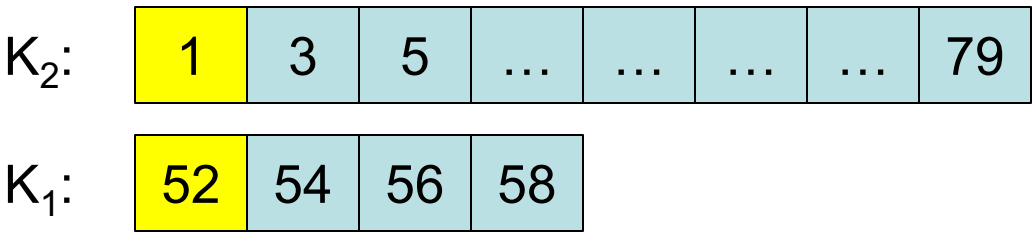
Can you use a hashtable to implement skipTo()?

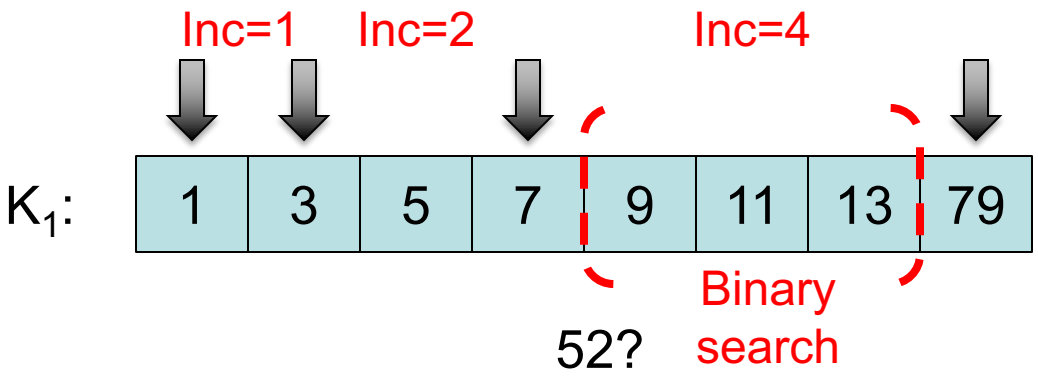
Better than next()

- What's the worst case for sequential merge-based intersection?
- {52, 1} ➔ Image K_2's cursor
  - To the positi
  - Essentially, **skipTo**(52) $K_2[i] >= 52$ (K2's list is sorted).
  - Takes many sequential call of
  - Could use binary search in the rest of the list
  - Cost: $\lceil \log_2(N_{remainder}) \rceil$



skipTo(id)

- Galloping search (gambler's strategy)
  - [Stage 1] Doubling the search range until you overshoot
  - [Stage 2] Perform binary search in the last range
- Performance analysis (worst case)
  - Let the desti s away.
  - ≈ $\log_2 n$ probes in Stage 1 + ≈ s in Stage 2
  - Total = $2\lceil \log_2 (n+1) \rceil = O(\log_2$



Total Cost

- Galloping search (gambler's strategy)
  - Cost of the i-th probe: ≈ $2 \log_2(n_i)$
  - Total cost of K_1 probes: ≈ $2 \log_2(\prod_{i=1}^{|K_1|} n_i)$

  ≤ $2 \log_2(((\sum$ $\log_2(|K_2|/|K_1|)$
- Asymptotic r merge when $|K_2|/|K_1| = O(1)$, resemb search when $|K_1| = O(1)$

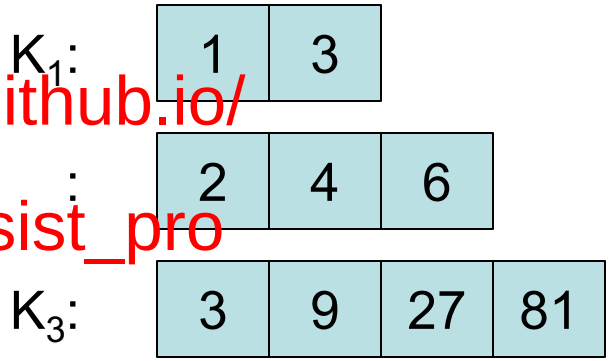
What about list intersection using binary search?

