Information Retrieval

WS 2012 / 2013

Lecture 3, Wednesday November 7th, 2012

(List intersection: fancy algorithms + lower bounds)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

UNI FREIBURG

- Organizational
  - Your experiences with Exercise Sheet 2 (Ranking)
- List intersection
  - Key algorithm in every search engine
  - Some algorithm engineering tips
  - Fancier algorithms + runtime analysis
  - Matching lower bound
  - **Exercise Sheet 3:**

    Implement the asymptotically optimal algorithm

    Is it really faster than the simple linear-time algorithm?

    Some optional theoretical tasks, good for exam preparation

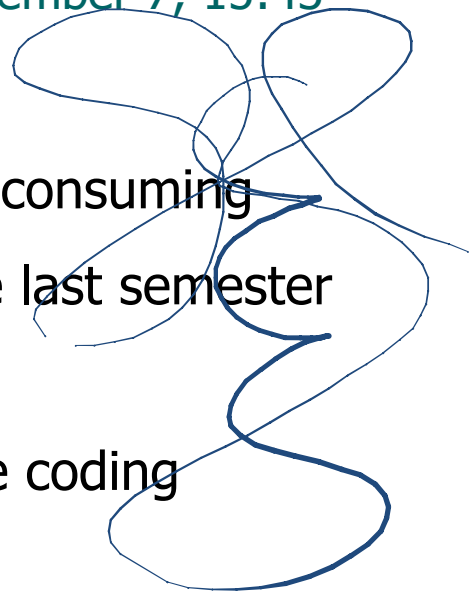
Experiences with ES2 (ranking)

UNI FREIBURG

- Summary / excerpts last checked November 7, 15:43
  - Feasible in reasonable time for most
  - Manual relevance assessment is quite time-consuming
  - Wiki table for comparing results please, like last semester

    Ok, we will have one again for this exercise !
  - Some of you are quite passionate about the coding

    Great, I hope you become addicted to it !
  - Lecture pen too thick / too many colors ... let's see today
  - How to write unit tests ... I'll live code an example today
  - More coding, less theory please ... ok, ok
  - Great recordings / streamed from iPhone to TV ... yeah!



Your results for ES2 (ranking)

UNI FREIBURG

- Some interesting observations
  - Binary (k = 0, b = 0) was clearly the worst
  - In the tf.idf list, some linguists and other theory guys

    Many occurrences of **either** "relativity" **or** "theory" can boost the score
  - Albert Einstein was top for tf.idf, but not even in the top-10 for BM25 with standard settings

    BM25 favors short articles that are mainly about rel. theo.

    To get AE at the top, a global document score (like the PageRank for web pages) would be more appropriate ... later lecture

List Intersection 1/5

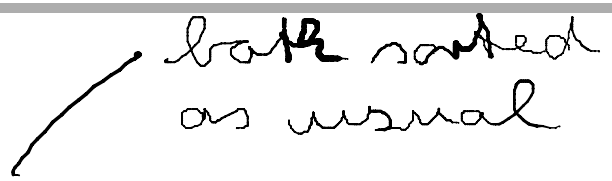
UNI FREIBURG

- Let's first revisit the linear-time algorithm ...
  - ... and see what we can improve there
  - Java: ArrayList<Integer> much worse than native int[]
  - C++: vector<int> is as good as int[] with option –O3
  - Minimize the number of branches within small-bodied loops
  - **Note 1:** hard to predict / understand some of the effects

    To understand what is really going on, look at the machine code (in C++) or byte code (Java) ... see seminar Java vs. C++ on Wiki
  - **Note 2:** when comparing algorithms, always repeat each run at least 3 times, to make caching effects transparent

