + 4\*4 + 3\*3 + 2\*2 + 1\*1 = 45**.

And for **x[7-m]** we have

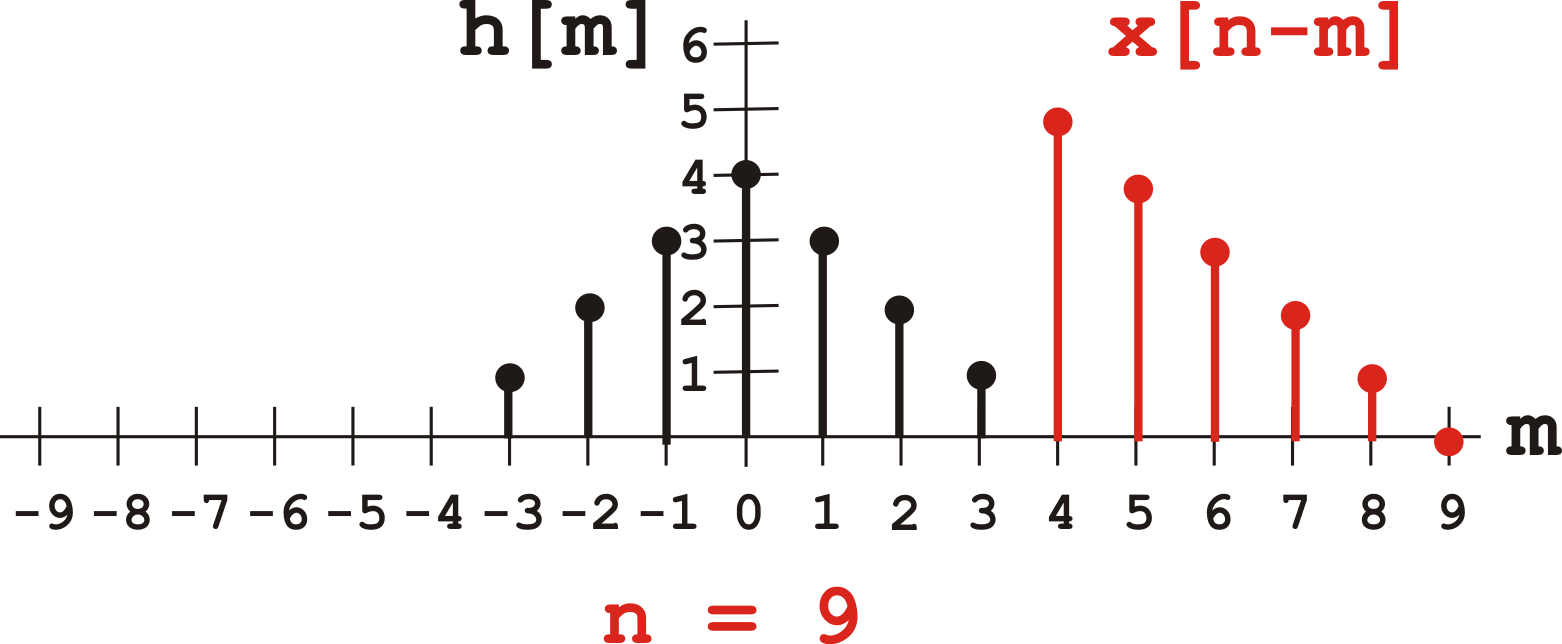


So **y[n] = y[7]**

**= x[7-(2)]\*h[2]**

**+ x[7-(3)]\*h[3]= 5\*2 + 4\*1 = 14**.

Finally for **x[9-m]** we have



And there is no overlap, so **y[9] = 0**.

Our resulting data is:

|  |  |
| --- | --- |
| **Y[n]**  **-2 1.0**  **-1 4.0**  **0 10.0**  **1 20.0**  **2 33.0**  **3 42.0**  **4 45.0**  **5 40.0**  **6 26.0**  **7 14.0**  **8 5.0** | **y[n]** |

