# Average-case Analysis and Lower Bounds by the Incompressibility Method

Tao Jiang
Dept of Computer Science
Univ. of California - Riverside

Joint work with Ming Li (Waterloo) and
Paul Vitanyi (CWI)

# *Outline*

1. Overview of Kolmogorov complexity

2. The incompressibility method

3. Trivial example: lower bound for sorting

4. Average complexity of Shellsort

5. Average complexity of Heapsort

6. Average complexity of sorting with networks of stacks and queues

7. Average complexity of boolean matrix multiplication

8. Average complexity of majority finding

9. Expected length of longest common subsequence

10. Expected size of Heilbronn's triangles

11. String matching and one-way pointers (heads)

## *Kolmogorov Complexity*

1. Introduced by R. Solomonoff (1960), A.N. Kolmogorov (1965), P. Martin-Lof (1966), and G. Chaitin (1969).

2. Also known as *descriptional complexity*, *algorithmic information complexity*, *Solomonoff-Kolmogorov-Chaitin complexity*.

3. It is concerned with the *a priori* probability, information content and randomness of an *individual* object.

010101010101...

011001010001...

A good introduction is:

M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Application*, Springer-Verlag.

## *Kolmogorov Complexity of an Object*

**Def.** The Kolmogorov complexity of object $x$, denoted $C(x)$, is the length of the *shortest program* that prints $x$.

What kind of programs?
Pascal, C, Java, or even some pseudo-language?

**Invariance Theorem.** (Solomonoff'60)
For any two languages $L_1$ and $L_2$,

$$|C_{L_1}(x) - C_{L_2}(x)| \leq c$$

for any $x$, where $c$ is some constant depending only on $L_1$ and $L_2$.

*I.e.*, the language does *not* matter as long as it's fixed.

**Def.** The conditional Kolmogorov complexity of $x$ relative to $y$, denoted $C(x|y)$, is the length of the shortest program that prints $x$ when given $y$ as input.

Program $\equiv$ description/encoding

## *An Example* — $C(0^n)$

Program 1:

$$\mathsf{Print}(\text{“}\underbrace{0\cdots0}_{n}{}'');$$

The length of description is $n + O(1)$ bits.

Program 2:
For $i$ := 1 to $n$ Print(“0”);

The length is $\log n + O(1)$ bits.

Are there shorter programs?

Yes if, *e.g.*, $n = 2^m$ for some integer $m$.

In general, $C(x)$ is uncomputable.

## *Incompressibility of Objects*

Let $S$ be a finite set of objects and $x \in S$.

**Def.** Object $x$ is $c$-*incompressible* if $C(x|S) \geq \log|S| - c$.

0-incompressible objects are also said to be Kolmogorov *random*.

**Incompressibility Lemma.** There are at least

$$(1 - \frac{1}{2^c})|S| + 1$$

$c$-incompressible elements in $S$.

**Proof.** Let $n = \log|S|$. There are at most

$$\sum_{l=0}^{n-c-1} 2^l = 2^{n-c} - 1 = \frac{2^n}{2^c} - 1 = \frac{|S|}{2^c} - 1$$

programs of lengths less than $n-c$. Each program prints at most one element of $S$. Q.E.D.

1. At least one element of $S$ is 0-incompressible.

2. More than half are 1-incompressible.

3. More than $3/4$ are 2-incompressible.

4. More than $(n-1)/n$ are $\log n$-incompressible.

## *The Incompressibility Method*

To prove a (lower or upper) bound:

1. Take an $f(n)$-incompressible object $x$ to construct a "typical" instance/input.

2. Establish the bound for this *fixed* instance $x$.

   Show that if the bound does not hold for $x$ then we can *compress* $x$ by giving a clever, short encoding of $x$.

**Remarks:**

(a) Such a typical instance

- possesses all statistical properties;
- makes a proof easy;
- cannot be recursively constructed.

(b) The result usually holds in the average/expected case since most objects are incompressible.