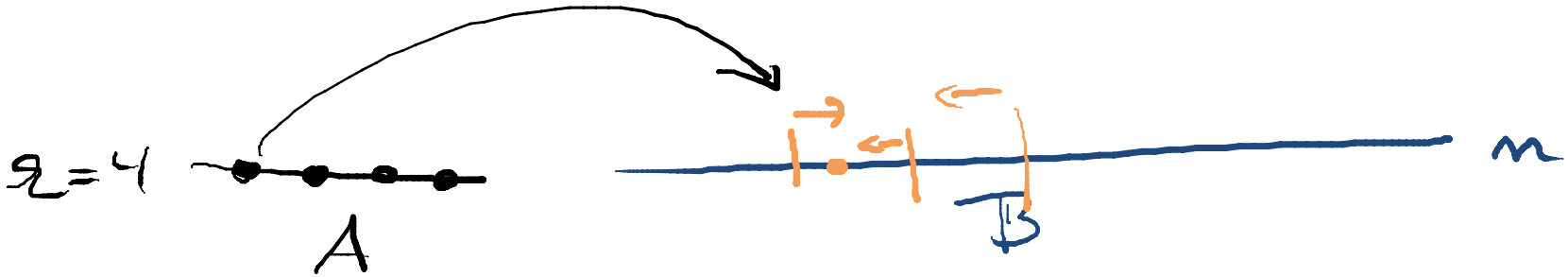
List Intersection 2/5

- Algorithmic improvement 1



  - Call the smaller list A, and the longer list B
  - Search the elements from A in B, using binary search

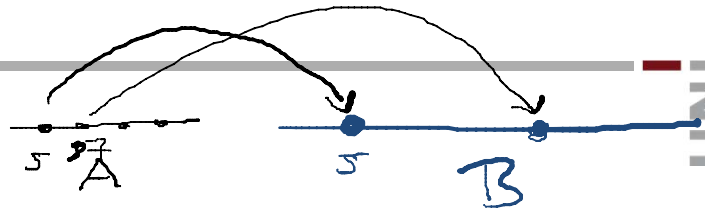
    k = #elements in A, n = #elements in B
  - This has time complexity $\Theta(k \cdot \log n)$



Bad: not so good if $k \approx n$

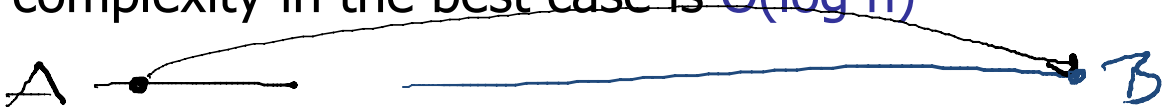
then $n \cdot \log n$

List Intersection 3/5



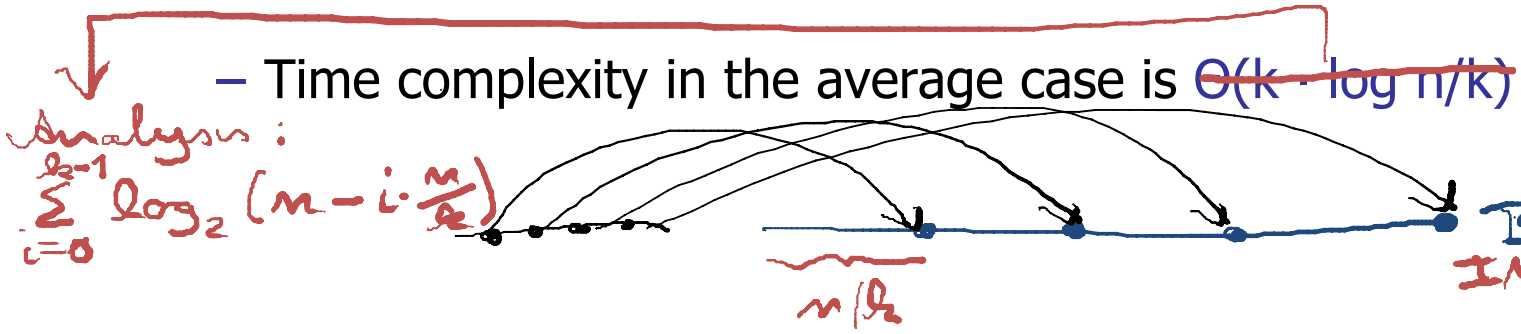
- Algorithmic improvement 2
  - **Observation:** when we have located element A[i] in B, that is, we have found j with B[j-1] < A[i] ≤ B[j] ...

