A Fast Set Intersection Algorithm for Sorted Sequences

Ricardo Baeza-Yates

Center for Web Research

Dept. of Computer Science

Universidad de Chile

rbaeza@dcc.uchile.cl

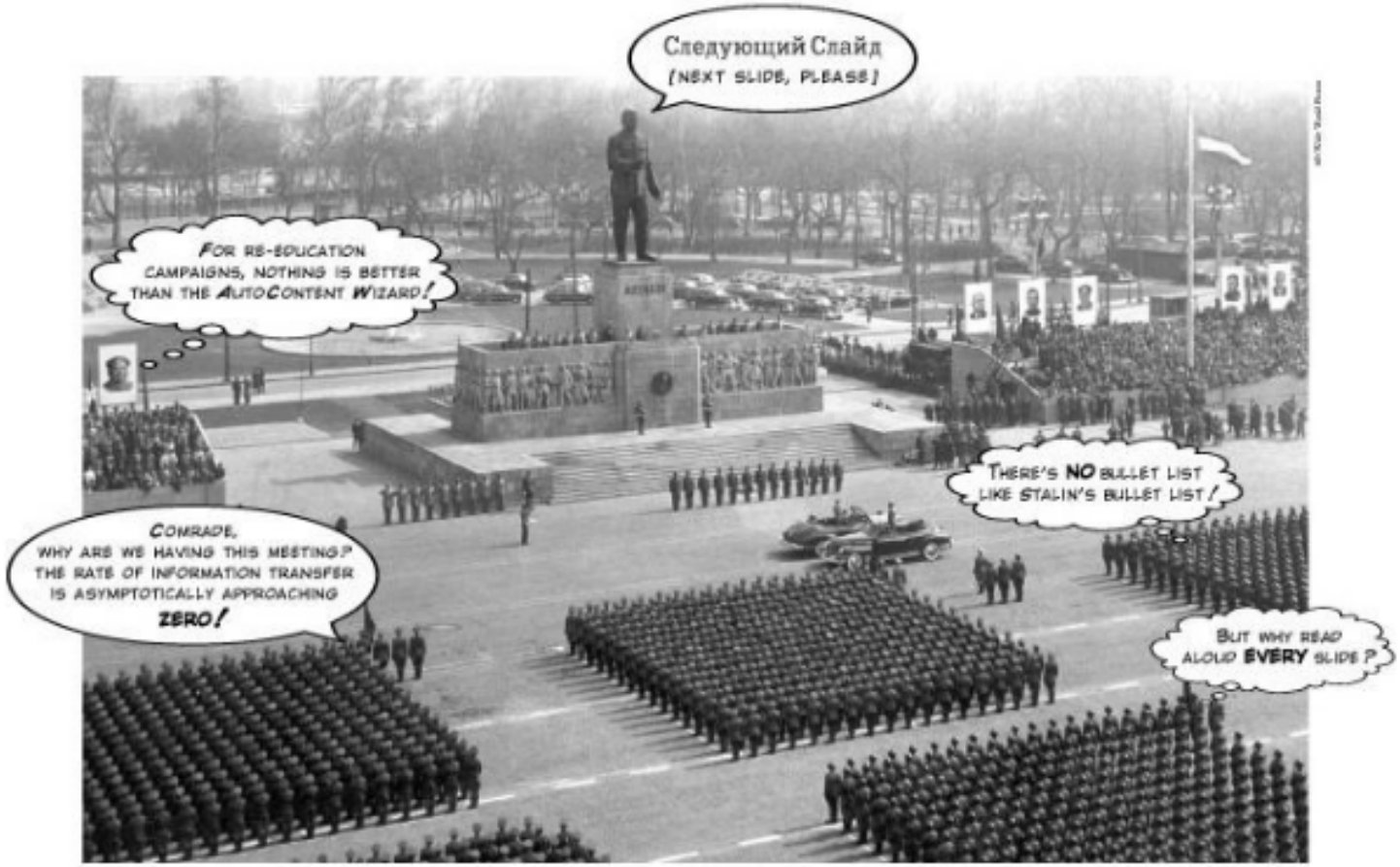
July, 2004

Agenda

- Free Bonus!
- The Problem
- Related Work
- A New Algorithm and Its Analysis
- Experimental Results
- Applications
- Conclusions

The Cognitive Style of PowerPoint

Edward Tufte, 2003



Introduction

Multiple searching:

Given an n-element data multiset, D, drawn from an ordered universe, search D for each element of an m-element query multiset, Q, drawn from the same universe. An algorithm solving the problem must report any elements in both multisets.