*Success Stories of the Incompressibility Method*

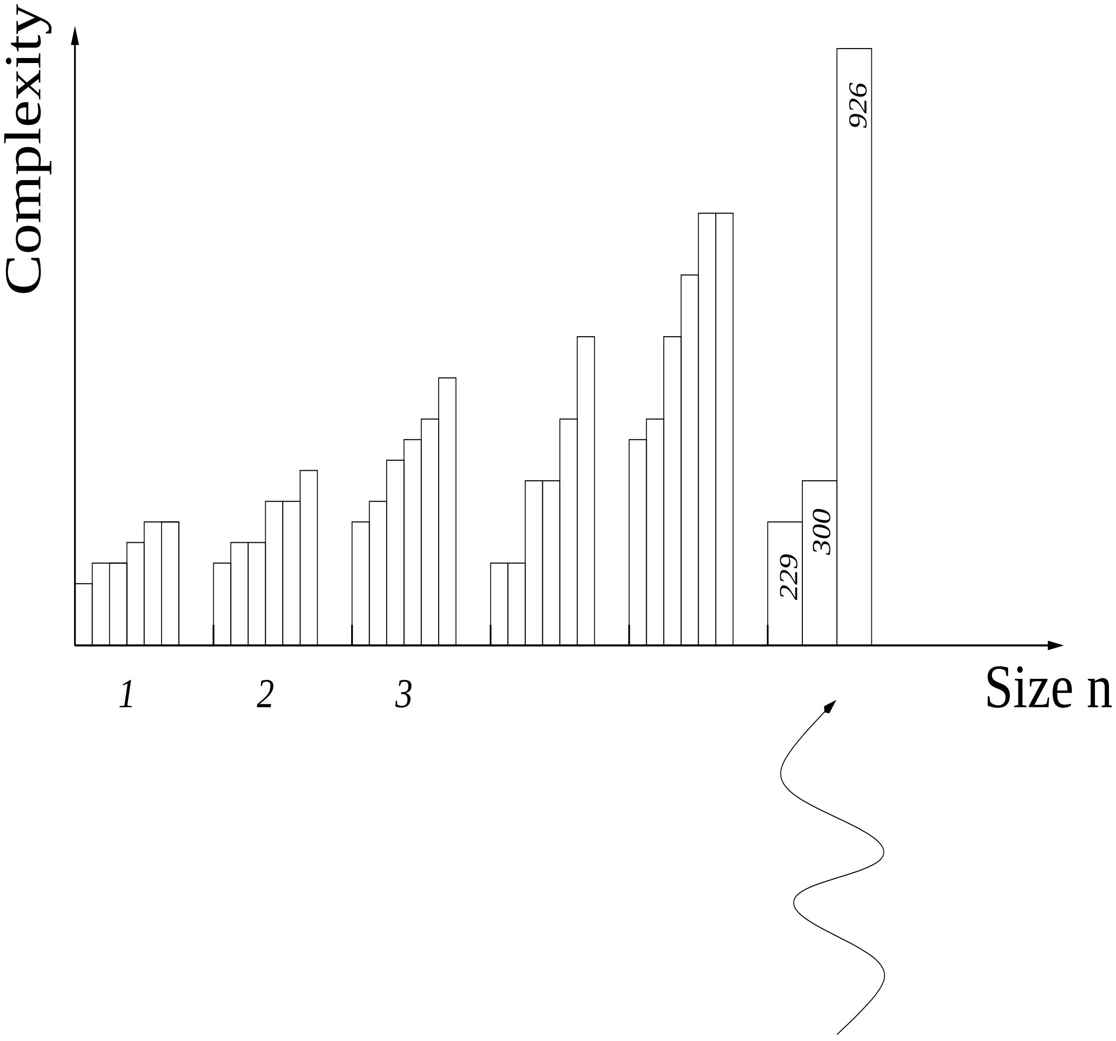
1. Solution of many longstanding open problems concerning the complexity of various models of computation.

2. Simpler proofs for many results in combinatorics, parallel computation, VLSI, formal language/automata theory, etc.

3. Average-case analysis of classical algorithms such as Heapsort, Shellsort, Boolean matrix multiplication, etc.

# *Average-Case (Time) Complexity Analysis*

Assume all instances of size $n$ occur with equal probability.



Election 2000:
The Florida recount

## *A Trivial Example: Lower Bound for Sorting*

**Theorem.** Any comparison based sorting algorithm requires $\Omega(n \log n)$ comparisons to sort $n$ elements on the average.

**Proof.** Let $A$ be any comparison based sorting algorithm. Fix a 1-incompressible permutation $I$ of $\{1, \ldots, n\}$ such that

$$C(I|A,n) \geq \log n! - 1$$

Suppose $A$ sorts $I$ in $m$ comparisons. We can describe $I$ by listing

- a description of this encoding scheme in $O(1)$ bits,
- the binary outcomes of the $m$ comparisons in $m$ bits.

Since $m + O(1) \geq C(I|A,n)$,

$$m \geq C(I|A,n) - O(1) \geq \log n! - O(1) = \Omega(n \log n)$$

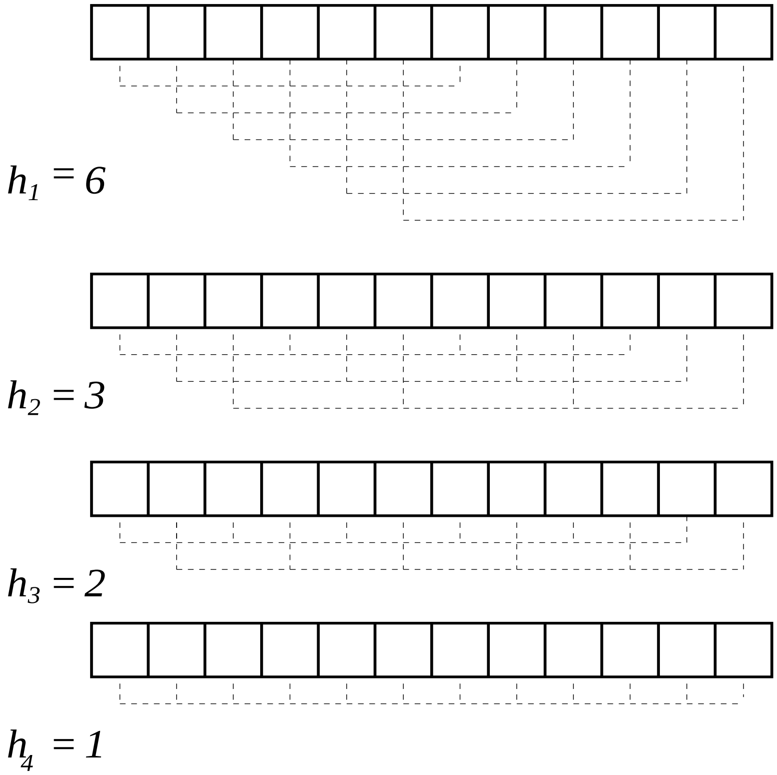
Since more than half of the permutations are 1-incompressible, the average number of comparisons required by $A$ is

$$\Omega(n \log n)/2 = \Omega(n \log n)$$

# *Average Complexity of Shellsort*

**Algorithm** p-pass Shellsort with increments
$h_1 > \ldots > h_p = 1$;

```
1. Input: list $\pi = x_1, \ldots, x_n$;
2. For $k := 1$ to $p$
3.     Divide list into $h_k$ equally spaced sublists
       (or chains), each of length $n/h_k$;
4.     Perform Insertion Sort within each sublist;
```