    ... then for all elements A[i'] with i' > i, it suffices to look at elements B[j'] with j' ≥ j
  - Time complexity in the best case is Θ(log n)



  - Time complexity in the worst case is Θ(k · log n)

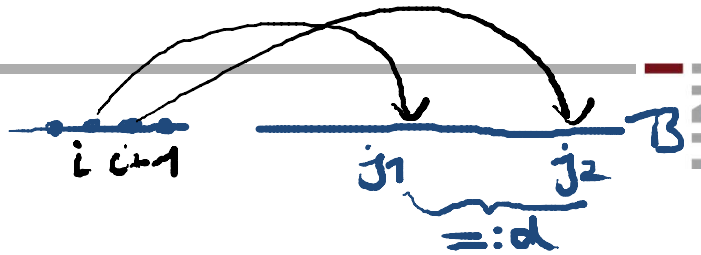
  - Time complexity in the average case is ~~Θ(k · log n/k)~~



for the first n/2 elements from A the cost is $\geq \log_2 \frac{n}{2} = \log_2 n - 1$

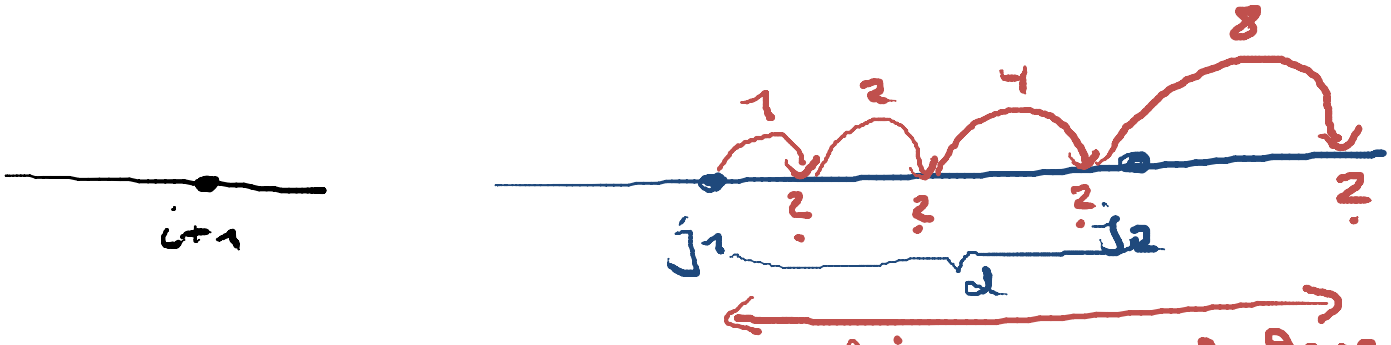
SO: NO REAL IMPROVEMENT

List Intersection 4/5



- Algorithmic improvement 3
  - **Goal:** when elements A[i] and A[i+1] are located at postions j_1 and j_2 in B, then, with $d := j_2 - j_1$ ("gap") ...

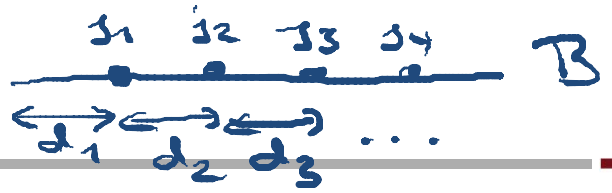
    ... spend only time $\Theta(\log d)$ to locate element A[i+1]
  - **Idea:** first do an exponential search, to get an upper bound on the range, then a binary search as before



Range of the binary search will be at most $2d$ ($\log 2d = \log d + 1$)