The complexity metric is the number of three-way comparisons ($<, =, >$) between any pair of elements, worst case or average case (Rawlins, 1992; Baeza-Yates *et al.*, 1993)

Our problem:

Multiple search when D and Q are **sets** already **ordered**, hence we obtain the intersection of both sets. We use $n = |D|$, $m = |Q|$ and $n > m$.

Related Work

- Lower bounds for the element uniqueness problem and merging (sorted case) do not apply to the multiple search problem (but the converse is true).

- Fernandez de la Vega *et al.* (1998) analyzed the average case of a simplified version of Hwang-Lin's binary merge (1972) finding that if $\alpha = n/m$ with $\alpha > 1$ and not a power of 2, we get

$$\left(r + \frac{1}{1 - \left(\frac{\alpha}{\alpha+1}\right)^{2^r}} \right) \frac{n}{\alpha}$$

  with $r = \lfloor \lg_2 \alpha \rfloor$.

- Adaptive multiple set intersection algorithm (Demaine *et al.*, 2000 & 2001). They also defined the difficulty of a problem instance, which was refined later by Barbay and Kenyon (2002).

Simple Solutions

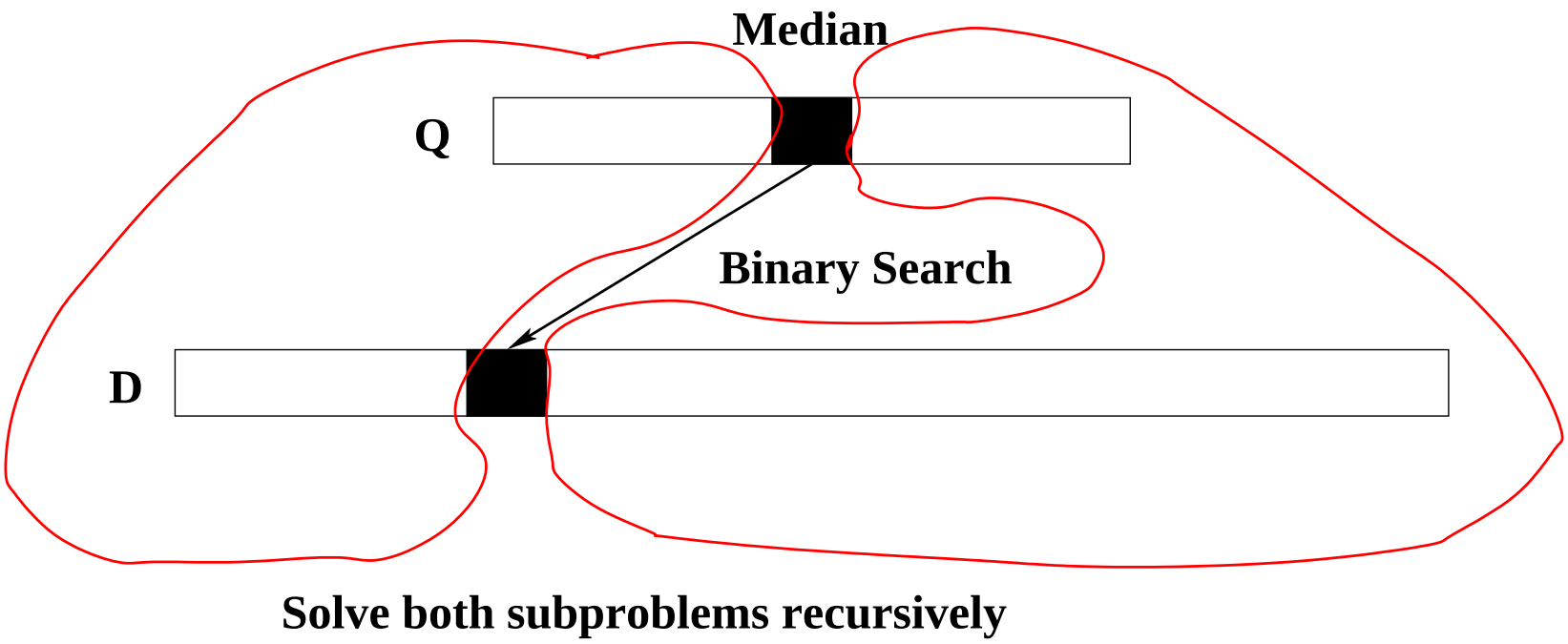
- Merging: $O(n + m)$
- Binary search the smaller set in the larger: $O(m \lg n)$

Can we do it faster?
Perhaps on average?