## *Previous Results on Shellsort*

**Worst case**:

1. $O(n^2)$ time in $\log n$ passes (Shell'59)
2. $O(n^{3/2})$ time (Papernov-Stasevitch'65)
3. $O(n \log^2 n)$ time in $\log^2 n$ passes (Pratt'72)
4. $O(n^{1+\epsilon/\sqrt{\log n}})$ time in $(8/\epsilon^2) \log n$ passes (Incerpi and Sedgewick'85 and Chazelle'??)
5. $\Omega(n^{1+\epsilon/\sqrt{p}})$ time for $p$ passes (Plaxton, Poonen and Suel'92)

**Average case**:

1. $\Theta(n^{5/3})$ time for 2-pass (Knuth'73)
2. Complicated analysis for 3-pass (Yao'80)
3. $O(n^{23/15})$ for 3-pass (Janson and Knuth'96)

**Open**:

1. Average complexity of general $p$-pass Shellsort (Plaxton, Poonen and Suel'92; Sedgewick'96,97)
2. Can Shellsort achieve $O(n \log n)$ on the average? (Sedgewick'97)

## *An Average Lower Bound for Shellsort*

**Theorem.** For any $p \leq \log n$, the average-case running time of a $p$-pass Shellsort is $\Omega(pn^{1+1/p})$ under uniform distribution of input permutations.

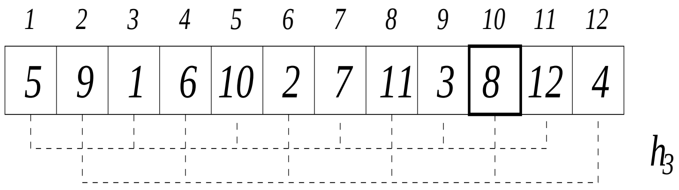
1. $p = 1$: $\Omega(n^2)$ is tight for Insertion Sort.
2. $p = 2$: $\Omega(n^{3/2})$ vs $Theta(n^{5/3})$ of Knuth.
3. $p = 3$: $\Omega(n^{4/3})$ vs $O(n^{23/15})$ of Janson and Knuth.
4. $p = \log n / f(n)$: $\Omega((n \log n) 2^{f(n)}/f(n))$.
5. $p > \log n$: $\Omega(pn)$ is a trivial lower bound.

Hence, in order for a $p$-pass Shellsort to achieve $O(n \log n)$ time on the average, $p = \Theta(\log n)$.

### *Proving the Average Lower Bound*

Fix any $(h_1, \ldots, h_p)$ Shellsort $A$. We prove $\Omega(pn^{1+1/p})$ is a lower bound on the average number of inversions.

**Def.** For any $1 \leq i \leq n$ and $1 \leq k \leq p$, consider the $h_k$-*chain* containing element $i$ at the beginning of pass $k$. Let $m_{i,k}$ be the number of elements that are to the left of $i$ and larger than $i$.



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 9 | 1 | 6 | 10 | 2 | 7 | 11 | 3 | $\boxed{8}$ | 12 | 4 |

*e.g.* $m_{8,3} = 2$

**Fact.** The Insertion Sort in pass $k$ makes precisely $\sum_{i=1}^{n} m_{i,k}$ inversions (or $\sum_{i=1}^{n} m_{i,k} + 1$ comparisons).

Let $M$ denote the total number of inversions:

$$M := \sum_{k=1}^{p} \sum_{i=1}^{n} m_{i,k}$$

*Proving the Average Lower Bound – II*

Fix a $\log n$-incompressible permutation $\pi$ of $\{1,\ldots,n\}$ with

$$C(\pi|n,A) \geq \log n! - \log n$$

**Fact.** Given all the numbers $m_{i,k}$'s for the input permutation $\pi$, we can uniquely reconstruct $\pi$.

**Proof.** Reconstruct the initial permutation of each pass backward.

Hence,

$$C(m_{1,1},\ldots,m_{n,p}|n,A) \geq C(\pi|n,A) \geq \log n! - \log n$$

Since there are $\binom{M+np-1}{np-1}$ possible divisions of $M$ into $np$ nonnegative integral summands $m_{i,k}$'s,

$$\log M + \log \binom{M+np-1}{np-1} \geq C(m_{1,1},\ldots,m_{n,p}|n,A) \geq \log n! - \log n$$

Noting $\log M = O(\log n)$, a careful calculation shows

$$M = \Omega(pn^{1+\frac{1}{p}})$$

The average number of inversions required is thus:

$$\frac{n-1}{n} \cdot \Omega(pn^{1+\frac{1}{p}}) + \frac{1}{n} \cdot 0 = \Omega(pn^{1+\frac{1}{p}})$$

## *The Average Complexity of Heapsort*

**Algorithm** Heapsort(*var* $A[1..n]$);

```
1. Heapify $A$;

2. For $i$ := 1 to $n$
       $A[i]$ := DELETEMIN($A[i..n]$);
```

