Tester for Nearly-Sortedness and its Applications in Databases

Arie Matsliah

Sagi Ben-Moshe
Eldar Fischer
Yaron Kanza
Outline

• New definition: “nearly-sorted”

• **Tolerant tests** for the property of being nearly-sorted

• Applications in **Databases**
## Example (data)

<table>
<thead>
<tr>
<th>Address</th>
<th>Fee ($)</th>
<th>Size (m²)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main rd 29, ...</td>
<td>500</td>
<td>70</td>
<td>...</td>
</tr>
<tr>
<td>First ave 3, ...</td>
<td>850</td>
<td>120</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Apartments for rent
select all apts. in NY, having size between 70 and 80 m², present sorted by monthly fee
Example (processor)

parse query
...
select apts. of right size
...
sort result according to fee
...
return result

Random access is expensive
Sequential pass is cheap
In-memory computation is cheapest
Facts

• Some queries/operations can be processed more efficiently if the data is ordered (sorted according to some attribute).

• Additional examples: natural join, intersection, union, except, ...

• However, in many cases processor cannot assume the data is ordered.
Observation (from experiments)

- Monitoring the “sort” function of a DB-management system (PostgreSQL)

- In many cases, even before sorting the data is “nearly sorted”

- Idea:
  1. test whether the data is “nearly sorted”
  2. if it is – use sorting algorithm that is tailored for nearly-sorted data
Ingredients

1. **property tester** for the property of being *nearly-sorted*
   * few queries = **few random access** to data
   * must be **tolerant**

2. **efficient** sorting algorithm that works if the data is *nearly-sorted*
   * no random access, **few** sequential passes
   * **always** correct (discovers failure)
Definition: Nearly-Sorted

- \( f: [n] \rightarrow \mathbb{R} \)
- \( \mathbb{R} \) – attribute values, total order (\(<, \leq, >, \geq\))

- \( f \) is sorted if for all \( i < j, \ f(i) \leq f(j) \)
- \( f \) is \( k \)-sorted if for all \( i, j: i \leq j-k \rightarrow f(i) \leq f(j) \)
- \( 1 \)-sorted \( \iff \) sorted
- \( f \) is \( \epsilon \)-close to being sorted if
  for some \( E \subseteq [n], \ |E| \leq \epsilon n: f|_{[n]\setminus E} \) is sorted
- \( f \) is \((\epsilon, k)\)-nearly-sorted if
  for some \( E \subseteq [n], \ |E| \leq \epsilon n: f|_{[n]\setminus E} \) is \( k \)-sorted
Example 1

- $1/n$-close ($\varepsilon = 1/n$), n-sorted ($k = n$)
Example 2

- $1/2$-close ($\varepsilon = 1/2$), $2$-sorted ($k = 2$)
Example 3

• $(1/5, 2)$-nearly-sorted
• (Tolerant) Test for nearly-sortedness

• Algorithm for sorting nearly-sorted functions

• Experiments
Testing Nearly-Sortedness

$([\varepsilon_1, \varepsilon_2], [k_1, k_2])$-test:
- ACCEPT w.p. $2/3$ if $(\varepsilon_1, k_1)$-nearly sorted
- REJECT w.p. $2/3$ if not $(\varepsilon_2, k_2)$-nearly sorted

- $([0, \varepsilon], [1, 1])$-test = tester for monotonicity
  #queries = $O(\log(n)/\varepsilon)$
  [BRW, DGLRRS, EKKRRV, GGLRS, FLNRRS, HK]

- $([\varepsilon, c\varepsilon], [1, 1])$-test = tolerant tester for monotonicity
  #queries = $\tilde{O}(\log(n)/\varepsilon)$
  [PRR, ACCL]

- $([\varepsilon, c\varepsilon], [k, ck])$-test = tolerant test for nearly-sortedness
  #queries = $O(\log(n)/\varepsilon)$
  [this work]
**k-Violations and \((\delta,k)\)-Active Indices**

- \((i,j)\) is a \(k\)-violation if \(i \leq j-k\) and \(f(i) > f(j)\)
k-Violations and (δ,k)-Active Indices

- (i,j) is a k-violation if i ≤ j-k and f(i) > f(j)

- i is (δ,k)-active if for ≥ δ(j-i) indices h ∈ [i,j], (i,h) is a k-violation

50  □  ...  □  □  ...  □  5

k

δ(j-i) values < 50
k-Violations and $(\delta,k)$-Active Indices

- $(i,j)$ is a $k$-violation if $i \leq j - k$ and $f(i) > f(j)$

- $i$ is $(\delta,k)$-active if for $\geq \delta(j - i)$ indices $h \in [i,j], (i,h)$ is a $k$-violation