A Simple but Good Average Case Algorithm

Double Binary Search:

Can be seen as a balanced version of Hwang and Lin's algorithm adapted to set intersection.



At anytime we exchange Q and D if $|Q| > |D|$.

Improvements

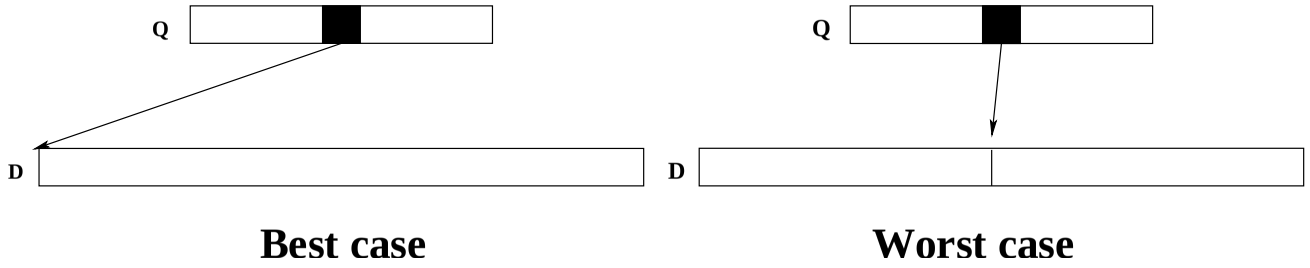
Small improvements in theory:

We compare the smallest elements of both sets with the largest elements in both sets, is $O(1)$.

Otherwise, we search the smallest and largest element of D in Q, to find the real overlap, using just $O(\lg m)$ time.

Doing the opposite is not worth it for small m.

Analysis



Best case: $\lceil \lg(m+1) \rceil \lceil \lg(n+1) \rceil$

If $m = O(n)$ is $O(\lg^2 n)$

Worst case: if $m = 2^k - 1$ we have

$$W(m,n) = \lceil \lg(n+1) \rceil + W((m-1)/2, \lceil n/2 \rceil) + W((m-1)/2, \lfloor n/2 \rfloor)$$

Then

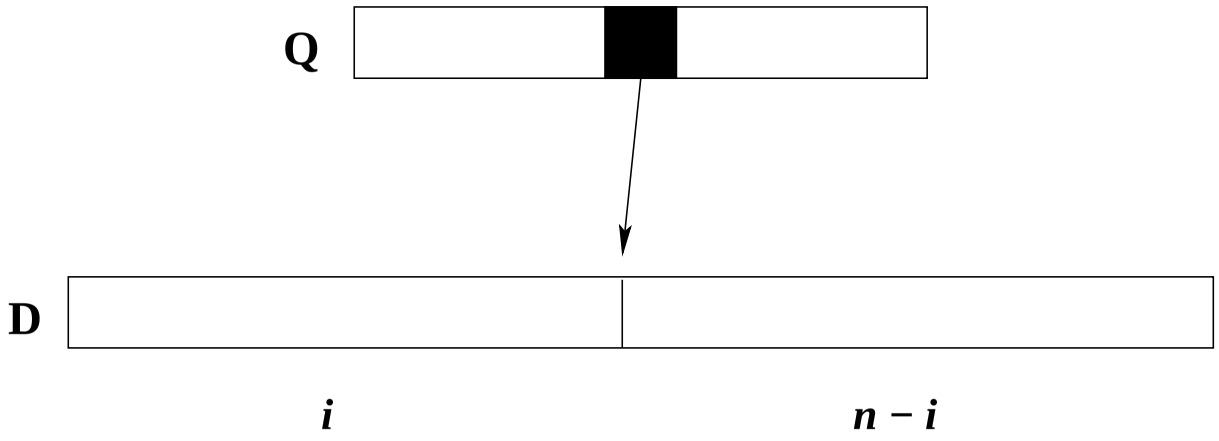
$$W(m,n) = 2(m+1)\lg((n+1)/(m+1)) + 2m + O(\lg n)$$

If $n = \alpha m$ it is $O(n)$ and the ratio between this algorithm and merging is $2(1+\lg(\alpha))/(1+\alpha)$.