**Fact.** Heapify requires $O(n)$ time.

**Fact.** Heapsort runs in $O(n \log n)$ time in the worst case.

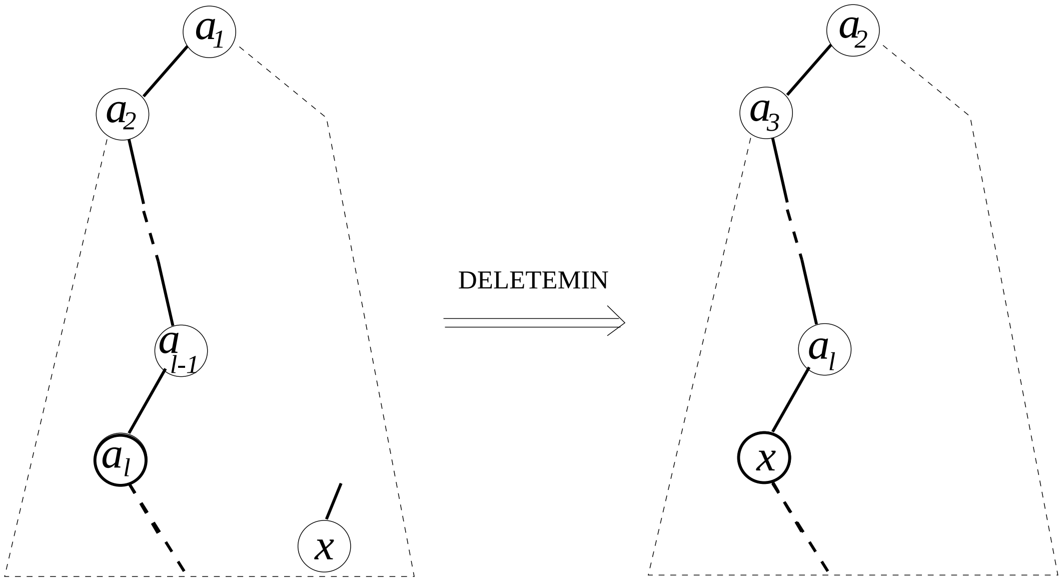
## *Two Implementations of DELETEMIN*

Williams' DELETEMIN($A[1..n]$);

```
$x := A[n]$; $k := 1$;
While $x > \min(A[2k], A[2k+1])$
    $A[k] := \min(A[2k], A[2k+1])$; //shift up//
    $k := 2k$ or $k := 2k+1$; //move down//
$A[k] := x$;
```

Floyd's DELETEMIN($A[1..n]$):

1. Find the path leading to a leaf while shifting all elements up;
2. Climb up the path and insert $x$ at its correct location.



**Fact.** Williams requires $2l$ comparisons and Floyd requires $2\log n - l$ comparisons.

*Avg Analysis of Williams and Floyd Heapsorts*

**Theorem.** Both Williams and Floyd Heapsorts require $2n \log n$ comparisons in the worst case.

**Theorem.** (Schaffer and Sedgewick'92)
On the average, Williams Heapsort requires $2n \log n - O(n)$ comparisons and Floyd Heapsort requires $n \log n + O(n)$ comparisons.

The following is a simple incompressibility proof due to Ian Munro.

Fix a $\log n$-incompressible input permutation $\pi$ of $\{1, \ldots, n\}$ with

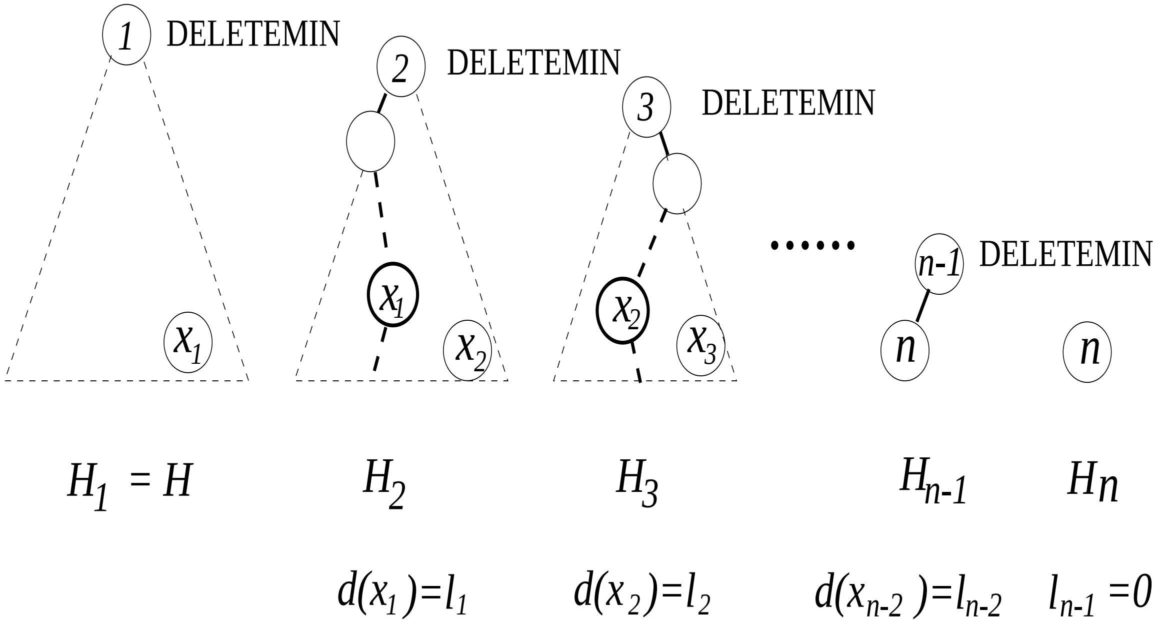
$$C(\pi | n) \geq \log n! - \log n$$

Denote the resulting heap of Heapify as $H$.

