Hadamard matrices *H*(*k*) are 2*k* × 2*k* matrices defined recursively for *k* = 0,1,2,... as follows.

|  |  |  |
| --- | --- | --- |
| *H*(0) | = | 1 |
|  |  |  |
| *H*(*k*) | = | *H*(*k* - 1) | *H*(*k* - 1) | for *k* ≥ 1 |
| *H*(*k* - 1) | −*H*(*k* - 1) |

Let **v** be a column vector of dimension 2*k*. The task is to compute the matrix-vector product **w** = *H*(*k*) **v**. Denote *n* = 2*k*.

In this assignment, you first implement a Θ(*n*2) algorithm to compute **w**. This is based upon an explicit construction of the matrix *H*(*k*) followed by using the standard matrix-multiplication formula. Subsequently, you take a divide-and-conquer approach in order to arrive at a Θ(*n* log *n*) algorithn for computing **w**.

Write functions that do the following.

1. Given *k*, return *H*(*k*) (a two-dimensional array of a suitable data type). This function need not be recursive.
2. Given *n*, return a random *n*-dimensional vector of floating-point numbers (a one-dimensional array of double). Each element of the vector should lie in the interval [0,1].
3. Given a matrix, print the matrix.
4. Given a vector, print the vector.
5. Given *H*(*k*) and **v**, return *H*(*k*) **v**. This function should use the standard matrix-vector multiplication formula, and run in Θ(*n*2) time.
6. Given *k* (or *n*) and **v**, return *H*(*k*) **v**. This should be a recursive function that runs in Θ(*n* log *n*) time. You do not explicitly need the matrix *H*(*k*) in this function. Indeed, if you look at every element of *H*(*k*), you end up in a Ω(*n*2) running time.

Your main() function first reads *k* from the user. It then generates *H*(*k*) and **v** by calling the first two of the above functions. *H*(*k*) and **v** are subsequently printed (by your third and fourth functions). Then, the fifth and sixth functions are called with appropriate parameters to compute **w** = *H*(*k*) **v**. After each of these two calls, the product vector **w** is printed.

**Sample Output**

k = 4

H(4) =

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1 1 -1

1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1

1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1

1 1 1 1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1

1 -1 1 -1 -1 1 -1 1 1 -1 1 -1 -1 1 -1 1

1 1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 1 1

1 -1 -1 1 -1 1 1 -1 1 -1 -1 1 -1 1 1 -1

1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1

1 -1 1 -1 1 -1 1 -1 -1 1 -1 1 -1 1 -1 1

1 1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 1 1

1 -1 -1 1 1 -1 -1 1 -1 1 1 -1 -1 1 1 -1

1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1

1 -1 1 -1 -1 1 -1 1 -1 1 -1 1 1 -1 1 -1

1 1 -1 -1 -1 -1 1 1 -1 -1 1 1 1 1 -1 -1

1 -1 -1 1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1

v =

0.8512300685

0.0207517855

0.8482068641

0.0330755948

0.8502696049

0.8616057950

0.6882541509

0.1463797275

0.8723036707

0.4770951562

0.4337855118

0.3760789113

0.1903727191

0.9634612719

0.5382524238

0.3956126549

+++ Method 1

Hv =

8.5467359110

1.9986141166

1.6274442326

-1.1160900076

-0.7216807849

2.1984352182

-0.5669766350

1.8217878630

0.0528112715

2.3536814546

0.1084376006

0.0403628079

-0.8648091456

0.0317074200

-1.2061076179

-0.6846726084

+++ Method 2

Hv =

8.5467359110

1.9986141166

1.6274442326

-1.1160900076

-0.7216807849

2.1984352182

-0.5669766350

1.8217878630

0.0528112715

2.3536814546

0.1084376006

0.0403628079

-0.8648091456

0.0317074200

-1.2061076179

-0.6846726084