Average Case Analysis (1)

We use two pessimistic assumptions:

- We never find the median of Q in D
- The median will divide D in sets of size i and $n - i$ with the same probability for all i



Average Case Analysis (2)

If $A(m,n)$ denotes the average case number of comparisons, for m of the form $2^k - 1$ we have

$$A(m,n) = \lceil \lg(n+1) \rceil + \frac{1}{n+1} \sum_{i=0}^{n} (A((m-1)/2, i) + A((m-1)/2, n-i))$$

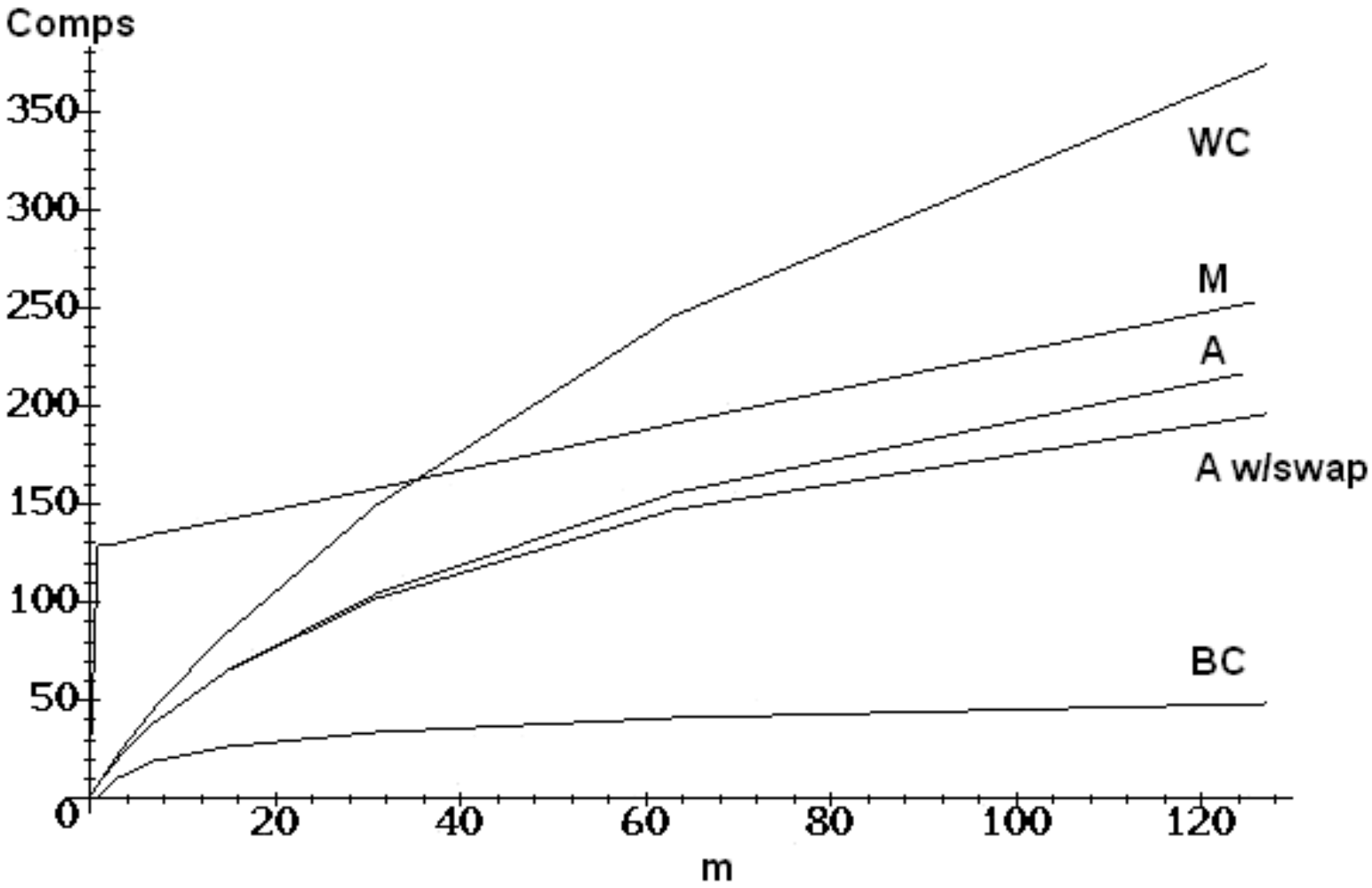
We show that

$$A(m,n) = (m+1)(\ln((n+1)/(m+1)) + 3 - 1/\ln(2)) + O(\lg n)$$

by using generating functions in two variables and the Maple symbolic algebra system.

