

Outline

1. List Ranking
2. Sorting
3. Pattern Matching
We choose some non-consecutive nodes and we splice them out.
We keep track of the number of nodes we remove.
And then rebuild the list and obtain the ranking.
We choose some non-consecutive nodes and we splice them out.

We keep track of the number of nodes we remove.

And then rebuild the list and obtain the ranking.
We choose some non-consecutive nodes and we splice them out.
We keep track of the number of nodes we remove.
And then rebuild the list and obtain the ranking.
We choose some non-consecutive nodes and we splice them out. We keep track of the number of nodes we remove and then rebuild the list and obtain the ranking.
We choose some **non-consecutive** nodes and we splice them out.

We keep track of the number of nodes we remove.

And then **rebuild** the list and obtain the ranking.
We choose some non-consecutive nodes and we splice them out
We keep track of the number of nodes we remove
And then rebuild the list and obtain the ranking
We choose some non-consecutive nodes and we splice them out.
We keep track of the number of nodes we remove.
And then rebuild the list and obtain the ranking.
Randomized List-Ranking
By Anderson and Miller, 1990

Work-optimal algorithm
- It works with $\frac{n}{\log n}$ processors
- It splices out $\Omega \left(\frac{n}{\log n}\right)$ nodes at each step
- Hence the total running time is $O(\log n)$

How it works
- Each processor takes care of a queue of $\log n$ nodes, consecutive in memory
- It starts by trying to splice out the head of the queue, and proceeds until no more nodes are left in its queue
- Then the nodes are reinserted into the list, until the rank is totally computed
A problem arises when the heads of the queues happen to be consecutive. We must choose which nodes to splice out and which to leave untouched.

Randomized choice

- We randomly assign 0 or 1 (with probability \( p = \frac{1}{2} \)) to each node we would like to splice out.
- We can splice out a node only if it has value 1 and is followed by a 0-valued node (or by a non-head-of-queue node).
- In this way, no consecutive nodes can be spliced out.
Randomized List-Ranking
Handling Consecutive Nodes

- A problem arises when the heads of the queues happen to be consecutive
- We must choose which nodes to splice out and which to leave untouched

