The case for a Learned Sorting algorithm

ANI KRISTO
KAPIL VAIDYA
UGUR CETINTEMEL
SANCHIT MISRA
TIMKRASKA

Outline

- Sorting
- Learned Sort basic algorithm
- Complexities faced
- Modified Learned Sort
- Model Architecture (RMI)
- Extensions
- Evaluation
- Optimization

Introduction

Sorting is a fundamental concept in computer science which helps in many algorithms and data structures.

Sorting is an important part of the data preprocessing in machine learning algorithms

There are a wide variety of sorting algorithms which works well both for small keys and large keys.

Learned Sort

- Is a distribution sort which takes an array of elements and uses a learned model of CDF of the data.
- Output is predicted using the trained empirical CDF model.
- Out-performs sequential Radix Sort , Tim Sort being tested upon 1 billion normally-distributed double-precision keys.

A first Learned Sort

Algorithm 1 A first Learned Sort

Input A - the array to be sorted
Input F_A - the CDF model for the distribution of A
Input o - the over-allocation rate. Default=1
Output A' - the sorted version of array A

1: **procedure** LEARNED-SORT(A, F_A, o)
2: $N \leftarrow A$.length
3: $A' \leftarrow$ empty array of size $(N \cdot o)$
4: **for** x **in** A **do**
5: pos $\leftarrow \lfloor F_A(x) \cdot N \cdot o \rfloor$
6: **if** EMPTY($A'[pos]$) **then** $A'[pos] \leftarrow x$
7: **else** COLLISION-HANDLER(x)
8: **if** $o > 1$ **then** COMPACT(A')
9: **if** NON-MONOTONIC **then** INSERTION-SORT(A')
10: **return** A'

Q. In the intermediate steps, Insertion sort is used, what other sorting method could be used here?

Insertion sort works efficiently for nearly sorted arrays. Tim Sort is a hybrid sorting algorithm that combines merge sort and insertion sort.

Not a perfect model.. Why?

- Duplicate keys- will it be compensated by over-allocation?
- What if the keys are in the actual order and the CDF values made them out of order?
- How are we going to minimize the collisions and handle the sorting order of non-monotonic models in a better way?

There are options like:

- **Linear probing** : Checks for the next nearest empty slot to fill the value. Can misplace keys.
- **Chaining** : We can chain keys using a linked list or sub-arrays which again increases the dynamic memory allocation.
- **Spill Bucket** : The colliding keys can be separately stored in a bucket which can be sorted and merged with the actual array at the end.