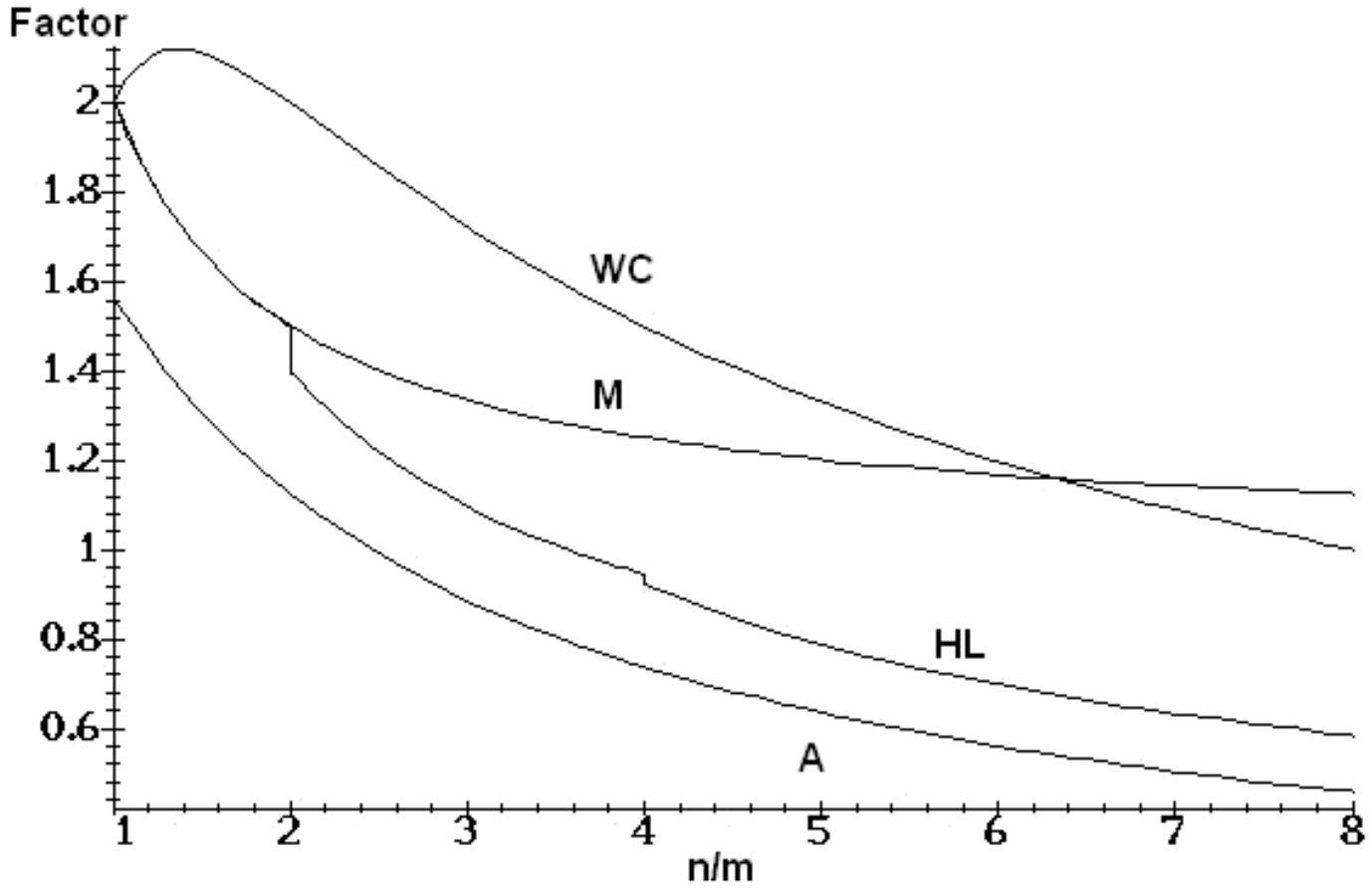
For $n = \alpha m$, the ratio between this algorithm and merging is $(\ln(\alpha) + 3 - 1/\ln(2))/(1+\alpha)$.