Randomized choice

- We *randomly* assign 0 or 1 (with probability $p = \frac{1}{2}$) to each node we would like to splice out
- We can splice out a node only if it has *value 1* and is followed by a *0-valued* node (or by a non-head-of-queue node)
- In this way, no consecutive nodes can be spliced out

```
<table>
<thead>
<tr>
<th>X</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
A problem arises when the heads of the queues happen to be consecutive. We must choose which nodes to splice out and which to leave untouched.

Randomized choice

- We randomly assign 0 or 1 (with probability $p = \frac{1}{2}$) to each node we would like to splice out.
- We can splice out a node only if it has value 1 and is followed by a 0-valued node (or by a non-head-of-queue node).
- In this way, no consecutive nodes can be spliced out.
Consider the following r.v.'s

- $X_{i,t}$: 1 if processor $i$ splices out a node at time $t$, 0 otherwise.
- $Y_{i,t}$: Total nodes spliced out by processor $i$ up to time $t$.

\[ Y_{i,t} := \sum_{\tau \leq t} X_{i,\tau} \]

- $B_t$: Mutually independent Bernoulli r.v.'s with parameter $p = \frac{1}{4}$.
- $Z_t$: Binomial associated to $B_t$'s:

\[ Z_t := \sum_{\tau \leq t} B_{\tau} \]

∀$t$, a given processor $i$ splices out a node with probability $\geq \frac{1}{4}$:

\[ \Pr(X_{i,t} = 1) \geq \Pr(B_t = 1) = \frac{1}{4} \]

This implies

\[ \Pr(Y_{i,t} \geq k) \geq \Pr(Z_t \geq k) \quad \Pr(Y_{i,t} \leq k) \leq \Pr(Z_t \leq k) \]
• On average, a processor finishes its list in time upper bounded by $4 \log(n)$

• The probability that at time $t$ a processor still has to finish is

$$\Pr(Y_{i, t} < \log(n)) \leq \Pr(Z_t < \log(n)) \leq \Pr(Z_t \leq \log(n))$$

• If we let $t = 20 \log(n) \Rightarrow \mu = 5 \log(n)$ and apply the Chernoff bound we obtain

$$\Pr(Z_t \leq \log(n)) = \Pr\left(\mu - Z_t \geq \mu \frac{4}{5}\right) \leq \exp\left(\frac{-16\mu}{25 \cdot 3}\right)$$

$$< \exp\left(\frac{-\mu}{5}\right) = \exp(-\log(n)) = \frac{1}{n}$$
The probability that processor $i$ has not finished its nodes at time $t = 20 \log(n)$ (event $F_i$) is less than $\frac{1}{n}$.

The probability $Q(n)$ that there is at least one processor which has not finished at time $t = 20 \log(n)$ is

$$Q(n) = \Pr \left( \bigcup_{i=1}^{n} F_i \right) \leq \sum_{i=1}^{n} \Pr (F_i) \leq \frac{1}{\log(n)}$$

Therefore the expected runtime $T(n)$ satisfies the following inequality:

$$T(n) \leq \frac{\log(n) - 1}{\log(n)} 20 \log(n) + \frac{1}{\log(n)} (20 \log(n) + T(n))$$

$$\Rightarrow \quad T(n) \in O(\log(n))$$
Outline

1. List Ranking
2. Sorting
3. Pattern Matching
Let \( a \) be a vector of \( n \) elements to be sorted:

\[
a[1], a[2], \ldots, a[n]
\]

- We have \( p \) processors
- The number of processors is constrained:

\[
p \leq \sqrt{\frac{n}{2 \ln n}} \quad \Rightarrow \quad p \leq \frac{n}{p}
\]

- For simplicity, we assume that \( p \) divides \( n \)
- We also assume all the \( n \) elements to be distinct (just to simplify the analysis)
- We’ll give the algorithm on a PRAM, but the original version is for distributed memory machines
- It can be easily implemented on both shared and distributed memory systems
We choose \( p + 1 \) splitters:

\[-\infty = S_0 < S_1 < \cdots < S_i < S_{i+1} < \cdots < S_p = +\infty\]

Elements are divided into \( p \) buckets \( B_1, B_2, \ldots, B_p \), such that

\[x = a[k] \in B_i \iff S_{i-1} < x \leq S_i\]

The buckets are then sorted independently in parallel (a bucket for each processor)

We want the buckets to contain almost the same amount of data

Hence we need a smart way to choose the splitters (we’ll do it by sampling)