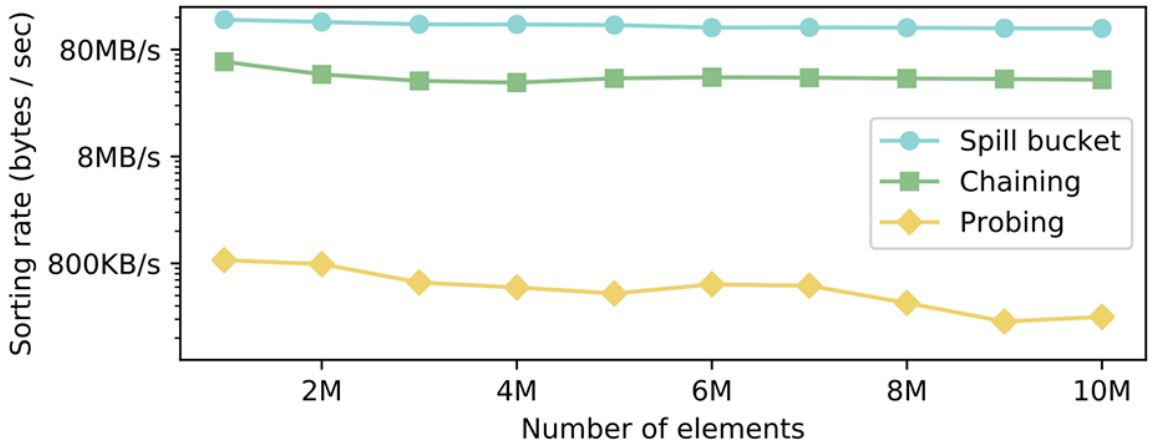
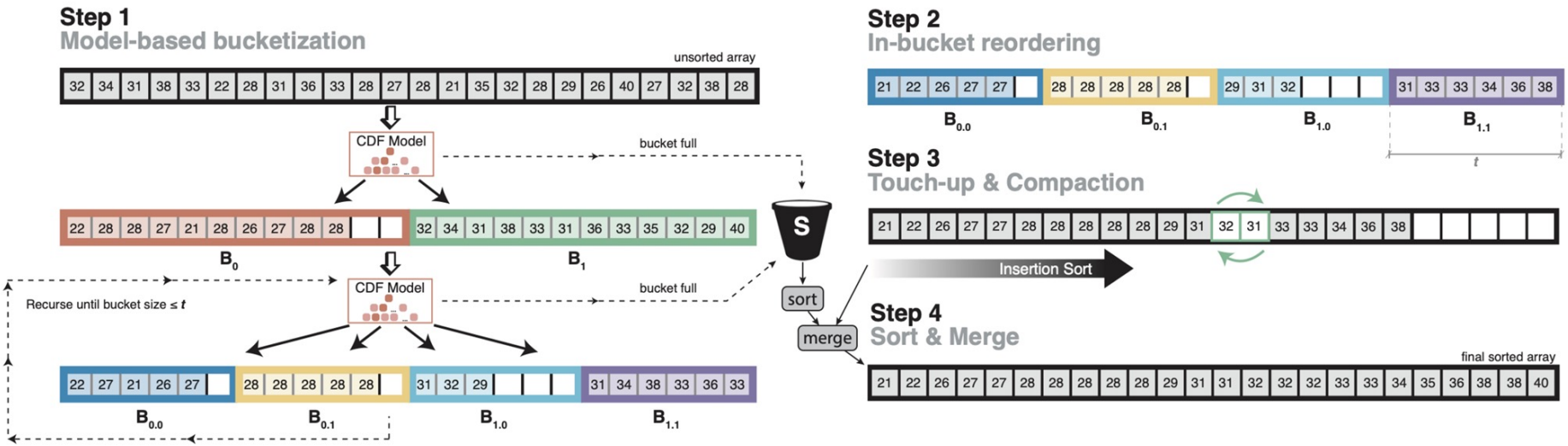


Figure 2: The sorting rate for different collision handling strategies for Algorithm 1 on normally distributed keys.

Q. There are three methods discussed to handle collision out of which the spill bucket method is chosen. Are there any other methods that can be used here?

Histograms can also be able to handle collisions by dividing the data into small buckets based on the frequency distribution of data that fits the cache-line. Only for a small number of collisions.

Final Learned Sort – Cache Optimized Learned Sorting



RMI- Recursive Model Index.

- Models like Neural Networks, order preserving hash functions may be beneficial but are too expensive to train.
- It is tree like structure which partitions data into subsets.
- These subsets are then recursively partitioned in similar way, with each subset having associated with a linear model.

Algorithm 4 The training procedure for the CDF model

Input A - the input array
Input L - the number of layers of the CDF model
Input M^l - the number of linear models in the l^{th} layer of the CDF model
Output F_A - the trained CDF model with RMI architecture

```
1:  procedure TRAIN(A, L, M)
2:      S ← SAMPLE(A)
3:      SORT(S)
4:      T ←[][][]                          ▷ Training sets implemented as a 3D array
5:      for i ← 0 up to |S| do
6:          T[0][0].add((S[i], i/|S|))
7:      for l ← 0 up to L do
8:          for m ← 0 up to M^l do
9:              F_A[l][m] ←linear model trained on the set {t | t ∈ T[l][m]}
10:             if l + 1 < L then
11:                 for t ∈ T[l][m] do
12:                     F_A[l][m].slope← F_A[l][m].slope · M^{l+1}
13:                     F_A[l][m].intercept← F_A[l][m].intercept · M^{l+1}
14:                     i ← F_A[l][m].slope ·t + F_A[l][m].intercept
15:                     T[l + 1][i].add(t)
16:     return F_A
```

Choice of the CDF model

Extensions

- Making Learned sort in-place:
- The cache we are using now is *dynamic* which helps in speeding up the sorting process
- In-place version refers to maintaining a *constant memory* for example., in the in-place version each bucket has a small buffer of size equal to the block size which can be used to sort the elements without any additional requirement of memory again.