For $n = 128$ and all powers of 2 for $m \leq n$:



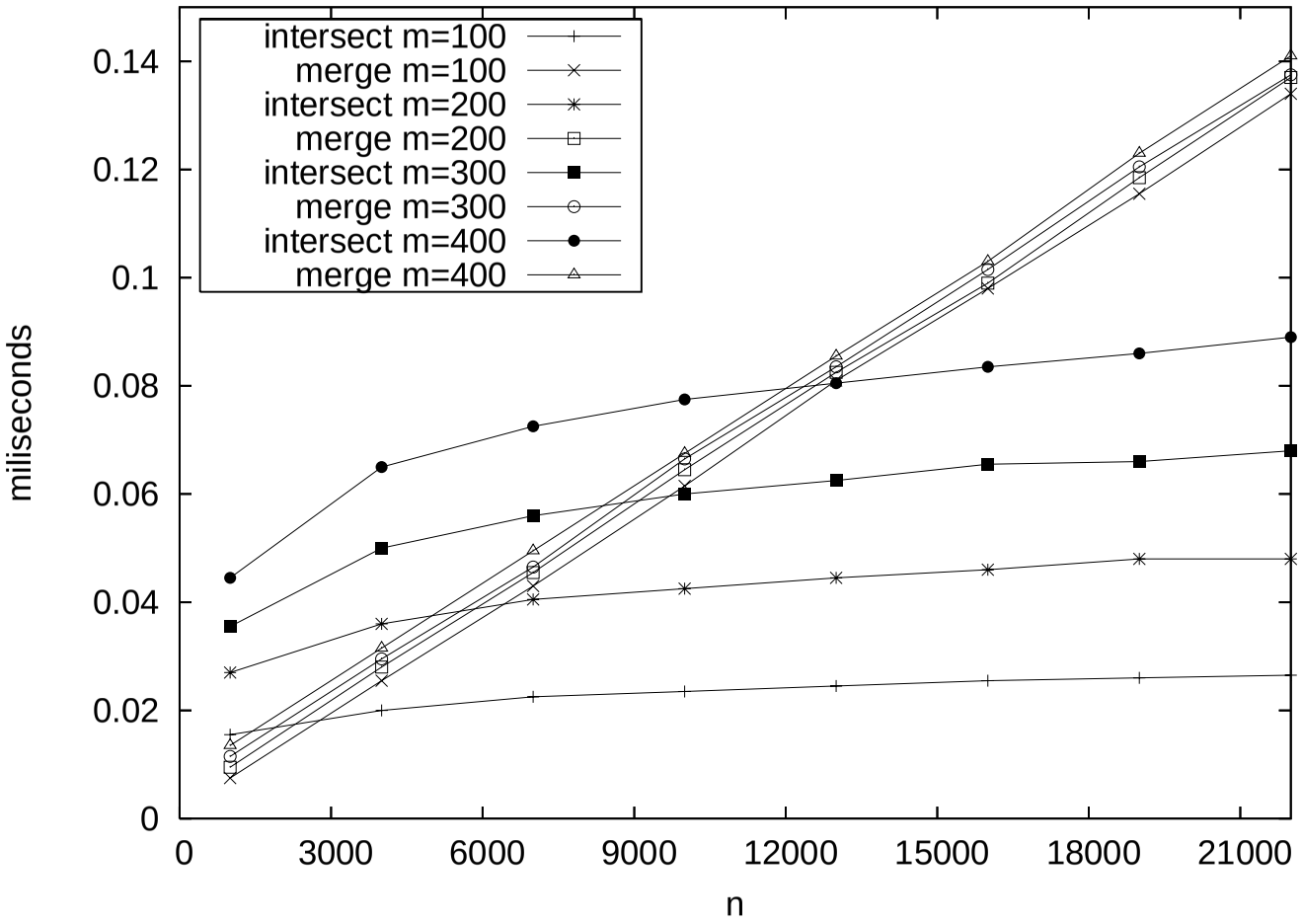
WC is worse then M for $1 < \alpha < 6.3197$ having its maximum at $\alpha = 2.1596$ where it is 1.336 times slower than M.

AC is at most 0.7913 times M when $\alpha = 1.2637$ and 0.7787 when $\alpha = 1$.



