1) Write a function to generate random vectors ($y$) in the $N$ dimensional binary space according to the following distribution:

$$P(y) = \prod_{i=1}^{N} p_i^{y_i} (1 - p_i)^{1-y_i}$$

$y \in \{0,1\}^N, P(y_i = 1) = p_i, P(y_i = 0) = 1 - p_i$

e.g.:

```python
def generateY(p, N):
    ...
    return y
```

2) Write a function to estimate the risk using basic Monte Carlo sampling (Risk Analysis lecture slides 57):

$$\text{risk} = P(y^T h > C) \approx \frac{1}{K} \sum_{y^T h > C} 1$$

To generate the samples, use `generateY` function.

e.g.:

```python
def calcMC(p, h, K):
    ...
    return risk
```

3) Try the algorithm for different $K$ ($K = 10, K = 100$ and $K = 1000$), then compare the mean squared error (between the real and the estimated risk) and the running times of the estimation. As a benchmark, use the Brute-force algorithm.

Hint: Measuring execution time

As before, to test the program generate an example $h$ and $p$, with $N=17$. 