Since the key size is fixed for datasets, we implement MSD radix sort which has O(wn) complexity, with w being key size and n being data size. To tackle skewed data distribution, after first round of radix sort, we sort big buckets in parallel by breaking them up into smaller buckets, and sort tiny buckets using std::sort to shortcut radix sort. From I/O perspective, we perform data preprocessing (separating keys and values) and partial sorting (read a trunk, and then sort it when reading the next trunk) when we read data, which is done by randomly reading in 256KB trunks, on the CPU local to main memory in the NUMA machine. When writing, we leverage Linux page cache to avoid writing to disk, and the judging program can read from page cache directly. To allow the page cache to grow uninterruptedly as the data is being sorted, we allocate memory in 4KB sizes and deallocates main memory each time we have written 500MB records to page cache. Finally, for 60GB dataset, we first sort data in 5GB trunks and write 3.5GB of each trunk to disk, before performing 12-way merge sort on all trunks, which overlaps reading sorted trunks from disk with writing merged trunks to disk.