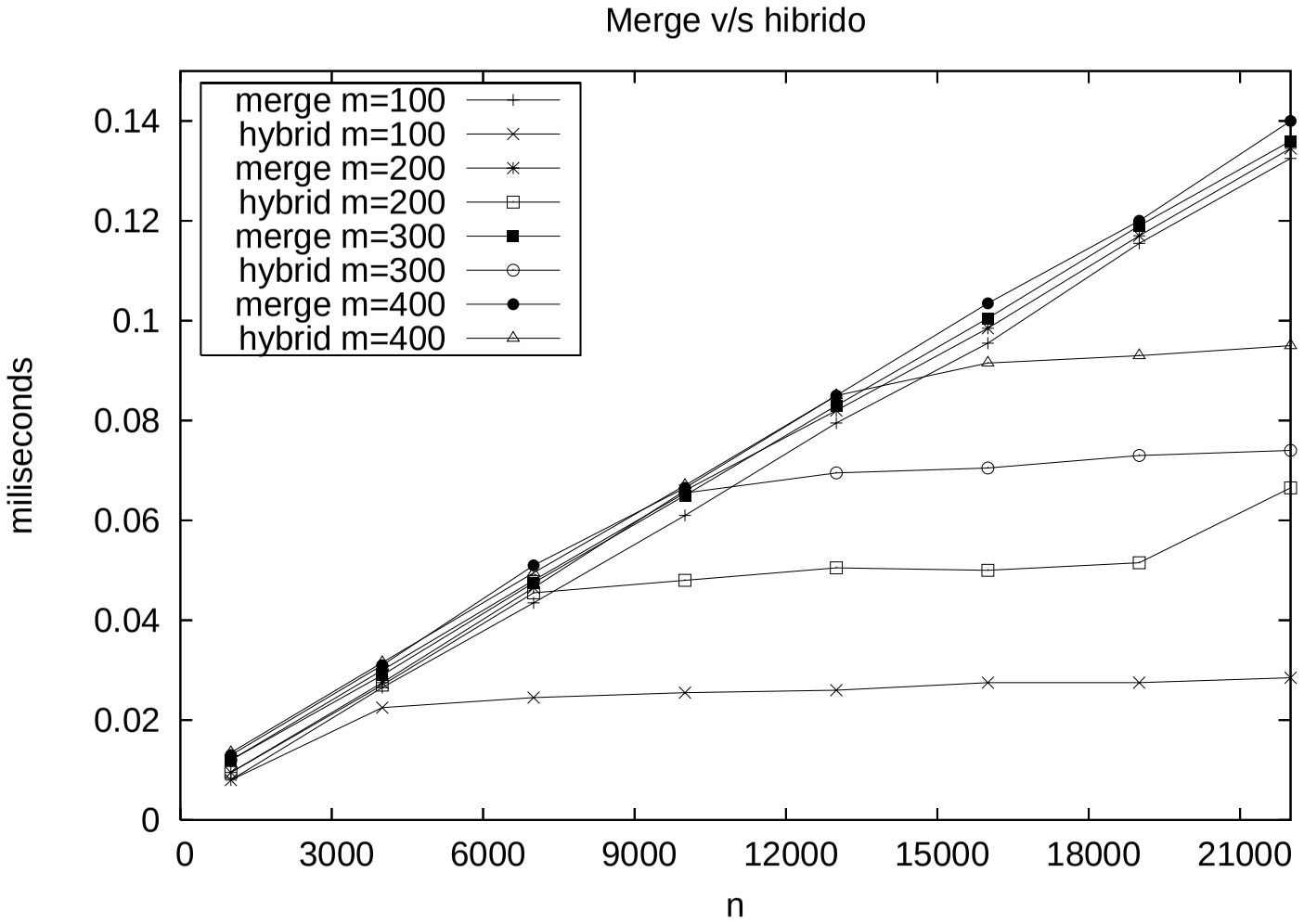
Experimental Results

We use uniformly random integer numbers in the range 1 to 10^9 and we implemented the algorithms using the Gcc 3.3.3 compiler in a Linux platform running on an Intel Xeon of 3GHz.



Hybrid Algorithm

When merge is better, stop the recursion. We found the empirical line when this happens in our implementation.



Query Processing in Inverted Indexes

Inverted indexes are a vocabulary and a list of references per word to its occurrences in documents and optionally to where they occur per document (full inversion)