**Fact.** $C(\pi | n, H) = O(n)$

Hence, $C(H|n) \geq C(\pi|n) - C(\pi|n, H) \geq \log n! - O(n)$.

## *Insertion Depths are the Key!*



**Claim.** Williams needs $\sum_{i=1}^{n-1} 2l_i$ comparisons.

**Claim.** The number of Floyd's comparisons is

$$\sum_{i=1}^{n-1} 2\log(n-i) - l_i = 2\log(n-1)! - \sum_{i=1}^{n-1} l_i$$

**Claim.** $\sum_{i=1}^{n-1} l_i \geq C(H|n)$

**Proof.** Given the location of $x_i$ in $H_{i+1}$, we can recover $H_i$ from $H_{i+1}$. The location can be described by the path from the root of $H_{i+1}$ to $x_i$, in $l_i$ bits.

## *The Final Calculation*

Therefore, Williams Heapsort requires at least

$$2C(H|n) \geq 2\log n! - O(n) \approx 2n\log n - O(n)$$

comparisons and Floyd Heapsort requires at most

$$2\log(n-1)! - C(H|n) \leq \log n! + O(n) \approx n\log n + O(n)$$

comparisons on $\log n$-incompressible permutation $\pi$.

On the average, Williams and Floyd Heapsorts need

$$\frac{n-1}{n} \cdot (2n\log n - O(n)) + \frac{1}{n} \cdot n = 2n\log n - O(n)$$

$$\frac{n-1}{n} \cdot (n\log n + O(n)) + \frac{1}{n} \cdot (2n\log n) = n\log n + O(n)$$

comparisons, respectively.

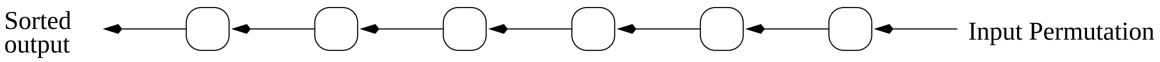
**Theorem.** Both Williams and Floyd Heapsorts require $n\log n + O(n)$ data moves on the average.

## *Sorting with Networks of Stacks and Queues*

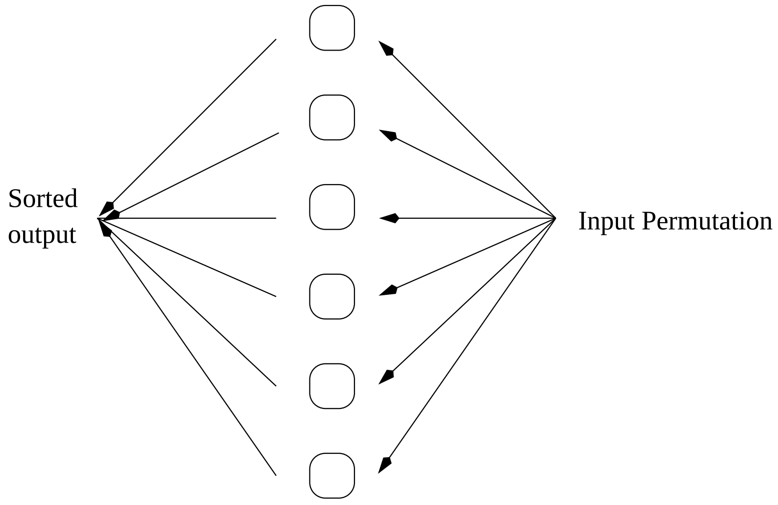
Knuth and Tarjan studied the problem in the early 70s.

The main question is: assuming the stacks or queues are arranged sequentially or in parallel, how many stacks or queues are needed to sort $n$ elements with comparisons only?

Here, the input sequence is scanned from left to right and the elements follow the arrows to go to the next stack or queue or output.



A sequence of stacks or queues



stacks/queues

*Sorting with Sequential Stacks*

**Fact.** In the worst case, $\log n$ stacks suffice, and $\frac{1}{2}\log n$ stacks are necessary (Tarjan'72).

**Theorem.** On the average, at least $\frac{1}{2}\log n$ stacks are needed for sequential stack sort.

**Proof.** Fix a $\log n$-incompressible permutation $\pi$. Assume $k$ stacks are sufficient to sort $\pi$. Since exactly $n$ elements pass through each stack, we can encode the sequence of pushes and pops on a stack uniquely as a binary string of $n$ 0's and $n$ 1's. Hence, the input permutation is described by $2kn$ bits. Therefore,

$$2kn \geq \log n! - \log n = n\log n - O(\log n)$$

and approximately $k \geq \frac{1}{2}\log n$.

## *Sorting with Parallel Stacks*

**Fact.** The permutation $2, 3, 4, \ldots, n, 1$ requires $n - 1$ parallel stacks.

We show that on the average, the number of parallel stacks needed to sort $n$ elements is $\Theta(\sqrt{n})$.

The bound is implied by the connection between sorting with parallel stacks and *longest increasing subsequences* (Tarjan'72) and the bounds on the length of longest increasing subsequences of random permutations given in Kingman'73, Logan and Shepp'77, and Kerov and Versik'77), using deep results from probability theory (such as Kingman's ergodic theorem).

## *An Average-Case Upper Bound*

**Theorem.** On the average, $O(\sqrt{n})$ parallel stacks are needed to sort $n$ elements.

**Proof.** Fix a $\log n$-incompressible permutation $\pi = x_1, \ldots, x_n$. The stacks are named $S_0, S_1, \ldots$. We use the following trivial algorithm (Tarjan'72) to sort $\pi$. Here, the stacks are named $S_0, S_1, \ldots$.

