

SOLUTION of (4-3.b):

C++ code 4.3.4: Solution of (4-3.b)

```
2 Eigen::VectorXd polyMult_fast(const Eigen::VectorXd &u,  
3                             const Eigen::VectorXd &v) {  
4     // Initialization  
5     const unsigned int m = u.size();  
6     const unsigned int n = v.size();  
7  
8     Eigen::VectorXd u_tmp(m + n - 1);  
9     u_tmp.head(m) = u;  
10    u_tmp.tail(n - 1).setZero();  
11  
12    Eigen::VectorXd v_tmp(m + n - 1);  
13    v_tmp.head(n) = v;  
14    v_tmp.tail(m - 1).setZero();  
15  
16    Eigen::VectorXd uv;  
17  
18    // TODO: (4-3.b) Multiply polynomials u and v efficiently.  
19    // START  
20    // Specify that real DFTs should only return/expect the conjugate  
21    // symmetric  
22    // half of the complex input/output  
23    Eigen::FFT<Eigen::FFTOption::Scaled | Eigen::FFTOption::HalfSpectrum> fft;  
24  
25    // Complex outputs only have size floor((m + n - 1)/2) + 1  
26    Eigen::VectorXcd u_tmp_ = fft.fwd(u_tmp);  
27    Eigen::VectorXcd v_tmp_ = fft.fwd(v_tmp);  
28  
29    // Since u_tmp_ and v_tmp_ are conjugate symmetric,  
30    // we can use a Complex-To-Real inverse FFT which only requires  
31    // the conjugate symmetric half + the size of the output uv  
32    uv = fft.inv(u_tmp_.cwiseProduct(v_tmp_), m + n - 1);  
33    // END  
34    return uv;  
}
```

Get it on 🐘 GitLab (polydiv.hpp).

Compare this code with [Lecture → Code 4.2.2.5], which implements the discrete convolution of two vectors. The EIGEN method `conservativeResizeLike()` is used for **zero padding** here.