# Homework 1: Odd-Even Sort

Deadline: July 10, 2019, 8:00

## Goal

This assignment helps you get familiar with MPI by implementing odd-even sort. We encourage you to optimize your program by exploring different parallelizing strategies.

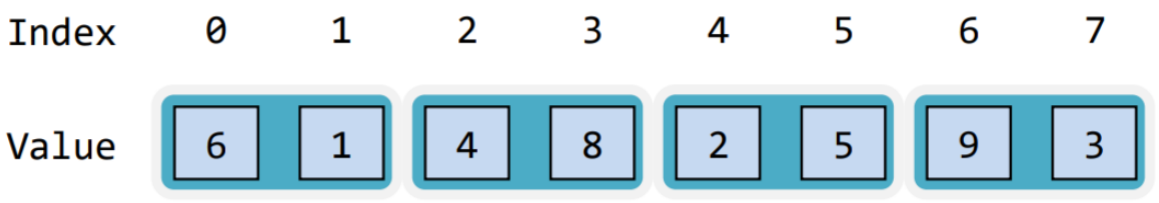
## Problem Description

In this assignment, you are required to implement odd-even sort algorithm using MPI Library **under the restriction that MPI process can only send data messages to its neighbor processes**. Odd-even sort is a comparison sort which consists of two main phases: *even-phase* and *odd-phase*.

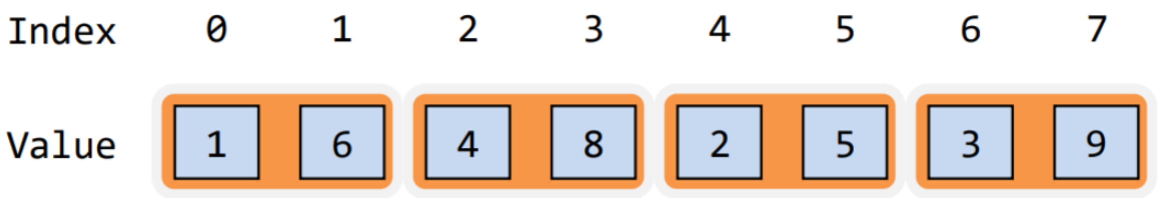
In even-phase, all even/odd indexed pairs of adjacent elements are compared. If a pair is in the wrong order, the elements are switched. Similarly, the same process repeats for odd/even indexed pairs in odd-phase. The odd-even sort algorithm works by alternating these two phases until the list is completely sorted.

In order for you to understand this algorithm better, the execution flow of odd-even sort is illustrated step by step as below: (We are sorting the list into ascending order in this case)

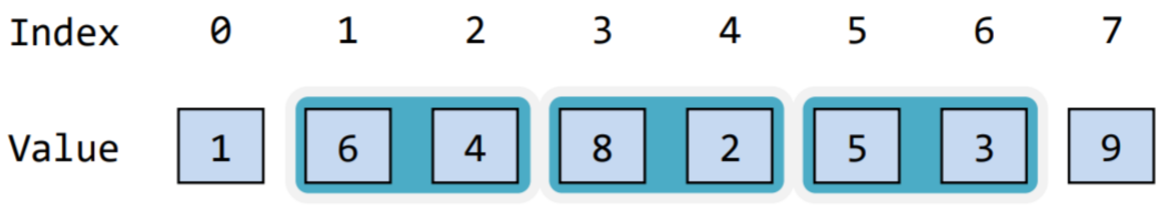
1. [Even-phase] even/odd indexed adjacent elements are grouped into pairs.



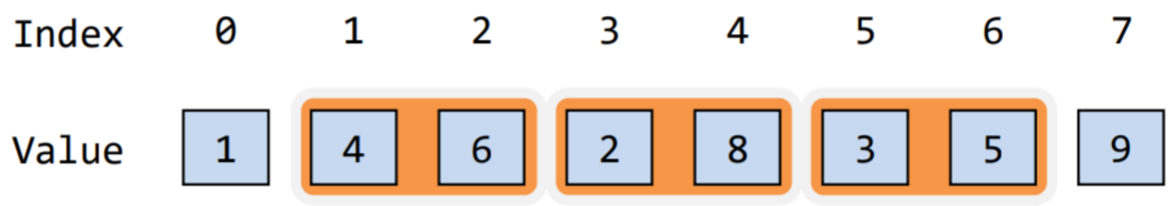
2. [Even-phase] elements in a pair are switched if they are in the wrong order.



3. [Odd-phase] odd/even indexed adjacent elements are grouped into pairs.



4. [Odd-phase] elements in a pair are switched if they are in the wrong order.



5. Run even-phase and odd-phase alternatively until no swap-work happens in both even-phase and odd phase.

## Input / Output Format

1. Your programs are required to read an input file, and generate output in another file.
2. Your program accepts 3 input parameters. They are:

    1. (integer) the size of the array n (1 <= n <= 536870911)
    2. (string) the input file name
    3. (string) the output file name

    We will run your program with an equalviant of:

    ```
    srun -nNPROC ./hw1 n inputfile outputfile
    ```

    Where `NPROC` is the number of processes
3. The input file contains n 32-bit floats in binary format. The first 4 bytes represents the first floating point number, the fifth to eighth byte represents the second one, and so on. Please refer to the sample input files
4. The output file should follow the same format of the input file. Please refer to the sample output files.

> ### Note:
>
> The float here refers to **IEEE754 binary32**, as known as **single-precision floating-point**.
>
> You can use the `float` type in C/C++.
>
> The input is guaranteed to contain none of the following:
>
> - `-INF`
> - `+INF`
> - `NAN`
>
> **Any other valid float values are possible** to show up in the input.

## Optimization Hints & Rules

- During the odd/even iterations, you can only send & receive array data to & from neighboring MPI processes. There is no restriction on how much data you can send in 1 iteration.
- There is no restriction on how the data should be sorted within an MPI process. You can use library functions for this.

- There is no restriction on how the data can be passed between MPI processes before & after the odd/even iterations (e.g. when doing I/O).
- There is no restriction on how messages other than array data (for example, the terminate condition) can be passed betwenn MPI processes.
- You can try to overlap computation time and communication time as much as possible.
- If you are not sure whether your implementation follows the rules, please discuss with TA for approval.

# Grading

1. Correctness (70%)
   - There are 40 test cases at `/home/ipl19/x/hw1/cases`. You get 1.75 points for each case passed.
2. Performance (20%)
   - You need to pass all the correctness test cases in order to get performance points.
   - Points are given according to the relative performance of your program among all the students.
3. Demo (10%)
   - Each student is given 5 minutes to explain the implementation followed by some questions from TA.
   - Points are given according to your understanding and explanation of your code, and your answers of the TA questions.

# Submission

Put your source code under at `~/homework/hw1` in `apollo31`:

- `~/homework/hw1/Makefile`
- `~/homework/hw1/hw1.cc` or `~/homework/hw1/hw1.c`

Then use `hw1-judge` to check & submit your code, **before the deadline**. You can submit **as many times as you want**.

View submissions at http://ipl.cs.nthu.edu.tw/s/hw1.

If you found any problems with the judge script or the scoreboard, contact TA immediately.

Note:

- Example `Makefile` at `/home/ipl19/x/hw1/Makefile`
- Example code for MPI-IO at `/home/ipl19/x/hw1/mpiio.cc`
- Testcases at `/home/ipl19/x/hw1/cases/`
- You can use the `hw1-floats` command to view the values in the input/output files.

Warning:

You are allowed to discuss and exchange ideas with others, but you are required to write the code on your own. You’ll got 0 points if we found you cheating.