Sorting

Wenyu Liang

February 6, 2023

1 Program description

In this program, I implement four sorting algorithms and compare their performance with data set of different size.

2 What I learned

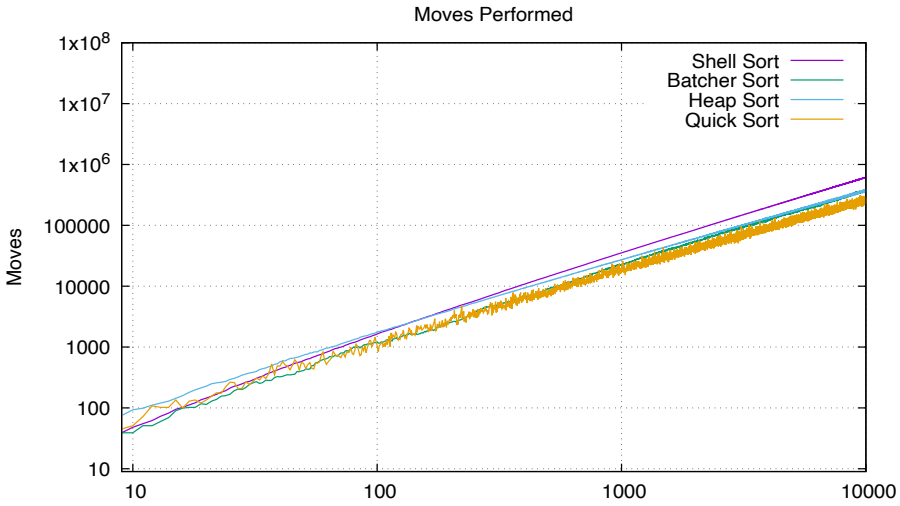
1. **heap sort:** Heap sort leverage the heap data structure to do sorting. Heap is a binary tree which has parents nodes greater and equal to child nodes. It's efficient because its binary tree characteristics, which means we can do only $\log_n$ comparisons. This is a relatively **stable** algorithm, O (n log n) in average, but it performs poorly when heap has unbalanced nodes.

2. **shell sort:** Shell sort will use another gaps sequence to help sort. The idea is to compares all the pairs in the to be sorted array that are gap indices away from each other. Pairs are swapped if they are in the wrong order. Consequently, the performance of this algorithm also highly depends on the gap sequence. This is a **unstable** algorithm, it performs poorly O(n^2) when the array has been sorted or have many duplicates.

3. **Quick sort:** Quick sort use divide and conquer to partition the array into smaller than pivot and greater than pivot, and then sort these two parts. Quick sort has O(n^2) when the pivot element chosen at at each step is either the largest or smallest in the subarray. In average cases, it's O(n log n).

4. **Batcher sort:** Batcher sort is a parallel sorting algorithm. The bit-wise operations are used to efficiently split the list into sub-lists. This is a **stable** algorithm, so the best, average, and worst-case time complexities are all O(n log n).

3 Performance comparisons using default seed



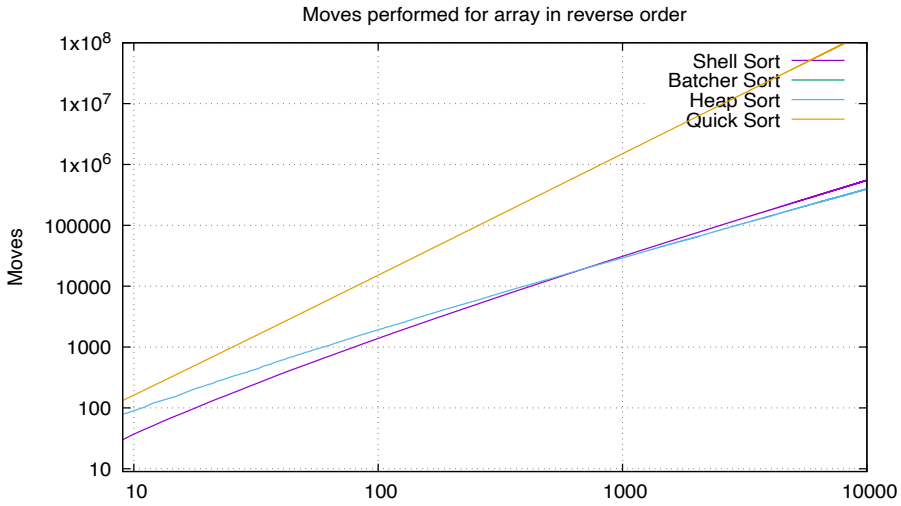
1. **Shell sort**: The moves increases linearly. It is a smooth line, which means shell sort is relatively stable with the given gap sequences. However, when the dataset becomes larger, it takes more moves than other algorithms.

2. **Heap sort**: It takes more moves when the dataset is small and when data size approaches 10000. It ranks in the middle among the 4.

3. **Batcher sort**: It's very stable when the data size is either big or small. When the data size is small, it performs the best.

4. **Quick sort**: It's clear that the moves of quick sort fluctuates very frequently, which means it's not that stable. However, it performs the best when the data size is big.

4 Performance comparisons with sorted array



1. **Shell sort**: Performs best for small dataset but eventually exceed heap sort.
2. **Heap sort**: Grows slowly, and performs the second best for large dataset.
3. **Batcher sort**: No moves for batcher sort!!!
4. **Quick sort**: Moves number goes up fastest among 4