### Algorithm Parallel-Stack-Sort

```
1. For $i = 1$ to $n$ do
       Scan the stacks from left to right, and push $x_i$ on the the
          first stack $S_j$ whose top element is larger than $x_i$. If
          such a stack doesn't exist, put $x_i$ on the first empty
          stack.

2. Pop the stacks in the ascending order of their top elements.
```

## *An Average-Case Upper Bound*

We need show that algorithm Parallel-Stack-Sort uses $O(\sqrt{n})$ stacks on the permutation $\pi$.

**Claim.** If the algorithm uses $m$ stacks on $\pi$ then we can identify an increasing subsequence of $\pi$ of length $m$ (Tarjan'72).

**Claim.** The permutation $\pi$ has no increasing subsequence of length longer than $e\sqrt{n}$, where $e$ is the natural constant.

**Proof.** Suppose that $\sigma$ is a longest increasing subsequence of $\pi$ and $m = |\sigma|$. We can encode $\pi$ by:

1. a description of this encoding scheme in $O(1)$ bits;
2. the number $m$ in $\log m$ bits;
3. $\sigma$ as a combination in $\log \binom{n}{m}$ bits;
4. the locations of the elements of $\sigma$ in $\pi$ in at most $\log \binom{n}{m}$ bits; and
5. the remaining $\pi$ with the elements of $\sigma$ deleted in $\log(n - m)!$ bits.

## *An Average-Case Upper Bound*

This description takes a total of

$$\log(n-m)! + 2\log\frac{n!}{m!(n-m)!} + \log m + O(1) + 2\log\log m$$

bits. Using Stirling approximation and the fact $\sqrt{n} \leq m = o(n)$, we upper bound the above expression as:

$$\begin{aligned} & \log n! + \log\frac{(n/e)^n}{(m/e)^{2m}((n-m)/e)^{n-m}} \\ & +O(\log n) \\ \approx\ & \log n! + m\log\frac{n}{m^2} + (n-m)\log\frac{n}{n-m} \\ & +m\log e + O(\log n) \\ \approx\ & \log n! + m\log\frac{n}{m^2} + 2m\log e + O(\log n) \end{aligned}$$

Since this must exceed $\log n! - \log n$, we obtain (approximately) $m \leq e\sqrt{n} = O(\sqrt{n})$.

*An Average-Case Lower Bound*

**Theorem.** On the average, $\Omega(\sqrt{n})$ parallel stacks are required to sort a permutation.

**Proof.** Let $A$ be a sorting algorithm using parallel stacks. Again, fix a $\log n$-incompressible permutation $\pi$. Suppose $A$ uses $T$ parallel stacks to sort $\pi$. We encode the sequence of moves in the sorting process as a sequence of the following terms:

- push to stack $i$,
- pop stack $j$,

where the element to be pushed is the next unprocessed element from the input sequence and the popped element is written as the next output element.

Each of these terms requires $\log T$ bits, and totally we use precisely $2n$ terms. Thus we have a (unique) description of $\pi$ in $2n \log T$ bits, which must exceed $n \log n - O(\log n)$. So, $T \geq \sqrt{n} = \Omega(\sqrt{n})$.

## *Sorting with Parallel Queues*

Sorting cannot be done with a sequence of queues.

**Fact.** The permutation $n, n-1, \ldots, 1$ requires $n$ queues to sort (Tarjan'72).

**Theorem.** On the average, $\Theta(\sqrt{n})$ parallel queues are needed to sort $n$ elements.

The bound is implied by the connection between sorting with parallel queues and *longest decreasing subsequences* (Tarjan'72) and the bounds in Kingman'73, Logan and Shepp'77, and Kerov and Versik'77, with sophisticated proofs. We can prove the above result using simple incompressibility arguments similar to the case of parallel stacks.

## *Average Complexity of Boolean Matrix Multiplication*

Problem: Multiply two $n \times n$ boolean matrices $A = (a_{i,j})$ and $B = (b_{i,j})$.

Previous results:

1. Best worst-case time complexity $O(n^{2.376})$ due to Coppersmith and Winograd, 1987.

2. In 1973, O'Neil and O'Neil gave a simple algorithm described in next slide that runs in $O(n^3)$ time in the worst case but achieves an average time complexity of $O(n^2)$.

Here we analyze the average-case complexity of the algorithm using a simple incompressibility argument.

*The Algorithm of O'Neil and O'Neil*

**Algorithm** QuickMultiply($A, B$)

```
1. Let $C = (c_{i,j})$ denote the result of multiplying $A$ and $B$.
2. For $i :=$ 1 to $n$ do
3.     Let $j_1 < \cdots < j_m$ be the indices with $a_{i,j_k} = 1$,
       $1 \leq k \leq m$.
4.     For $j :=$ 1 to $n$ do
5.         Search the list $b_{j_1,j}, \ldots, b_{j_m,j}$ sequentially for a bit 1.
6.         Set $c_{i,j} = 1$ if a bit 1 is found, or $c_{i,j} = 0$ otherwise.
```

**Theorem.** Suppose that the elements of $A$ and $B$ are drawn uniformly and independently. Algorithm QuickMultiply runs in $O(n^2)$ time on the average.

## *The Average Complexity of QuickMultiply*

Let $n$ be a sufficiently large integer. The average time of QuickMultiply is trivially bounded between $O(n^2)$ and $O(n^3)$. Since $(n-1)2^{2n^2}/n$ of the $2^{2n^2}$ pairs of $n \times n$ boolean matrices are $\log n$-incompressible, it suffices to consider $\log n$-incompressible boolean matrices.

