## Q1

1. All the following information are gained or learned from netlib.org/blas/

Within each level, the functions are split into four categories: Single, Double, Complex and Double Complex. This categorization is purely based on the data type.

Level 1: perform scalar, vector and vector-vector operations.

Level 2: perform matrix-vector operations.

Level 3: perform matrix-matrix operations.

How to group: based on the different kinds of operations and their complexity.

1. Within the functions that are specialized as the input type is known, we could take a look at cublasSetVector() and cublasSetMatrix().

For cublasSetVector(), it copies elements from a vector in host memory space to a vector in GPU memory space. It is assumed that the elements are of bytes and column-major format. If a vector is part of a matrix, using a vector increment equals to 1 accesses a (partial) column of that matrix; using an increment equal to the leading dimension of the matrix results in accesses to a (partial) row of that matrix.

For cublasSetMatrix(), it copies a tile of elements from a matrix in host memory space to a matrix in GPU memory space. It is assumed that each element requires storage of bytes and that both matrices are stored in column-major format, with the leading dimension of the source matrix and destination matrix given in and , respectively. The leading dimension indicates the number of rows of the allocated matrix, even if only a submatrix of it is being used.

1. See mmul.cu
2. See task1.cu
3. See task1.pdf
4. The cuBLAS-based method is significantly faster than the tiled matrix multiplication in HW05. For smaller n (for example, ), cuBLAS takes only half of the time for the tiled method; for larger n (for example, ), cuBLAS only needs 10% of the time for tiled method. This clearly shows that cuBLAS has done some good optimization within its implementation.

## Q2