Multiple Term Conjunctive Queries

- K_1 AND K_2 AND … AND K_n
- SvS does not perform well if none of the associate
- In addition, it is block
- Can you design non-blocking multiple sorted array intersection algorithm?

Generalization

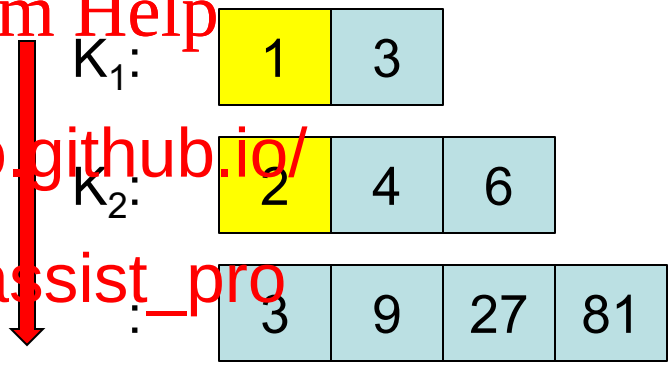
- Generalize the 2-way intersection algorithm
- 2-way:
  - {1, 2} ➔ move k_1's cursor
  - skipTo(2)
- 3-way:
  - {1, 2, 3} ➔ move k_1,k_2's cursor
  - skipTo(3)



eliminator = $Max_{1<=i<=n}(k_i.cursor)$

Optimization

- Mismatch found even before accessing K_3's cursor
- Choice 1: c
  cursors of other list
- Choice 2: settle the
  dispute within the first two lists ➔ max algorithm [Culpepper & Moffat, 2010]
  - Better locality of access ➔ fewer cache misses
  - Similar to SvS



Pseudo-Code for the Max Algorithm (Wrong)

- Input: $K_1, K_2, \ldots, K_n$ in increasing size

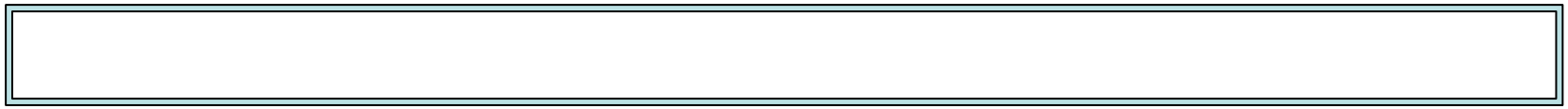
```
(1)   x := K1[1]; startAt := 2          //x is the eliminator
(2)   while x is defined do
(3)       for i = startAt to n do
(4)           y := Ki.skipTo(x)
(5)           if y > x th
(6)               x := K1                          //res          //restart_2
(7)               if y > x then startAt := i else startAt := 2 end if
(8)               break                            //match in all  lists
(9)           elsif i = n then                     //y = x
(10)              Output x
(11)              x := K1.next()
(12)          end if
(13)      end for
(14)  end while
```

A

B

C

D



The original code has a bug when in restart_1 cases

Pseudo-Code for the Max Algorithm (Fixed)

- Input: $K_1, K_2, \ldots, K_n$ in increasing size

```
(1)   x := K1[1]; startAt := 2
(2)   while x is defined do
(3)       for i = startAt to n do
(4)           y := Ki.skipTo(x)
(5)           if y > x th
(6)               x := K1
(7)               if y > x then startAt :=
(8)               break
(9)           elsif i = n then
(10)              Output x
(11)              x := K1.next()
(12)          end if
(13)      end for
(14)  end while
```

```
(4.1) if i = 1 then
(4.2)     if y > x then
(4.3)         x := y
(4.4)         break
              if
(4
                  startAt := 2 end if
```

References

- J. Shane Culpepper, Alistair Moffat: Efficient set intersection for inverted indexing. ACM Trans. Inf.
- F.K. Hwan e algorithm for merging two disjoint lin red sets. SIAM J. Comput. 1 1 (1972), pp. 31–39.
- Stefan Buettcher, Charles L. A. Clarke, Gordon V. Cormack, Information Retrieval: Implementing and Evaluating Search Engines, 2010 [Chapter 5]