Take a $\log n$-incompressible binary string $x$ of length $2n^2$, and form two $n \times n$ boolean matrices $A$ and $B$ straightforwardly. We show that QuickMultiply spends $O(n^2)$ time on $A$ and $B$.

Consider an arbitrary $i$, where $1 \leq i \leq n$. It suffices to show that the $n$ sequential searches done in Steps 4 – 6 of QuickMultiply take a total of $O(n)$ time.

*The Average Complexity of QuickMultiply*

By the statistical properties of incompressible strings, we know that at least

1. $n/2 - O(\sqrt{n \log n})$ of the searches find a 1 in the first step,
2. $n/4 - O(\sqrt{n \log n})$ searches find a 1 in two steps,
3. $n/8 - O(\sqrt{n \log n})$ searches find a 1 in three steps, and so on.

**Claim.** Each of the searches takes at most $4 \log n$ steps.

Hence, the $n$ searches take at most a total of

$$\begin{aligned}
& (\sum_{k=1}^{\log n} (n/2^k - O(\sqrt{n \log n})) \cdot k) \\
& +(\log n) \cdot O(\sqrt{n \log n}) \cdot (4 \log n) \\
< \; & (\sum_{k=1}^{\log n} kn/2^k + O(\log^2 n \sqrt{n \log n}) \\
= \; & O(n) + O(\log^2 n \sqrt{n \log n}) \\
= \; & O(n)
\end{aligned}$$

steps.

**Claim.** Each of the searches takes at most $4\log n$ steps.

**Proof.** Suppose that for some $j$, $1 \leq j \leq n$, $b_{j_1,j} = \cdots = b_{j_{4\log n},j} = 0$. Then we can encode $x$ as

1. A description of the above discussion.
2. The value of $i$.
3. The value of $j$.
4. All bits of $x$ except the bits $b_{j_1,j}, \ldots, b_{j_{4\log n},j}$.

This encoding takes at most

$$\begin{aligned} & O(1) + 2\log n + 2n^2 - 4\log n + O(\log\log n) \\ < \;& 2n^2 - \log n \end{aligned}$$

bits for sufficiently large $n$, which contradicts the assumption that $x$ is $\log n$-incompressible.

## *Average Complexity of Finding the Majority*

Let $x = x_1 \cdots x_n$ be a binary string. The *majority* of $x$ is the bit (0 or 1) that appears more than $\lfloor n/2 \rfloor$ times in $x$.

Previous results:

1. Saks,Werman’91; Alonso,Reingold,Schott’93:
   $n - \nu(n)$ comparisons are necessary and sufficient in the worst case.

2. Alonso,Reingold,Schott’97:
   On the average, finding the majority requires at most $2n/3 - \sqrt{8n/9\pi} + O(\log n)$ comparisons and at least $2n/3 - \sqrt{8n/9\pi} + \Theta(1)$ comparisons.

Here we prove an average-case upper bound tight up to the first major term, using a simple incompressibility argument.

## *Average-case Upper Bound*

**Algorithm** Tournament($x = x_1 \cdots x_n$)

```
If $n = 1$ then return $x_1$ as the majority.
Elseif $n = 2$ then
    If $x_1 = x_2$ then return $x_1$ as the majority.
    Else return “no majority”.
Elseif $n = 3$ then
    If $x_1 = x_2$ then return $x_1$ as the majority.
    Else return $x_3$ as the majority.
Let $y = \epsilon$.
For $i := 1$ to $\lfloor n/2 \rfloor$ do
    If $x_{2i-1} = x_{2i}$ then append the bit $x_{2i}$ to $y$.
If $n$ is odd and $\lfloor n/2 \rfloor$ is even then append the bit $x_n$ to $y$.
Call Tournament($y$).
```

**Theorem.** On the average, algorithm Tournament requires at most $2n/3 + O(\sqrt{n \log n})$ comparisons.

## *Average Complexity of Algorithm Tournament*

Fix a $\log n$-incompressible string $x = x_1 \cdots x_n$:

$$C(x|n, \text{Tournament}) \geq n - \log n$$

For any $m \leq n$, $\sigma(m)$ denotes the complexity of Tournament on any $\log n$-incompressible string of length $m$.

**Lemma.** (Li,Vitanyi'93)
Let $s = s_1 \cdots s_m$ be a $c$-incompressible binary string. Among the $m/2$ pairs $(s_1, s_2), \ldots, (s_{2\lfloor m/2 \rfloor - 1}, s_{2\lfloor m/2 \rfloor})$, $m/4 \pm O(\sqrt{mc})$ are complementary.

So, the new string $y$ obtained in Tournament is at most $n/4 + O(\sqrt{n \log n})$ bits long.

**Lemma.** $y$ is also $\log n$-incompressible.

Hence, we have recurrence relation:

$$\sigma(m) \leq \lfloor m/2 \rfloor + \sigma(m/4 + O(\sqrt{m \log n}))$$

A trivial expansion gives $\sigma(n) \leq 2n/3 + O(\sqrt{n \log n})$.

The average complexity of Tournament is thus:

$$(2n/3 + O(\sqrt{n \log n}))\frac{n-1}{n} + n\frac{1}{n} = 2n/3 + O(\sqrt{n \log n})$$

## *Expected Length of Longest Common Subsequence*

Given two sequences (*i.e.* strings) $s = s_1 \ldots s_m$ and $t = t_1 \ldots t_n$, $s$ is a *subsequence* of $t$ if for some $i_1 < \ldots < i_m$, $s_j = t_{i_j}$.