The Algorithm
Part 1/2

1. Each processor takes care of \( \frac{n}{p} \) elements, consecutive in memory.

2. Each element \( a[k] \) is given a (uniformly) random color \( c[k] \in \{1, \ldots, p\} \).
   Time: \( O\left(\frac{n}{p}\right) \)

3. A colored compactification is performed
   Time: \( O(p + \log n) \subset O\left(\frac{n}{p} + \log n\right) \)

4. Processor \( i \) takes care of the \( C_i \) elements with color \( i \). Each processor sequentially sorts its data independently in parallel.
   Time: \( O(\max_i C_i \log C_i) \)

5. Processor 1 outputs its \( p \)-quantiles as splitters

   \[ \forall i: 0 < i < p \quad S_i := \frac{iC_1}{p} \text{-th element (with color 1)} \]

   Time: \( O(p) \)
Each processor uses **binary search** within its own color to (sequentially) **partition** its elements into **p segments** defined by the splitters.

Time: $O(\max_i p \log C_i)$

Each $i$-th local bucket is labeled with color $i$

Time: $O(\max_i C_i)$

A **colored compactification** is performed

Time: $O(p + \log n) \subset O\left(\frac{n}{p} + \log n\right)$

Processor $i$ takes care of the $D_i$ elements with color $i$ (i.e., bucket $B_i$). Each processor **sequentially sorts its data independently** in parallel.

Time: $O(\max_i D_i \log D_i)$
Time Complexity

- If $\max_i C_i \in O\left(\frac{n}{p}\right)$ and $\max_i D_i \in O\left(\frac{n}{p}\right)$, then the total runtime is

$$T(n) \in O\left(\frac{n}{p} \log \left(\frac{n}{p}\right)\right) \subset O\left(\frac{n}{p} \log n\right)$$

i.e., the algorithm is work-optimal

- We want to prove that, given the initial random choice of colors, the above property holds with high probability, i.e.,

$$\exists \epsilon > 0 \text{ s.t. } \Pr(\text{execution is fast}) \geq 1 - n^{-\epsilon}$$
We now want to prove that $\max_i C_i \in O \left( \frac{n}{p} \right)$. In more detail

**Lemma 1**

With high probability

$$\forall p < \sqrt{\frac{n}{2 \ln n}} < \frac{n}{3 \ln n} \quad \max_i C_i < \nu := 2 \frac{n}{p}$$

The number of items tagged with some color $i$ is given by a binomial distribution, with parameter $q := \frac{1}{p}$, i.e.,

$$\Pr (C_i = k) = \binom{n}{k} q^k (1 - q)^{n-k} \quad E [C_i] = \mu = \frac{n}{p}$$
Reminder

For a Binomial distribution, the following Chernoff bound holds:

\[ \forall \delta \in ]0, 1] \quad \Pr ((X - \mu) \geq \delta \mu) \leq \exp \left( \frac{-\mu \delta^2}{3} \right) \]

Therefore, the probability that \( C_i \) is at least \( \nu \) is bounded as

\[ \Pr (C_i \geq \nu) = \Pr (C_i \geq 2\mu) = \Pr ((C_i - \mu) \geq \mu) \leq \exp \left( \frac{-\mu}{3} \right) = n^{-\frac{\mu}{3}} \log e = n^{-\frac{n}{3p \ln n}} = n^{-\epsilon_1} \]

with \( \epsilon_1 := \frac{n}{3p \ln n} \). Since \( p < \frac{n}{3 \ln n} \), we have \( \epsilon_1 > 1 \). Finally, we have

\[ \Pr \left( \max_i C_i \geq \nu \right) \leq \sum_i \Pr (C_i \geq \nu) \leq pn^{-\epsilon_1} \leq \sqrt{n}n^{-\epsilon_1} = n^{-\epsilon_2} \]

with \( \epsilon_2 > \frac{1}{2} \).
We now want to prove that \( \max_i D_i \in O \left( \frac{n}{p} \right) \). In more detail

**Lemma 2**

With high probability

\[
\forall p < \sqrt{\frac{n}{2 \ln n}} \quad \max_i D_i < 2\nu = 4\frac{n}{p}
\]

Let \( Q_i := \frac{C_1}{p} \) be the number of elements between \( S_{i-1} \) and \( S_i \) in processor 1 just after the splitters have been defined. Clearly,

\[
C_1 \leq \nu \quad \Leftrightarrow \quad Q_i \leq \frac{\nu}{p}
\]

Consider the following events:

\[ A : \quad D_i \geq 2\nu \quad B : \quad C_1 \leq \nu \]

We now want to prove that \( \Pr(A) \) is small.
Reminders