- Duplicates:
- Spill bucket can affect the performance if it becomes too large, which can be avoided by incorporating a heuristic during the training time by tracking the repeated keys and adding them to the *exception list.*

- Learning to sort strings:

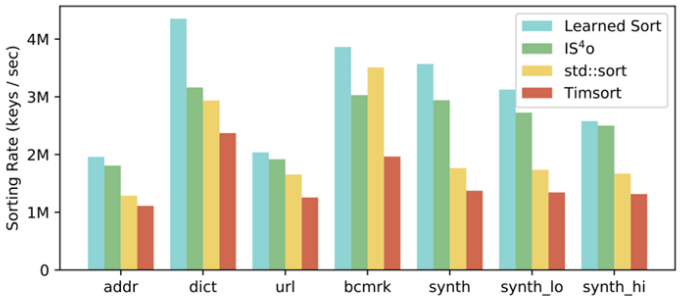


Figure 10: The sorting rate for various strings datasets.

- Excluding training time of the model.
- Has RMI architecture but taking strings as input vectors considering the length
- A sample of array is taken and and encoded using ASCII values.
- If we include the training time with the sorting time, Learned Sort still dominates the other algorithms but by a margin of of 2-8%

Q. Why is sorting Strings an issue? How come they do not tokenize strings to sort those values?

Strings can have varying lengths and can require large number of comparisons for the combinations to work considering their relative order too. Tokenizing may work for small dataset but to train a machine learning model we need a large dataset and that will be time consuming.

Q. Would a small enough neural network or deep learning model be able to achieve a good balance between model quality and runtime performance?

Smaller models may have fewer parameters and require less resources but may also reduce accuracy. Ultimately, good balance can be achieved by carefully choosing the size of the architecture and tuning the hyperparameters.

