Revisiting Search Procedures in Sorted Tables via Machine Learning

Giosuè Lo Bosco

Dipartimento di Matematica ed Informatica

Università di Palermo

Joint work with Domenico Amato and Raffaele Giancarlo -DMI Unipa

Outline

- Searching a Sorted Table in Main Memory
  - Context and Technology
  - Solutions
    - Good Old Binary Search: actual at least since 1590
    - Back to the Future: Interpolation Search A.D. 1957
  - Can We Beat Those Two?
    - Learned Indexes

GOAL: Methodological

Searching in a Sorted Table in Main Memory

- Binary Search Still Fundamental Problem: Big Memory, Big Data Fit in Main Memory
  - Databases
    - See VLDB Community, e.g. Rao-Ross 99, Kraska et al. 2018
    - Internet
      - Browsers, Advertisement,…, e.g., Khoung-Morin 2017

Searching in a Sorted Table in Main Memory

- Technology
  - Memory Hierarchy- L1, L2, L3 cache
  - Speculative executions, pipelining, etc
  - GPUs, TPUs

  - Branching is no good: do not ask, compute!!!

Binary Search

- Our Baseline to Compare Against
  - Binary Search [KM17]
    - For tables small enough to fit in L2 cache: branch-free binary search
    - Remaining Cases: Eytzinger layout of table with branch-free prefetching binary search
  - Interpolation Search
    - Textbook branchy-our version branch-free
- Why
  - Technically and scientifically sound work
  - Widely tested on dozens of processors
  - Even replicable work and working code

Binary Search

- ## Branch-Free Binary Search

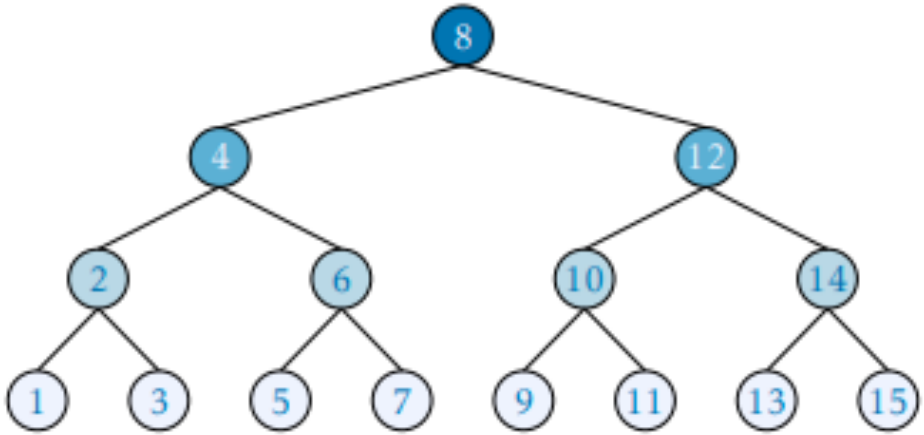
```
template<typename T, typename I>
I sorted_array<T,I>::_branchfree_search(T x) const {
    const T *base = a;
    I n = this->n;
    while (n > 1) {
        const I half = n / 2;
        base = (base[half] < x) ? &base[half] : base;
        n -= half;
    }
    return (*base < x) + base - a;
}
```

