```
%%writefile bitonic_cuda.cu
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <cuda_runtime.h>

// ---------- Utility helpers ----------

// next power of 2 >= n
int next_pow2(int n) {
    int p = 1;
    while (p < n) p <<= 1;
    return p;
}

// read space-separated integers from a file
int read_input(const char *path, int **out) {
    FILE *fp = fopen(path, "r");
    if (!fp) {
        perror("Failed to open input file");
        return -1;
    }

    int cap = 1024;
    int size = 0;
    int val;
    int *buf = (int*)malloc(cap * sizeof(int));
    if (!buf) {
        fclose(fp);
        fprintf(stderr, "Memory allocation failed\n");
        return -1;
    }

    while (fscanf(fp, "%d", &val) == 1) {
        if (size == cap) {
            cap *= 2;
            int *tmp = (int*)realloc(buf, cap * sizeof(int));
            if (!tmp) {
                free(buf);
                fclose(fp);
                fprintf(stderr, "Memory allocation failed\n");
                return -1;
            }
            buf = tmp;
        }
        buf[size++] = val;
    }

    fclose(fp);
    *out = buf;
    return size;
}
```

```
// ---------- GPU kernel: one bitonic stage ----------

__global__ void bitonic_stage(int *data, int j, int k, int n) {
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i >= (unsigned int)n) return;

    unsigned int ij = i ^ j;  // partner index in this stage
    if (ij > i && ij < (unsigned int)n) {
        // decide whether this half should be ascending or desc
        int ascending = ((i & k) == 0);

        int a = data[i];
        int b = data[ij];

        // standard bitonic compare-swap condition
        if ((ascending && a > b) || (!ascending && a < b)) {
            data[i]  = b;
            data[ij] = a;
        }
    }
}

// ---------- main ----------

int main(int argc, char **argv) {
    if (argc < 3) {
        printf("Usage: %s <input_file> <threads_per_block>\n",
        return 1;
    }

    const char *input_path = argv[1];
    int threads_per_block = atoi(argv[2]);
    if (threads_per_block <= 0) threads_per_block = 256;

    // read input
    int *h_data = NULL;
    int count = read_input(input_path, &h_data);
    if (count <= 0) {
        fprintf(stderr, "No data read from input\n");
        return 1;
    }

    // pad to next power of two
    int n = next_pow2(count);
    int *tmp = (int*)realloc(h_data, n * sizeof(int));
    if (!tmp) {
        fprintf(stderr, "realloc failed\n");
        free(h_data);
        return 1;
    }
    h_data = tmp;
    for (int i = count; i < n; ++i)
        h_data[i] = INT_MAX;  // sentinel so they move to the e

    // allocate device memory
```

```
    int *d_data = NULL;
    cudaMalloc(&d_data, n * sizeof(int));
    cudaMemcpy(d_data, h_data, n * sizeof(int), cudaMemcpyHostT

    int blocks = (n + threads_per_block - 1) / threads_per_bloc

    // timing
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start);

    // full bitonic sorting network
    for (int k = 2; k <= n; k <<= 1) {
        for (int j = k >> 1; j > 0; j >>= 1) {
            bitonic_stage<<<blocks, threads_per_block>>>(d_data
            cudaDeviceSynchronize();  // barrier between stages
        }
    }

    cudaEventRecord(stop);
    cudaEventSynchronize(stop);
    float ms = 0.0f;
    cudaEventElapsedTime(&ms, start, stop);

    // copy result back
    cudaMemcpy(h_data, d_data, n * sizeof(int), cudaMemcpyDevic

    // write to OutputFiles/cuda_output.txt
    system("mkdir -p OutputFiles");
    FILE *out = fopen("OutputFiles/cuda_output.txt", "w");
    if (!out) {
        fprintf(stderr, "Failed to open output file\n");
        cudaFree(d_data);
        free(h_data);
        return 1;
    }
    for (int i = 0; i < count; ++i)
        fprintf(out, "%d ", h_data[i]);
    fprintf(out, "\n");
    fclose(out);

    printf("Dataset size: %d (padded to %d)\n", count, n);
    printf("Threads per block: %d, Blocks: %d\n", threads_per_b
    printf("CUDA execution time: %.6f seconds\n", ms / 1000.0f)
    printf("Sorted output saved to OutputFiles/cuda_output.txt\

    cudaFree(d_data);
    free(h_data);
    return 0;
}
```

```
Overwriting bitonic_cuda.cu
```

```
!nvcc -arch=sm_70 bitonic_cuda.cu -o cuda_sort
```

```
!./cuda_sort InputFiles/input.txt 128
!./cuda_sort InputFiles/input.txt 256
!./cuda_sort InputFiles/input.txt 512
!./cuda_sort InputFiles/input.txt 1024
```

```
Dataset size: 50 (padded to 64)
Threads per block: 128, Blocks: 1
CUDA execution time: 0.000317 seconds
Sorted output saved to OutputFiles/cuda_output.txt
Dataset size: 50 (padded to 64)
Threads per block: 256, Blocks: 1
CUDA execution time: 0.000340 seconds
Sorted output saved to OutputFiles/cuda_output.txt
Dataset size: 50 (padded to 64)
Threads per block: 512, Blocks: 1
CUDA execution time: 0.000350 seconds
Sorted output saved to OutputFiles/cuda_output.txt
Dataset size: 50 (padded to 64)
Threads per block: 1024, Blocks: 1
CUDA execution time: 0.000282 seconds
Sorted output saved to OutputFiles/cuda_output.txt
```