Evaluation

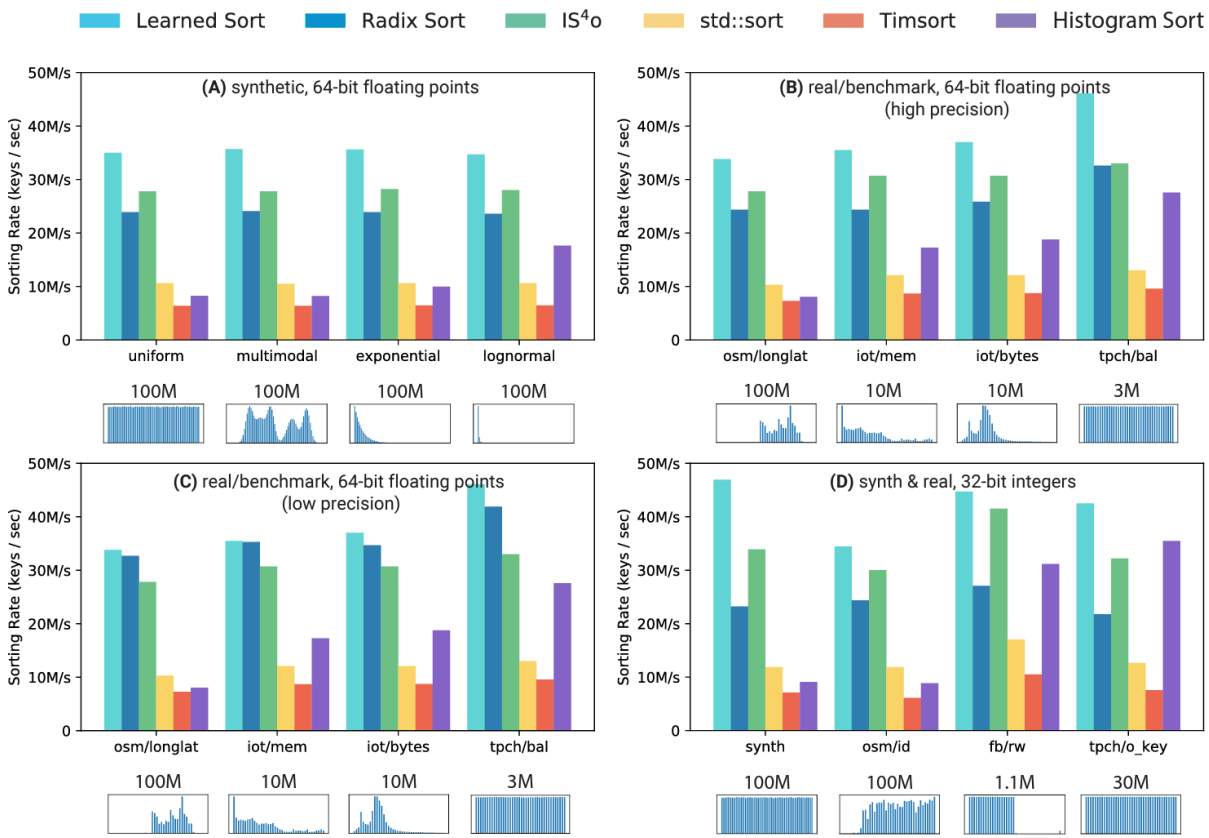


Figure 9: The sorting rate of Learned Sort and other baselines for real and synthetic datasets containing both doubles and integers. The pictures below the charts visualize the key distributions and the dataset sizes.

Optimizing- Sorting Algorithms

- The main idea is to generate an algorithm that can dynamically select the fastest version of the algorithm at the run time based on the input data.

- There can be two steps involved:

1. Tuning each sorting algorithm
2. Learning the selection function

- Let's consider learned sort as an example:

1. Find the best performing version of the Learned Sort – using empirical search – by trying out multiple hyper parameters and choosing the one's which gives the best performance.

2. Defining a set of heuristics that map input characteristics to the best version of algorithm –For example: based on the input data we will select between in-place and cache-optimized versions of the learned sort, by knowing which version adapts more to the input data and available memory.

- A machine learning algorithm called Composite sorting which uses genetic algorithm is built to optimize the sorting algorithms.

Q. How are L2 and L3 cache affecting the mapping time?

L2 is the second level of cache memory and L3 is third and is larger than L2 which is usually shares across multiple processor cores. We have to make sure that the data size is smaller than the cache size and most importantly close to the processor which makes it easy and faster to access the data to process unlike retrieving from the main memory which is much slower.

Q. What kind of modifications would need to be made to this model to handle dynamic data (inserts, deletes, merges that are typically seen in databases)?

The training of CDF model is now training the entire input data at once in the beginning as we know the data beforehand. But we can modify it by updating the CDF model after each insert or any update happening and update training sets accordingly. Also, now a new CDF model is constructed every time the input data changes. We can modify it by involving recalculation only for the slope and intercept and not the entire model.

Q. Would it be possible to apply a classification model, or even a non-linear model, to this problem?

Yes, we can apply non-linear models like Neural Networks or even multi layer Neural Networks like Transformers to perform transformations on input data but these are computationally expensive to train and complex. We may get performance improvement in certain scenarios and can directly predict the CDF values but are too expensive to handle.

Q. Give some real-world examples of this system being used and how did they benefit from it?

In theory Learned Sort achieves an average of 30% higher throughput than the next best algorithm (IS4o) and 55% as compared to Radix Sort for larger data sizes. In real-world it is still a research project and hasn't been applied to any scenarios yet.

I thank Professor Dong Dai for this opportunity and Thank you everyone!!!