**Output**

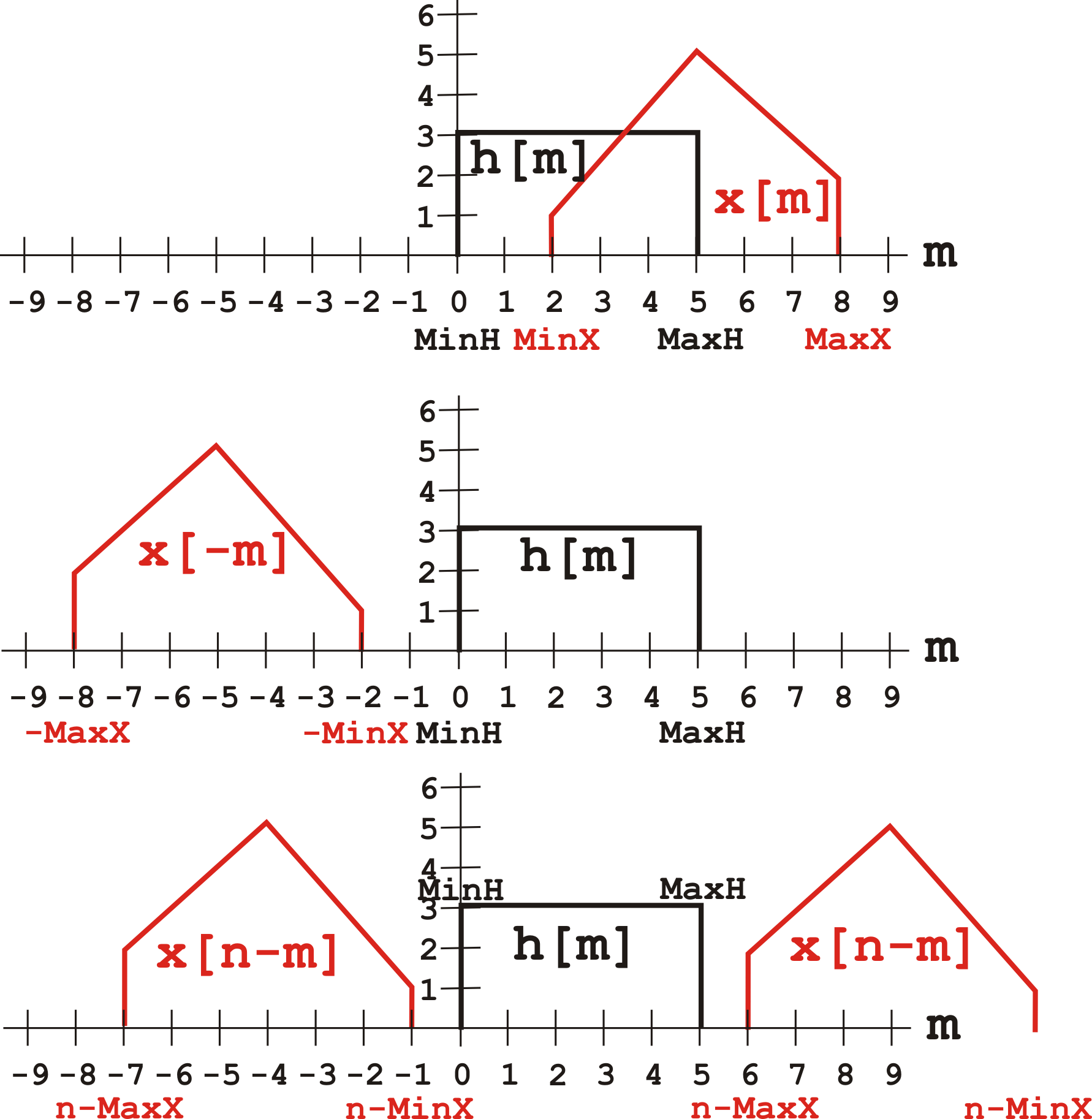
The output should have the same form as the input files, with sequential indices in the first column and the floating point output values separated from the indices by a tab as shown in the example above.

**Helpful Notes**

Discrete convolution is straightforward and simple as far as the calculation sum goes. The only interesting part comes when calculating the limits on the sum for the translated and non-translated function.

I suggest that you find a minimum and maximum value for your n index and perform the sum, but checking each time to insure the index is valid for your output array.

To do this, I provide some graphs that should make the problem much simpler to tackle.



Notice from above that when **n-MinX < MinH**, or **n < (MinH + MinX)**

and **n-MaxX > MaxH**, or **n > (MaxH + MaxX)** the result is zero, so you only have to calculate values for when **n** is between these limits. And you should keep these limits in mind when you create your output vector dynamically.

**Further Considerations**

Your program should be structured neatly, easily readable, and well commented. Furthermore, variable and function names should be such that the software is as “self-commenting” as possible.

**Creation and Submission**

***Each individual student must complete their own program. Copying other students’ code will be tested for and will not be tolerated.***

Use the following line to compile your program

**gcc -Wall -g prog4.c -o prog4**

The code you submit must compile using the **–Wall** flag and no compiler errors or warnings should be printed. To receive credit for this assignment your code must compile and at a minimum perform some required function.

Code that does not compile or crashes before performing some required function will not be accepted or graded. **All students must do a final check on one of the CES Ubuntu machines to verify that gcc shows no warning messages before submitting your project.**

Submit your program on Blackboard before midnight on **Monday, March 11th**.