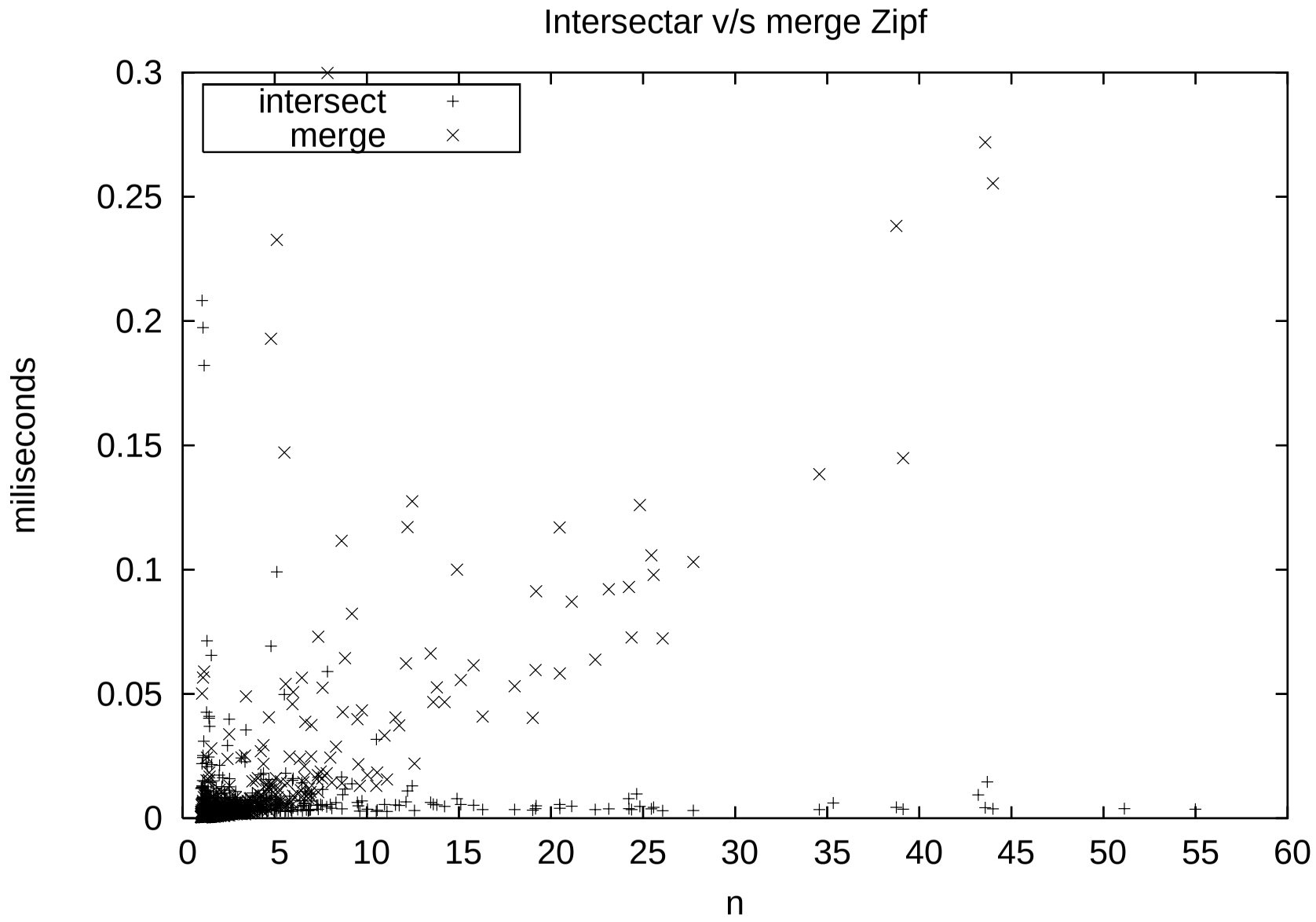
Lists usually use linked blocks of references (variable size)

The references are sorted in:

- Boolean model: intersection is basic for set operations
- Vectorial model: by chunks of doc-ids with similar term-frequency
- Word positions per document: shifted intersection
  (full inversion, needed for phrase search)
- Proximity search: that is $i \pm k$

Web Issues

- Global ranking schemes (e.g. PageRank): document ids are defined by the ranking
- The distribution of word occurrences is a power law, and the same is true with query frequencies.
- However the correlation is almost null.
- That means that the average lengths of the lists, n and m, when sampled, will satisfy $n = \Theta(m)$ (uniform), rather than $n = m + O(1)$ (power law).


Intersectar v/s merge Zipf
intersect
merge
miliseconds
n
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
35
40
45
50
55
60

Concluding Remarks

- In practice, Web queries are short: 1 to 3 words
- Hence, there is almost no need to do multiset intersection and if so, they can be easily handled by pairing the smaller sets firsts
- We need partial evaluation, as most people only look at less than two result pages
- Adaptive algorithm that finds first the first answers and in the right order?

  That implies first to do the recursion on the left side

Thanks to Alejandro Salinger, Phil Bradford, Joe Culberson, and Greg Rawlins.