

## Assignment-2

# Parallel Sorting Algorithms

Sorting is an essential routine used by several downstream applications. GPUs, due to their massive parallelism, can play an essential role in realising faster versions of sorting algorithms.

We have listed some popular sorting algorithms that can be parallelised on GPUs.

SI No	Algorithm Name	Reference
1.	Radix Sort	<a href="https://gpuopen.com/download/Introduction_to_GPU_Radix_Sort.pdf">https://gpuopen.com/download/Introduction_to_GPU_Radix_Sort.pdf</a>
2.	Bitonic Sort	<a href="https://www.geeksforgeeks.org/dsa/bitonic-sort/">https://www.geeksforgeeks.org/dsa/bitonic-sort/</a> <a href="https://developer.nvidia.com/gpugems/gpugems2/part-vi-simulation-and-numerical-algorithms/chapter-46-improved-gpu-sorting">https://developer.nvidia.com/gpugems/gpugems2/part-vi-simulation-and-numerical-algorithms/chapter-46-improved-gpu-sorting</a>
3.	Sample Sort	<a href="https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5470444">https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5470444</a>
4.	Quick Sort	<a href="https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&amp;arnumber=10094180">https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&amp;arnumber=10094180</a>
5.	Merge Sort	<a href="https://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort">https://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort</a>

## The GPU Sorting Challenge

- Pick one algorithm of your choice.
- Implement it on the GPU and optimise it as much as possible to get the maximum performance. Also, try to conserve the amount of memory consumed.
- We will publish a leaderboard (updated daily) that captures the performance of each submitted entry.
- Based on the leaderboard rankings, you should constantly strive to improve your implementation to achieve better performance.
- You can resubmit as many times till the deadline.
- The top 3 implementations under each category will receive bonus marks.

- Note: The reference resources are given to help you quickly get bootstrapped with the implementation. You are free to modify and optimise the original algorithm to outperform other submissions. All we care about is the correctness of the result and performance.

## Inputs/Outputs

The list of input sequences is provided in CSV format. Each row in the CSV file corresponds to one input sequence. The number of rows will be the number of input sequences and denoted by 'N'.

Each sequence need not be of the same length. However, it is not required to scan the sequence to determine its length. The length is indicated by the first entry in the row, followed by the corresponding entries for the sequence. There is a static upper bound on the sequence length, denoted as 'L'. The data type of the sequence entries will be `uint32`. You are expected to sort the sequences in ascending order. The results should be stored in a separate CSV file. The format will be exactly the same as the input CSV file, but the sequences are sorted.

A sample Python script (GenSequence.py) is provided for you to generate the input sequences as necessary for your testing. The script takes two input values:

- Max length of the sequence (L)
- Number of sequences (N)

Also, the sample input and output files are provided to illustrate the corresponding file/data formats. Your program should strictly accept inputs and generate the outputs in this format only.

## Directory Structure and Information

- The directory that you submit must follow the following format:
  1. The directory name must be your roll\_number.zip (in **UPPERCASE**)
  2. The directory should only contain the *main.cu* file.(The file name should be only *main.cu*)
- The TAs should only be running the executable using the following command:
 

```
./<executable_name> <path_to_the_input_file>.csv -L <length upper bound> -o <output_file_name>.csv
```
- The output file name should be <Input\_file\_name>\_output.csv. If the output file name is incorrect, the autograder will assign **zero** for the corresponding test case.
- The compilation command that will be used to compile the programs is `"nvcc main.cu -O3 -o main.out"`

- After the end of each day (After 10 PM), your ranks (along with average execution times and Peak memory usages) will be available to you in the following format.
- Notice that the private test cases will not be provided until the end of this challenge.

## Evaluation

We will test your program with different L and N values for correctness and also measure the end-to-end time taken (using the `time` command). The execution time will be the average of multiple runs.

Three types of TCs will be used with a max value for N = 30,000 and L = 128.

- o Random
- o Semi-sorted
- o Sorted

## Leaderboard logistics

- Private test cases will not be shared till the challenge ends. The test cases will remain fixed for the entire duration.
- Leaderboard will be updated regularly every day for the submissions received on that day.
- Sample Leaderboard is shown below:

Author	Execution time	Peak Global and Shared Memory Used	Category(Algo)	Rank

## References

1. <https://cds.iisc.ac.in/wp-content/uploads/ParallelSorting.pdf>
2. <https://users.wfu.edu/choss/CUDA/docs/Lecture%2010.pdf>