2. Sample code:

```
__global__ void setColReadRowPad(int *out)
{
    // static shared memory
    __shared__ int tile[BDIMY][BDIMX + IPAD];

    // mapping from thread index to global memory offset
    unsigned int idx = threadIdx.y * blockDim.x + threadIdx.x;

    // shared memory store operation
    tile[threadIdx.x][threadIdx.y] = idx;

    // wait for all threads to complete
    __syncthreads();

    // shared memory load operation
    out[idx] = tile[threadIdx.y][threadIdx.x];
}
```

3. Sample code:

```
__global__ void setColReadRow(int *out)
{
    // static shared memory
    __shared__ int tile[BDIMX][BDIMY];

    // mapping from 2D thread index to linear memory
    unsigned int idx = threadIdx.y * blockDim.x + threadIdx.x;

    // convert idx to transposed coordinate (row, col)
    unsigned int irow = idx / blockDim.y;
    unsigned int icol = idx % blockDim.y;

    // shared memory store operation
    tile[threadIdx.x][threadIdx.y] = idx;

    // wait for all threads to complete
    __syncthreads();

    // shared memory load operation
    out[idx] = tile[irow][icol];
}
```

4.

(1)

```
Host_function(){

        //No need to copy A,B to gpu as they are already on the GPU

        Grid_dim=(1);
        Block_dim=(1024);

        Kernel_function<<< Grid_dim, Block_dim >>>( gpu_A, gpu_b,
           gpu_c);

        Mem_copy(C, gpu_c, DeviceToHost);
}
```

```
//Each thread is responsible for a single row of matrix C
Kernel_function(A, B, C) {
        my_id = threadIdx.x;

        For k = 0 to 1023
                Local_c = 0;
                For i = 0 to 1023
                        Local_c += A[my_id][i] X B[i][k]
                EndFor
                C[my_id][k] = Local_c;
        EndFor
}
```

Lower bound on the execution time is given by the data transfer time between global memory and the cuda cores for processing.

Input data A, B required by a single thread $= 1K \times 1K \times 2$. So, total input data transferred for all the threads $= 1K \times 1K \times 1K \times 2$.
Global memory bandwidth $= 100$ GB/s.
So, total input data transfer time $= \frac{1K \times 1K \times 1K \times 2}{100GB/s} = 0.02s$.

Output data C produced by a single thread $= 1K$ So, total output data produced by all the threads $= 1K \times 1K$
Total output data transfer time $= \frac{1K \times 1K}{100GB/s} \approx 1 \times 10^{-5}s$

So, total time $\approx 0.02001s$.

(2)

```
Host_function(){

//No need to copy A,B to gpu as they are already on the GPU

Grid_dim=(1);
Block_dim=(32 X 32);

Kernel_function<<< Grid_dim, Block_dim >>>( gpu_A, gpu_b, gpu_c);

Mem_copy(C, gpu_c, DeviceToHost);
}

//Each thread is responsible for a single element in each block of
   matrix C
Kernel_function(A, B, C) {
my_idx = threadIdx.x;
my_idy = threadIdx.y;

__shared__ A_s[32][32], B_s[32][32]

For iterx = 0 to 32 - 1
        For itery = 0 to 32 -1
                local_c = 0;
```

```
                For z = 0 to 32 -1
                        A_s[my_idx][my_idy] = A[iterx*32 +
                            my_idx][z*32 + my_idy]
                        B_s[my_idx][my_idy] = B[z*32 +
                            my_idx][itery*32 + my_idy]
                        __syncthreads()
                         For j = 0 to 32 - 1
                                local_c += A_s[my_idx][j]*B_s[j][my_idy]
                         EndFor
                EndFor
                C[iterx*32 + my_idx][itery*32 + my_idy] =
                    local_c
        EndFor
EndFor

}
```

Total data transferred from Global Memory to shared memory: $32 \times 32 \times 32 \times 32 \times 32 \times 2 = 2^{26}$

Global memory bandwidth $= 100$ GB/s.

So, total input data transfer time $= \frac{2^{26}}{100GB/s} \approx 64 \times 10^{-5}s$

Total data transferred from Shared memory to cuda cores: $32 \times 32 \times 32 \times 32 \times 32 \times 2 = 2^{26}$

Shared memory bandwidth $= 1$ TB/s.

so, total data transfer time to cude cores: $= \approx 2 \times 10^{-8}s$

Total output data produced by all the threads $= 32 \times 32 \times 32 \times 32$

Total output data transfer time $= \frac{32 \times 32 \times 32 \times 32}{100GB/s} \approx \quad 10^{-5}s$

So, total time = 0.00065002 s.

Other reasonable answers are also correct

5. This is correct. Threads in the same warp execute in a lockstep fashion, and therefore are always synchronized.

6.

Assume a[2048] is the array to be sorted

Host function:

```
Main( ) {

        cudaMalloc (gpu_a, sizeof(int)*2048);
        cudaMemcpy(gpu_a,  a,  sizeof(int)*2048, HostToDevice);

        dim3 dimGrid(1);
        dim3 dimBlock(1024);

        odd_even_sort<<<dimGrid, dimBlock >>> (gpu_a);

        cudaMemcpy(a,  gpu_a,  sizeof(int)*2048, DeviceToHost);
        cudaFree(gpu_a);
}
```

Kernel function:

```
__global__  odd_even_sort( int *a ){

    my_id = threadIdx.x ;
    __shared__ int  a_share[2048];

    a_share[my_id*2] = a[my_id*2];
    a_share[my_id*2+1] = a[my_id*2+1];
    _syncThread();

    for ( i=0 ; i< 2048; i++) {
        if (i is even)
            a_share[my_id*2] = smaller(a_share[my_id*2],  a_share[my_id*2+1]);
            a_share[my_id*2+1] = bigger(a_share[my_id*2], a_share[my_id*2+1]);
        else if (my_id != 1023)
            a_share[my_id*2+1] = smaller(a_share[my_id*2+1], a_share[my_id*2+2]);
            a_share[my_id*2+2] = bigger(a_share[my_id*2+1], a_share[my_id*2+2]);

        _syncThread();
    }
    a [my_id*2]    = a_share[my_id*2];
    a [my_id*2+1] = a_share[my_id*2+1];
}
```

Other reasonable answers are also correct