Variant of Knuth Uniform Search

Binary Search

- Eyzinger Layout-Example

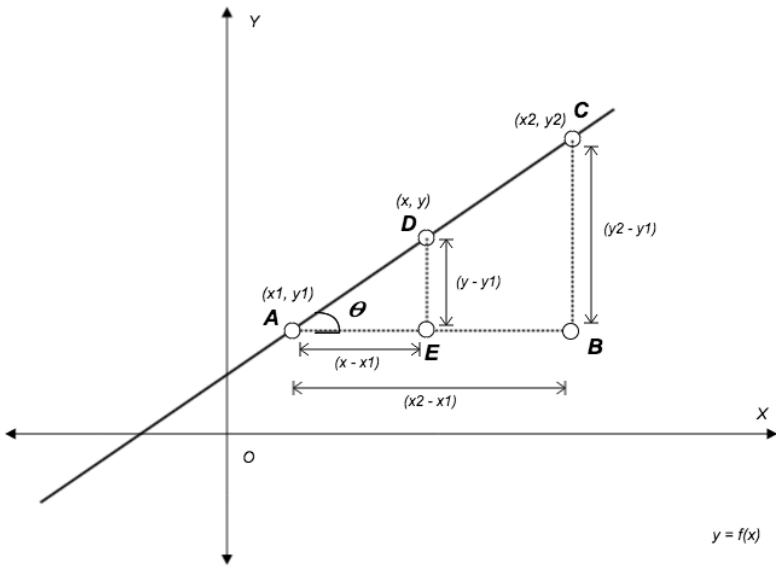
1	2	3	4	5	6	7	8	9	10	11	12	13	14



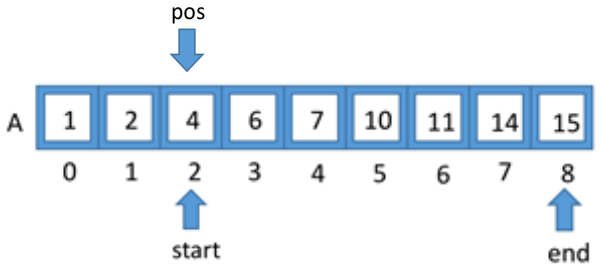
8	4	12	2	6	10	14	1	3	5	7	9	11	13	15

Interpolation Search

- Interpolation Search [P57]



$$x - x1 = (y - y1) * \frac{x2 - x1}{y2 - y1}$$



$$pos = start + \frac{(end - start)}{(A[end] - A[start])} \times (k - A[start])$$

$$k = 4$$

$$pos = 0 + \frac{8 - 0}{15 - 1} \times (4 - 1) = 1$$

$$k > A[pos] \Rightarrow start = pos + 1 = 2$$

$$pos = 2 + \frac{8 - 2}{15 - 4} \times (4 - 4) = 2$$

Interpolation Search

- ## Branch-Free Interpolation Search

```
int BranchfreeIS(int * A, int x,  int left, int right){

    const int *base = A+left;
    int i = left;
    int n = right-left+1;

    int lo = base-A;
    int hi = (base-A)+n-1;

    while (n > 1 && x>=*(A+lo) && x<=*(A+hi)) {


        lo = base-A;
        hi = (base-A)+n-1;
        int pivot = lo + (((double)(hi - lo) /
                          (A[hi] - A[lo])) *
                         (x - A[lo]));
        n = (*(A+pivot) < x) ? n-(pivot-lo+1) : n - (hi - pivot+1);
        base = (*(A+pivot) < x) ? A+(pivot+1) : base;
    }
    return (*base < x) + base - A;
}
```

Alternatives to Sorted Table Search

- ## A Short Digression into Regression

$sorted\ list\ X = \{x_1, x_2, .., x_n\}\ of\ n\ keys$

$F(z) = \dfrac{|\{y \in X | y \leq z\}|}{n}$ $\quad c.d.f$

$predecessor\ index\ of\ key\ z$

$p(z) = \lceil F(z) \times n \rceil$

x_i *independent variable*

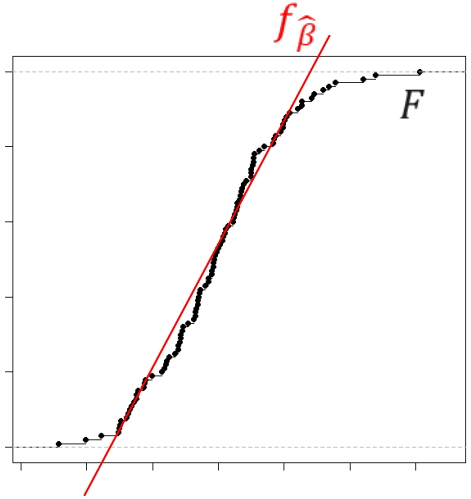
$\Downarrow$

$F(x_i)$ *outcome*

Regression problem

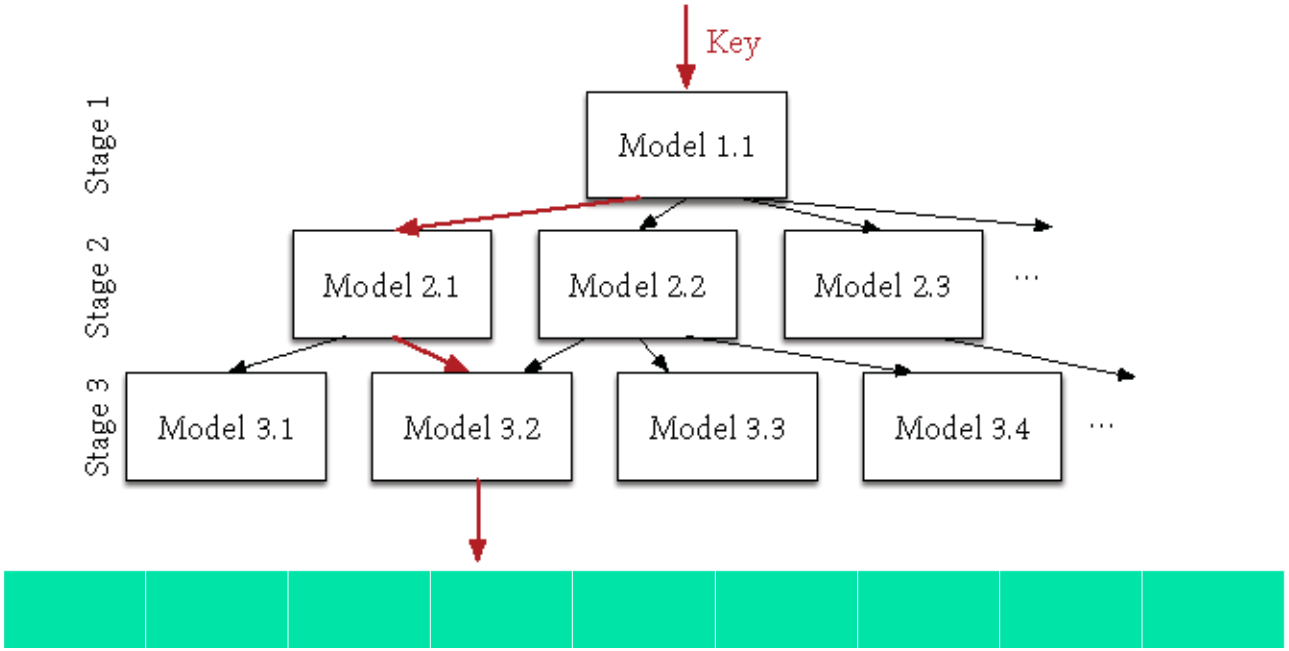
Assuming a model f_β

find $f_{\hat{\beta}} \approx F$



Alternatives To Sorted Table Search

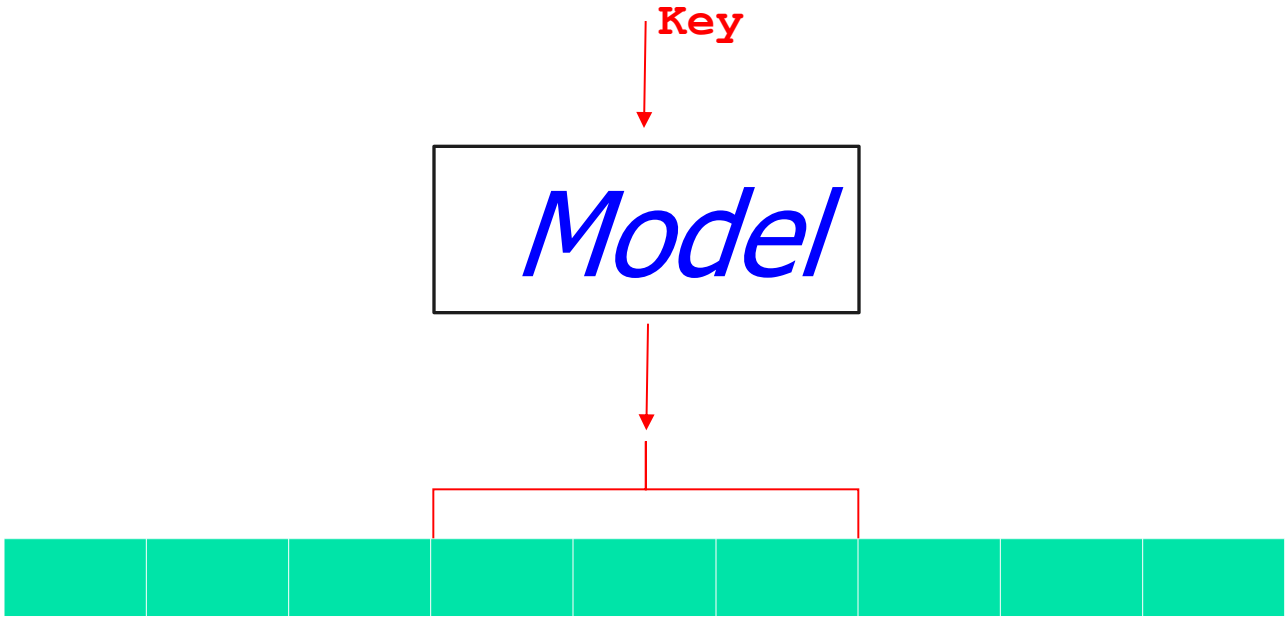
- Recursive Model Indexes-K18



- Note: a Model can be an NN or Regression.

Alternatives To Sorted Table Search

- Atomic Index



- Simple, universal in terms of learnability, very little engineering involved

Alternatives To Sorted Table Search

- Atomic Index= Sorted table search with a hint

Results-To Branch or not To Branch

- Binary Search: Always more convenient branch-free code, with prefetching [Morin]

- Our experiments
  - Binary Search: from "always better" to "no worse than".
  - Interpolation Search: Branchy somewhat better, explicit prefetching does not apply

The branchy Hamletic Doubt will not make History

Results-GPU

- GPU Operational Range
  - Small batches of queries-small table: I/O Bound
    - not competitive with respect to learned search

Query	IO (s)	Op. (s)	Pred.(s)
Uni01 50%	2.70×10^{-7}	6.27×10^{-7}	4.03×10^{-7}

Results-GPU

- GPU Operational Range
  - Large Batches of Queries and Table in the MB or small GB: I/O Bound
    - Learned search not competitive

Query	IO (s)	Op. (s)	Pred.(s)
Uni03 50%	1.89×10^{-9}	2.55×10^{-9}	4.44×10^{-9}

Results

- Open: Learned search on tables that
  - Fit in main memory
  - Do not fit in GPU
- From Now On: CPU only with data in the few GB or no GPU

Results

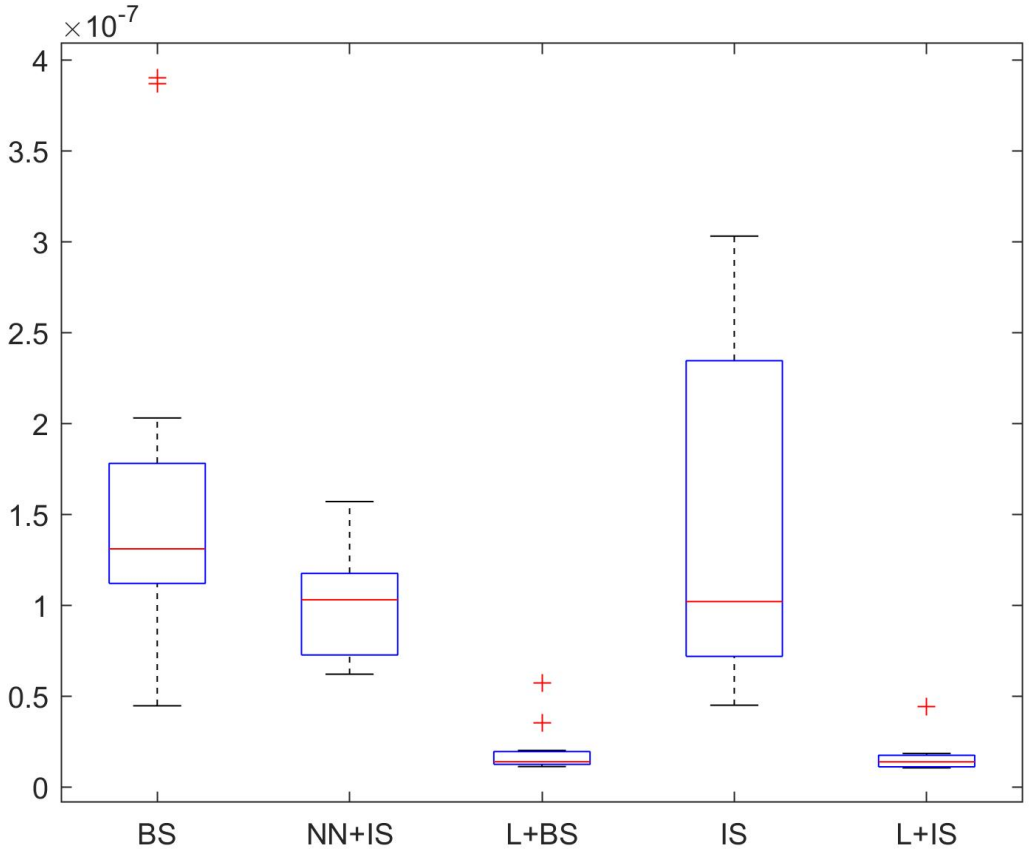
- What Model Complexity can we afford ?

- Linear Models
  - Multi-variate Linear Regression
  - NN0 ReLU
- Non-Linear Models
  - NN1 and NN2

…and the winner is: ax+b

Results: Query

- Binary/interpolation + Simple Univariate Linear Regression



Results: Query

- Binary/interpolation + Simple Univariate Linear Regression
  - Coherent speed-up of the basic search procedure, i.e., LR oracle is a booster
  - LR is stabilizer for Interpolation Search
- Curiosity: Boosting Effect also for Sequential and Randomized Binary Search

Conclusions

- Good Intuition on Kraska et al part
  - GPU architectures are methodologically irrelevant: anyone can use them, including standard binary search
  - Simplicity in Prediction pays-off
  - Nothing really new in terms of the "power of learning": Very specific…

Open

- Layouts and Learning: Which is more convenient- Theoretic Interest Only
- Really Large Datasets, e.g., GB of elements in table, MB of elements for query
  - Does the boosting effect fade ?
  - Do we need more prediction queries ?
- What can we learn from Query processing, e.g., to help branch and memory predictions ?