Exp. search $1 + 2 + \cdots + 2^i \leq 2d \Rightarrow i \leq \log 2d$

List Intersection 5/5

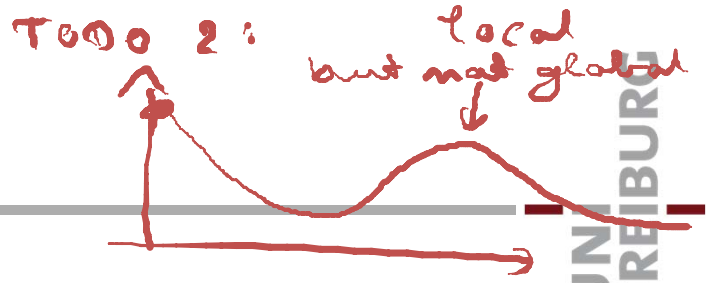


- Time complexity of this algorithm
  - Let $d_1, \ldots, d_k$ be the gaps between the locations of the k elements of A in B

    d_1 = from beginning to first location
  - Note that $\Sigma_i d_i \leq n$ = the number of elements in B
  - Then the time complexity is $O(\Sigma_i \log d_i)$
  - **Goal:** find a formula that is independent of the d_i
  - **Idea:** maximize $\Sigma_i \log d_i$ under the constraint $\Sigma_i d_i \leq n$
  - This is called optimization with side constraints or Lagrangian optimization ... next slide

RESULT $\sum_{i=1}^{k} \log d_i \leq \sum_{i=1}^{k} \log \frac{n}{k} = k \cdot \log \frac{n}{k}$

Lagrangian Optimization



- By example of the task from the previous slide
  (there is an optional theoretical exercise, where you can practice the technique for yourself)

$d_j = n$
$d_i = 0 \quad i \neq j$

$$\max \sum_{i=1}^{k} \ln d_i \quad \text{s.t.} \quad \sum_{i=1}^{k} d_i \leq n$$

$$\mathcal{L} = \sum_{i=1}^{k} \ln d_i + \lambda \left(\sum_{i=1}^{k} d_i - n \right)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \sum_{i=1}^{k} d_i - n \stackrel{!}{=} 0 \quad \Rightarrow \quad \sum_{i=1}^{k} d_i = n$$

$$\forall j \quad \frac{\partial \mathcal{L}}{\partial d_j} = \frac{1}{d_j} + \lambda \stackrel{!}{=} 0 \quad \Rightarrow \quad d_j = -\frac{1}{\lambda}$$

$\Rightarrow$ all d_j are equal and sum to n

$\Rightarrow \forall j \quad d_j = \frac{n}{k}$

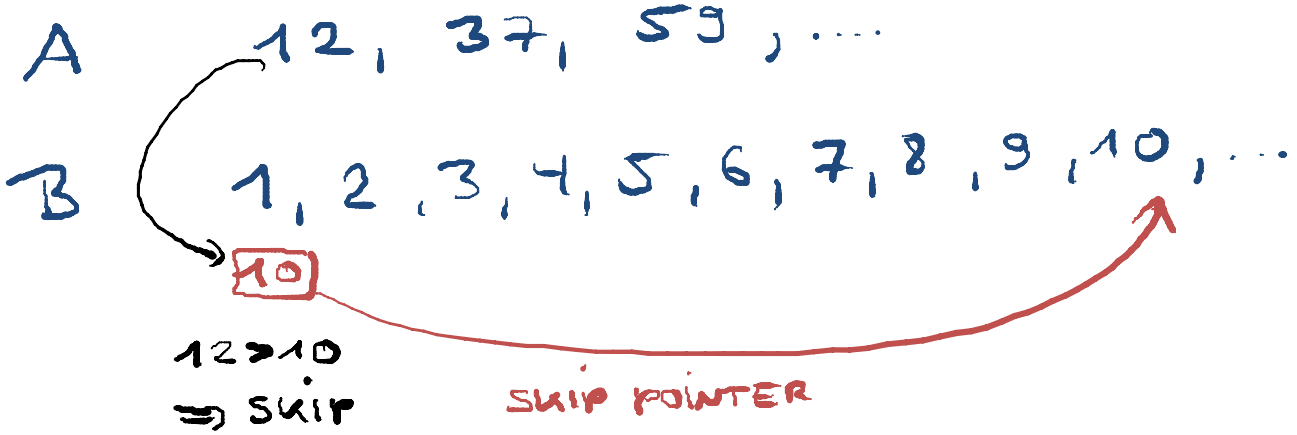
TODO 1: $\frac{\partial^2 \mathcal{L}}{\partial^2 d_j} < 0$

TODO 1: Check whether MIN or MAX

TODO 2: CHECK BORDERS

Skip Pointers

- A heuristic approach to increase performance
  - **Idea:** place pointers at "strategic" positions in the list, which allow to "skip" large parts during intersection
  - **Question:** where to place how many pointers?



Lower bounds 1/5

UNI FREIBURG

- Let's first look at both union and intersection
  - For list **union**, the simple linear-time algorithm has a time complexity of $\Theta(k + n)$

    This is optimal, because the result of the union contains $k + n$ elements, and every algorithm has to output them
  - For list **intersection**, the exponential-binary-search algorithm has a time complexity of $\Theta(k \cdot \log (n/k))$

    Exercise (optional): prove that $k \cdot \log (n/k)$ never worse than $n + k$ (and usually much better, e.g. for $k = \text{const}$)

    But maybe we can do even better?

    (Note: the output size is at most k for intersection)

Lower bounds 2/5

UNI FREIBURG

- Recall: lower bound for comparison-based sorting
  - There are $n!$ possible outputs
  - The algorithm has to distinguish between all of them
  - Each comparison distinguishes between two cases
  - Hence we need at least $\log_2 n!$ comparisons
  - By Stirling's formula $(n/e)^n \leq n! \leq n^n$
  - Hence $\log_2 n! \sim n \cdot \log n$
  - Hence every comparison-based sorting algorithm has a running time of $\Omega(n \cdot \log n)$
  - Note: not true for non-comparison based algorithms

    For example, 0-1 sequences can be sorted by counting

Lower bounds 3/5

UNI FREIBURG

- Similar argument for intersecting two lists A and B
  - As before, k = #elements in A, n = #elements in B
  - How many different ways are there to locate the k elements from A within the n elements from B
  - **Observation:** each such way corresponds to a tuple $(j_1,\ldots, j_k)$ where $0 \le j_1 \le \ldots \le j_k \le n$

    j_i is simply the location of the i-th element of A in B
    location 0 means before the first element
    location $i > 0$ means after the i-th element

    How many such tuples are there?

    THE ANSWER: $\binom{n+k}{k} \ge \left(\frac{n}{k}\right)^k$

    $\log\left(\frac{n}{k}\right)^k = k \cdot \log\frac{n}{k}$

Lower bounds 4/5

- There is a similar quantity which is easy to count
  - The number of tuples $(i_1,\ldots, i_k)$ where $1 \le i_1 < \ldots < i_k \le m$
  - This is just the number of size-k subsets of $\{1,\ldots,m\}$
  - Their number is just m over $k = m! / (k! \cdot (m - k)!) \ge (m/k)^k$
  - Let us relate this to the number we are interested in:

    The number of tuples $(i_1,\ldots, i_k)$ where $0 \le i_1 \le \ldots \le i_k \le n$

Lower bounds 5/5

UNI FREIBURG

- Now it's easy to derive the lower bound
  - By the previous arguments, the number of ways to locate the k elements from A in the n element of B is

    m over $k \geq (m/k)^k$, where $m = n + k$
  - With the same argument as in the sorting lower bound, any comparison-based algorithm hence has to make

    $\log_2 (m \text{ over } k) \geq \log_2 (m/k)^k$ comparisons
  - This is $k \cdot \log_2(1+n/k)$ and hence $\Omega(k \cdot \log_2(n/k))$

References

UNI FREIBURG

- In the Raghavan/Manning/Schütze textbook

  Section 2.3: Faster intersection with skip pointers

- Relevant Papers

  A simple algorithm for merging two linearly ordered sets
  F.K. Hwang and S. Lin SICOMP 1(1):31–39, 1980

  A fast set intersection algorithm for sorted sequences
  R. Baeza-Yates CPM, LNCS 3109, 31–39, 2004

- Relevant Wikipedia articles

  http://en.wikipedia.org/wiki/Lagrange_multiplier