- $j$ is $(\delta,k)$-active if for $\geq \delta(j - i)$ indices $h \in [i,j], (h,j)$ is a $k$-violation
k-Violations and \((\delta,k)\)-Active Indices

- either \(i\) or \(j\) must be \((\frac{1}{2}k/(j-i),k)\)-active

$$\Rightarrow$$ if \(f\) is not \((\varepsilon,k)\)-nearly sorted,

\[
\# (\frac{1}{2}k/(j-i),k)\text{-actives} \geq \varepsilon n
\]
Towards tolerant testing based on [ACCL]

Lemma

• if $f$ is $(\varepsilon,k)$-nearly sorted then
  \[ \# (1/4,k)\text{-actives} \leq 5\varepsilon n \]

• if $f$ is not $(6\varepsilon,6k)$-nearly sorted then
  \[ \# (1/3,k)\text{-actives} \geq 6\varepsilon n \]
counter=0
repeat $T = O(1/\varepsilon)$ times:
    pick $i \in [n]$
    If $i$ is $1/3$-active then counter++
if (counter/T > 5.5\varepsilon) REJECT
else ACCEPT

Problem: how to check if $i$ is $1/3$-active?
Activity-testing algorithm

input: $i, \delta$
- if $i$ is $(1/3, k)$-active,
  output YES w.p. $\geq 1 - \delta$
- if $i$ is not $(1/4, k)$-active,
  output NO w.p. $\geq 1 - \delta$

query complexity: $\tilde{O}(\log(1/\delta) \log(n))$

works by approximating the number of violations (by sampling) within neighborhoods of increasing size
(\([\varepsilon,6\varepsilon],[k,6k]\))-tolerant test

count=0
repeat T=O(1/\varepsilon) times:
    pick \(i \in [n]\)
    if \(AT(i,1/T) = YES\) then count++
if (count/T > 5.5\varepsilon)
    REJECT
else
    ACCEPT

query complexity: \(\tilde{\Theta}(\log(n)/\varepsilon)\)

• if \(f\) is \((\varepsilon,k)\)-nearly sorted
  \(\Rightarrow\) fraction of \((1/4,k)\)-actives \(\leq 5\varepsilon\)
  \(\Rightarrow\) ACCEPT w.p. \(\geq 2/3\)

• if \(f\) is not \((6\varepsilon,6k)\)-nearly sorted
  \(\Rightarrow\) fraction of \((1/3,k)\)-actives \(\geq 6\varepsilon\)
  \(\Rightarrow\) REJECT w.p. \(\geq 2/3\)
Sorting Nearly-Sorted relations

• Use the Replacement-Selection algorithm

• Thm: if $f$ is $(\varepsilon,k)$-nearly-sorted, then $RS$ with $M=\varepsilon n+k$, sorts $f$ in two passes
Replacement-Selection

6 3 1 4 5 2 7 8 10 9

1
Replacement-Selection

3 6 4 5 2 7 8 10 9

1 3
Replacement-Selection

1 3 4

6 4 5 2 7 8 10 9
Replacement-Selection
Replacement-Selection

1 3 4 5 6

6 2 7 8 10 9
Replacement-Selection
Replacement-Selection
Replacement-Selection
Replacement-Selection

1 3 4 5 6 7 8 9 10

2 10

Red box highlights the positions of 2 and 10.
Replacement-Selection
Replacement-Selection

\[
\begin{array}{cccccccc}
2 & 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]
Replacement-Selection

If #marked ≤ M, the data is sorted after two passes
Lemma: \#marked \leq M (= \varepsilon n + k)

Proof:
let \( E_1, \ldots, E_t \subseteq [n] \) be "bad" subsets of size \( \leq \varepsilon n \)
let \( D \) be their intersection
clearly \( D \leq \varepsilon n \)

Claim: If \( i \) is marked, then \( i \in D \implies \#\text{marked} \leq \varepsilon n \)
Proof:
for \( i \leq \varepsilon n + k \), no index is marked
let \( i \geq \varepsilon n + k \) be marked, assume \( i \) not in \( D \implies i \) not in \( E_h \) for some \( h \)

\[ \Rightarrow E_h > \varepsilon n \] (contradiction)
Experiments

- monitoring the “sort” function of PostgreSQL

- data was \((1/\sqrt{n}, \sqrt{n})\)-nearly sorted in most cases

- testing with parameters compatible with currently typical memory size is faster than making one pass

- in-memory sorting \((6/\sqrt{n}, 6\sqrt{n}]\)-nearly sorted data with RS is >2 times faster than standard quicksort

- more elaborate tests pending...
Experiments

50,000,000 Records

<table>
<thead>
<tr>
<th>( \epsilon ) + k</th>
<th>Our algorithm</th>
<th>QuickSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>500,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5,000,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20,000,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank you