Improved Selection Algorithms for Integers in Read-only Memory and Restore Models

Venkatesh Raman

IMSc, Chennai, India

(Joint work with Timothy Chan and Ian Munro, University of Waterloo)

Shonan Meeting, September 28, 2013

Problem

- The Classical Selection Problem: Given a list of n elements (from a totally ordered set), find their median or the k-th smallest element.

- Can be solved in $< 3n$ comparisons even using $O(1)$ storage cells.

- Find the median without changing the input permutation
- Motivation: Another program may need the original permutation.

Outline

- Read-only Memory
  --- Early work
  --- New for integers

- Restore Model selection
  --- for integers

- Model
  -- The usual RAM model
  -- Space is the number of extra variables/cells

One (initial) approach

- Assume that the input is in Read-Only Memory

SELECTION AND SORTING WITH LIMITED STORAGE

J.I. Munro *

Dept. of Computer Science
University of Waterloo
Ontario, Canada.

M.S. Paterson

Dept. of Computer Science
University of Warwick
Coventry, U.K.

n selecting from,or sorting, a file stored
d-only tape and the internal storage is
imited, several passes of the input tape
equired. We study the relation between
nt of internal storage available and the
f passes required to select the K^{th} highest
uts. We show, for example, that to find
an in two passes requires at least $\Omega(N^{\frac{1}{2}})$
ost $O(N^{\frac{1}{2}} \log N)$ internal storage. For
istic methods, $\Theta(N^{\frac{1}{2}})$ internal storage is
y and sufficient for a single pass method

storage. The elements are from some
ed set (for example the real numbers)
comparison can be made at any time be
elements within the random-access sto
Initially the storage is empty and th
placed with the reading head at the b
After each pass the tape is rewound t
position with no reading permitted.

<u>Notational note</u>.

For functions of several argumen
write $f(\chi) = O(g(\chi))$ when $\exists\, c >$ such
$|f(\chi)| < c.g(\chi)$ for all χ except thos

The Munro-Paterson Paper [FOCS'78]

- P-pass streaming alg'm for exact median (or selection) with $O(n^{1/P} \log^2 n)$ words of space

- lower bound of $\Omega(n^{1/P})$ (in comparison model)

Comparisons Bound

$s > \log^2 n$

- Munro-Paterson (1978)

  $O(n \log (n/s) + n \log_s n)$

- Frederickson (1986)

  $O(n \log^*(n/s) + n \log_s n)$

- Elmasry, Juhl, Katajainen, Satti (2013)

  $O(n \log^*((n/\log n)/s) + n \log_s n)$ time

(For example, when $s = n/\log n$, $O(n)$ time)

Comparisons Bound (for small s)

- Munro, R. (1992)
  $O(2^s n^{1+1/s})$
  $O(n \log \log n)$ randomized algorithm with $O(1)$ space

- R., Ramnath (1998)
  $O(s\, n^{1+1/s} \log n)$

For $s=O(1)$, time is $O(n^{1+\epsilon})$
For $s = O(\log n)$, time is $O(n \log^2 n)$
Recall
For $s= O(n/\log n)$, time is $O(n)$

Chan, SODA 2009

- Frederickson's bd is tight in streaming model
- $\Omega(n \log^*(n/s) + n \log_s n)$ time for det. multi-pass streaming alg'm in comparison model
- $\Omega(n \log \log_s n)$ expected time for RAM in comparison model
- Gave a matching upper bound in the randomized setting (improving on Munro-R 92)

What if input elements are integers from a bounded universe of size U?

Chan, Munro, R. (to appear in ISAAC 2013)

Two (deterministic) algorithms:

1. $O(n \log_s U)$ time,
2. $O(n \log n \log_s \log U)$ time

using $O(s)$ space

Combined to obtain $O(n \log^{1+\varepsilon} n)$ time using $O(1)$ space (of $O(\log U)$ bits each)

$O(n \log_s U)$ algorithm for integers using $O(s)$ space

For i=1 to log U do

By doing a simple count,

find the i^{th} bit of the answer.

$O(n \log U)$ using $O(1)$ space, can be generalized to

$O(n \log_s U)$ using $O(s)$ space by finding log s bits of the answer in each iteration.

O(n log log U) time to find approximate median

- Find the largest prefix p that has > n/2 elements (majority) in the input

- Let x, y, z be such that
  p0x is the smallest element with prefix p0
  p1y is the smallest element with prefix p1
  p1z is the largest element with prefix p1

- p0x, p1y or p1z has rank in [n/4, 3n/4] and hence an approximate median

Finding the prefix

- Do a binary search on the prefix length estimates
- O(log log U) iterations

- Each iteration involves checking for a `majority' element among the elements in the prefix length estimate.

  2-pass O(n) algorithm due to Boyer-Moore

To sum up, O(n log log U) time to find an approximate median.

Can be generalized to $O(n \log_s \log U)$ time (requires finding majority of s prefixes in O(n) time – uses dynamic counting tricks of Dietz)

Finding the median

- Once an approximate median is found, in $O(\log n)$ passes the median can be found

  -- using `filters' to capture the active elements

  (from Munro-Paterson, …)
- Thus $O(n \log n \log_s \log U)$ time to find median using $O(s)$ space or $O(s \log n + \log U)$ bits.

- Choosing $s = \log U / \log n$, and using the min of $O(n \log_s U)$ and $O(n \log n \log_s \log U)$, we get

- $O(n \log^{1+\epsilon} n)$ using $O(1)$ words of $O(\log U)$ bits of space.

New approach (Restore model)

- Elements can be moved around, but after the output, the input needs to be `restored' to original input.
- Relaxed than Read-only memory
- New model of computation between in-place and read-only memory

- Chan, Munro, R (SODA 2014),
  if the inputs are integers,
  can find the median in O(n) time using O(log n) space in restore model.

O(n) selection using O(log n) space in restore model

I O(n log U) time using O(log U) words of O(log n) bits of space

1. Do a quick-sort type partitioning around U/2, except don't move the leading bits.
2. Recurse on the appropriate part (including restore).
3. `Reverse' step 1 to restore the input

(Note: can also sort in O(n log U) time using O(log U) words of O(log n) bits of space)

Reducing the time to O(n)

- Run the previous algorithm as long as each part has at most cn elements (for some constant c).

- When one of the two parts > cn elements, the other one has at most dn = (1-c)n elements. So at that level, at most dn ONEs in the leading bits, so the leading bits can be

  a) extracted to leading part of the array, and

  b) compressed to log (n choose dn) for some d < 1 releasing O(n) bits that can be used to complete the selection using read-only memory alg'm.

Applying these ideas to sorting

- By applying read-only memory sorting algorithm for the tail, can obtain
  $O(n \log n)$ time with $O(\log n)$ space

- By doing s-ary partitioning, can be improved to
  $O(\text{RAM sort})$ time with $O(n^{\epsilon})$ space where RAM sort is
  the time to sort n integers in standard RAM.

Conclusions

- Selection from integers can be done much faster than general input in read-only memory
  ($O(n \log^{1+\epsilon} n)$ time with $O(1)$ space against $O(n^{1+\varepsilon})$ time)

- Selection can be done almost as well as in general RAM in Restore model for integers
  ($O(n)$ time with $O(\log n)$ space)

- Sorting can be done almost as well as in general RAM in Restore model for integers (but is as bad as in ROM if we remove integer assumption)

Open Problems

- $\Omega(n \log \log_s n)$ rand. lower bd for general non-comparison RAM model ??
- $\Omega(n \log^*(n/s))$ or $\Omega(n \log_s n)$ or $\Omega(n^{1+1/s})$ det. lower bd for comparison RAM ROM model ??
- Other problems in Restore model.

Thank You