Law of total probability

\[ \Pr(A) = \Pr(A|B) \Pr(B) + \Pr(A|\overline{B}) \Pr(\overline{B}) \]

Bayes’ law

\[ \Pr(A|B) = \frac{\Pr(B|A) \Pr(A)}{\Pr(B)} \]

Since \( \Pr(\overline{B}) \leq n^{-1} \) we have

\[ \Pr(A) \leq \Pr(A|B) \Pr(B) + n^{-1} = \Pr(B|A) \Pr(A) + n^{-1} \leq \Pr(B|A) + n^{-1} \]

We now need to compute an upper bound on

\[ \Pr(B|A) = \Pr(C_1 \leq \nu|D_i \geq 2\nu) = \Pr \left( Q_i \leq \frac{\nu}{p} \middle| D_i \geq 2\nu \right) \]
Reminder

For a Binomial distribution, the following Chernoff bound holds:

\[ \forall \delta \in ]0, 1[ \quad \Pr \left( (X - \mu) \leq -\delta \mu \right) \leq \exp \left( -\frac{\mu \delta^2}{2} \right) \]

Let \( q := \frac{1}{p} \) and \( r := 2\nu \), and let \( X \sim B(q, r) \) \( \Rightarrow \mu = 2\frac{\nu}{p} \). Then

\[
\Pr(B|A) = \Pr \left( Q_i \leq \frac{\nu}{p} \left| D_i \geq 2\nu \right. \right) \leq \Pr \left( X \leq \frac{\nu}{p} \right) \leq \exp \left( \frac{-\mu}{8} \right) \leq n^{-1}
\]

This (after a few steps) finally implies

\[
\Pr \left( \max_{i} D_i \geq 2\nu \right) \leq n^{-\frac{1}{2}}
\]
Outline

1. List Ranking
2. Sorting
3. Pattern Matching
The Problem

- We are given a (binary) pattern
  \[ X = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n \]

- And a text
  \[ Y = (y_1, y_2, \ldots, y_m) \in \{0, 1\}^m, \quad \text{with} \quad m \geq n \]

- Consider the substrings of \( Y \) of length \( n \):
  \[ Y(i) := (y_i, y_{i+1}, \ldots, y_{i+n-1}) \]

Goal

Find all the matches of \( X \) in \( Y \), i.e., find \( R \) such that

\[ R := \{ i \geq 0 : X = Y(i) \} \]

Note: The naive algorithm takes, in the worst case, time \( n(m - n + 1) \), while Knuth–Morris–Pratt one takes time \( \Theta(m + n) \)
Fingerprinting

- Let \( S \) be a finite set
- For all \( p \) in \( S \) we define a fingerprint function

\[
\forall p \in S \quad \phi_p : \{0, 1\}^n \rightarrow D_p
\]

- The cardinality of \( D_p \) is smaller than \( \#\{0, 1\}^n = 2^n \), so \( \phi_p \) is not injective
- We want \( \phi_p (Y(i + 1)) \) to be easily computable from \( \phi_p (Y(i)) \)
- Clearly, if \( Z_1 = Z_2 \) then \( \phi_p (Z_1) = \phi_p (Z_2) \)
- However, since \( \phi_p \) is not injective there exist \( Z_1 \neq Z_2 \) with \( \phi_p (Z_1) = \phi_p (Z_2) \)
- We want this to happen with low probability

\[
p \in \mathcal{U}(S) \quad \Pr (\phi_p (Z_1) = \phi_p (Z_2) \mid Z_1 \neq Z_2) \text{ is small}
\]
Modular Fingerprints

Definition

- We interpret the string $Z = (z_1, z_2, \ldots, z_n)$ as a number $\phi(Z)$

$$\phi(Z) := \sum_{i=1}^{n} z_i 2^{n-i} = z_1 2^{n-1} + z_2 2^{n-2} + \cdots + z_n$$

**Note:** $Z$ and $\phi(Z)$ have the same representation (n bits)

- Let $S_k$ be the set of prime numbers not larger than $k$

$$S_k := \{ p \in \mathbb{N} : p \leq k \land p \text{ is prime} \}$$

- Given some prime number $p \in S_{nm^2}$ let $\phi_p(Z)$ be the remainder of the division between $\phi(Z)$ and $p$:

$$\phi_p(Z) := \phi(Z) \mod p$$

**Note:** $\phi_p(Z)$ can be expressed with $O(\log m + \log n)$ bits
We can write $\phi(Z)$ as

$$\phi(Z) = ((z_1 \cdot 2 + z_2) \cdot 2 + z_3) \cdot 2 + z_4 \cdots$$

Let $a +_p b := a + b \mod p$ and $a \cdot_p b := a \cdot b \mod p$. We can compute $\phi_p(Z)$, using only finite arithmetic, as

$$\phi_p(Z) = ((z_1 \cdot_p 2 +_p z_2) \cdot_p 2 +_p z_3) \cdot_p 2 +_p z_4 \cdots$$

We can compute $\phi_p(Y(i+1))$ from $\phi_p(Y(i))$ in constant time as

$$\phi_p(Y(i+1)) = (\phi_p(Y(i)) -_p \phi_p(2^{n-1}) \cdot_p y_i) \cdot_p 2 +_p y_{i+n}$$
Let $\pi(k)$ be the number of primes up to $k$, i.e., $\pi(k) := \#S_k$.

**Fact 1**

The product $b(k)$ of the primes up to (a large enough) $k$ is larger than $2^k$:

$$\forall k \geq 29 \quad b(k) := \prod_{i \in S_k} i > 2^k$$

**Corollary:** If $a \leq 2^k [< b(k)]$, then it has less than $\pi(k)$ prime factors (proof by reductio ad absurdum).

**Fact 2**

$$\forall k \geq 17 \quad \frac{k}{\ln k} \leq \pi(k) \leq 1.26 \frac{k}{\ln k}$$
Let $Z_1 \neq Z_2$ be binary strings of length $n$ and

$$\delta(Z_1, Z_2) := |\phi(Z_1) - \phi(Z_2)|$$

- $\phi_p(Z_1) = \phi_p(Z_2)$ if and only if $p$ divides $\delta(Z_1, Z_2)$
- Note that $\delta(Z_1, Z_2) < 2^n$ and therefore it has fewer than $\pi(n)$ factors

If $p \in \mathcal{U}(S_k)$ (i.e., chosen uniformly at random in $S_k$) we have

$$\Pr(\phi_p(Z_1) = \phi_p(Z_2)) = \Pr(p \text{ divides } \delta(Z_1, Z_2)) \leq \frac{\pi(n)}{\pi(k)}$$

If $k = nm^2$ we have

$$\frac{\pi(n)}{\pi(k)} = \frac{\pi(n)}{\pi(nm^2)} \leq 1.26 \frac{n \ln n}{\ln n} - \frac{2 \ln m}{nm^2} \leq \frac{4 \ln m}{m^2}$$
// Choose q fingerprint functions
for i ← 1 to q do
    p_i ← random element of S ;
end

// Initialize variables
match ← false ;
r ← 0 ;
while match = false ∧ r ≠ m − n do
    r ← r + 1 ;
    match ← true ;
    // If some fingerprint differs, there is no match
    for i ← 1 to q do
        if φ_p_i(X) ≠ φ_p_i(Y(r)) then
            match ← false ;
        end
    end
end
If $k = nm^2$ we have

$$\Pr(\phi_p(Z_1) = \phi_p(Z_2)|Z_1 \neq Z_2) \leq \alpha := \frac{4 \ln m}{m^2}$$

If we choose $q$ primes independently, the probability that we still have a false match is

$$\Pr(\text{false match at given position}) \leq \alpha^q$$

The probability to have at least one false match when scanning the entire text is

$$\Pr(\text{false match}) \leq 1 - (1 - \alpha^q)^{m-n+1} \leq m\alpha^q = m\left(\frac{4 \ln m}{m^2}\right)^q$$
Let $\epsilon$ be the null string and $Z_1 Z_2$ the concatenation of strings $Z_1$ and $Z_2$

Consider the following map

$$
\begin{align*}
\phi(\epsilon) & := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\phi(0) & := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\
\phi(1) & := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}
\end{align*}
$$

which extends to all the strings as

$$
\phi(Z_1 Z_2) := \phi(Z_1) \phi(Z_2)
$$

Let $S_k$ be the set of prime numbers not larger than $k$

Given some prime number $p \in S_{n,m^2}$ let $\phi_p(Z)$ be

$$
\phi_p(Z) := \phi(Z) \mod p
$$
Matrix Fingerprints
Properties

- For example

$$\phi(00110) = \begin{pmatrix} 3 & 2 \\ 7 & 5 \end{pmatrix} \quad \phi_5(00110) = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix}$$

- We can compute $\phi_p(Y(i + 1))$ from $\phi_p(Y(i))$ in a very elegant way as

$$\phi_p(Y(i + 1)) = \phi_p^{-1}(y_i) \phi_p(Y(i)) \phi_p(y_{i+n})$$

where the inverses of $\phi_p(0)$ and $\phi_p(1)$ are

$$\phi_p^{-1}(0) = \begin{pmatrix} 1 & 0 \\ p-1 & 1 \end{pmatrix} \quad \phi_p^{-1}(1) = \begin{pmatrix} 1 & p-1 \\ 0 & 1 \end{pmatrix}$$

- The probability of false match is similar to the case of normal modular fingerprints (details omitted)
Matrix Fingerprints
Parallelization

- $\phi_p(X)$ can be computed in time $O(\log n)$ by $\frac{n}{\log n}$ processors.
- Consider the following functions:

  \[
  \psi_p(k) := \phi_p(y_1 y_2 \ldots y_k) = \prod_{i=1}^{k} \phi_p(y_i)
  \]

  $\psi_p(k)$ can be computed, for all $k \leq m$, in time $O(\log m)$ by $\frac{m}{\log m}$ processors (prefix-sum).

- Note that

  \[
  \phi_p(Y(i)) = \psi_p^{-1}(i - 1)\psi_p(i + n - 1)
  \]

  can be computed in constant time using $m$ processors.

- The algorithm is work-optimal when the number of processors is up to $\frac{m}{\log m}$. 