A *longest common subsequence* (LCS) of $s$ and $t$ is a longest possible sequence $u$ that is a subsequence of both $s$ and $t$.

For example, 0011 is an LCS of 010101 and 000111.

For simplicity, assume the alphabet $\Sigma = \{0, 1\}$.

Consider two random sequences drawn independently from the uniformly distributed space of all binary strings of length $n$. Tight bounds on the expected LCS length for such two random sequences is a well-known open question in string combinatorics. The best bounds are $0.762n$ and $0.838n$.

## *An Upper Bound on Expected Length of LCS*

**Theorem.** The expected LCS length is at most $0.867n + o(n)$.

**Proof.** Let $n$ be a sufficiently large integer. The expected length of an LCS of two random sequences of length $n$ is trivially bounded between $n/2$ and $n$. By the Incompressibility Lemma, again it suffices to consider $\log n$-incompressible sequences.

Take a $\log n$-incompressible string $x$ of length $2n$, and let $s$ and $t$ be the first and second halves of $x$ respectively. Suppose that string $u = u_1 u_2 \cdots u_m$ is an LCS of $s$ and $t$. We re-encode $s$ and $t$ with respect to $u$ as follows. Write

$$s = \alpha_1 u_1 \alpha_2 u_2 \cdots \alpha_m u_m s'$$

$$t = \beta_1 u_1 \beta_2 u_2 \cdots \beta_m u_m t'$$

Encode

$$s(u) = 0^{l(\alpha_1)} 1 0^{l(\alpha_2)} 1 \cdots 0^{l(\alpha_m)} 1 s'$$

$$t(u) = 0^{l(\beta_1)} 1 0^{l(\beta_2)} 1 \cdots 0^{l(\beta_m)} 1 t'$$

Hence, the string $x$ can be described by the following information in the self-delimiting form:

1. A description of the above discussion.

2. The LCS $u$.

3. The new encodings $s(u)$ and $t(u)$ of $s$ and $t$.

Items 1 and 2 take $m + O(1)$ bits. Since $s(u)$ contains at least $m$ 1's, By simple counting and Stirling approximation

$$\begin{aligned}
C(s(u)) &\leq \log \sum_{i=m}^{n} \binom{n}{i} + O(1) \\
&\leq \log \left[ \frac{n}{2} \binom{n}{m} \right] + O(1) \\
&\leq \log n + \log \binom{n}{m} + O(1) \\
&\leq 2 \log n + n \log n - m \log m \\
&\quad -(n-m) \log(n-m) + O(1)
\end{aligned}$$

The second step in the above derivation follows from the trivial fact that $m \geq n/2$.

## *An Upper Bound on Expected Length of LCS*

Similarly, we have

$$\begin{aligned} C(t(u)) &\leq 2\log n + n\log n - m\log m \\ &\quad -(n-m)\log(n-m) + O(1) \end{aligned}$$

Hence, the above description requires a total size of

$O(\log n) + m + 2n\log n - 2m\log m - 2(n-m)\log(n-m)$.

Let $p = n/m$. Since $C(x) \geq 2n - \log n$, we have

$$\begin{aligned} 2n - \log n &\leq O(\log n) + m + 2n\log n - 2m\log m \\ &\quad -2(n-m)\log(n-m) \\ &= O(\log n) + pn - 2np\log p \\ &\quad -2n(1-p)\log(1-p) \end{aligned}$$

Dividing both sides by $n$, we obtain

$$2 \leq o(1) + p - 2p\log p - 2(1-p)\log(1-p)$$

Solving the inequality numerically, we get $p \leq 0.867 - o(1)$.

Note: Baeza-Yates and Navarro improved the analysis and obtained a slightly better upper of $0.860$.

## *A Lower Bound on Expected Length of LCS*

Next we prove a lower bound on the expected length of an LCS of two random sequences of length $n$. The proof uses the following greedy algorithm for computing common subsequences (not necessarily the longest ones).

**Algorithm** Zero-Major($s = s_1 \cdots s_n, t = t_1 \cdots t_n$)

```
1. Let $u := \epsilon$ be the empty string.
2. Set $i := 1$ and $j := 1$;
3. Repeat steps 4–6 until $i > n$ or $j > n$:
4.     If $s_i = t_j$ then begin append bit $s_i$ to string $u$;
       $i := i + 1$; $j := j + 1$ end
5.     Elseif $s_i = 0$ then $j := j + 1$
6.     Else $i := i + 1$
7. Return string $u$.
```

*A Lower Bound on Expected Length of LCS*

**Theorem.** Given two random sequences $s$ and $t$ of length $n$, the above algorithm Zero-Major produces a common subsequence $u$ of length at least $0.66666n - O(\sqrt{n \log n})$.

**Proof.** Let $n$ be a sufficiently large integer, and take a $\log n$-incompressible string $x$ of length $2n$. Let $s$ and $t$ be the first and second halves of $x$.

We encode $s$ and $t$ using information from the computation of Zero-Major on strings $s$ and $t$. Consider the comparisons made by Zero-Major in the order that they were made, and create a pair of strings $y$ and $z$ as follows. For every comparison $(s_i, t_j)$,

1. If $s_i$ and $t_j$ are complementary, we append a 1 to $y$.
2. Otherwise, append a bit 0 to $y$. Furthermore, if the preceding comparison $(s_{i'}, t_{j'})$ involves complementary bits, we append a bit 0 to the string $z$ if $i' = i - 1$ or a bit 1 if $j' = j - 1$.

When one string ($s$ or $t$) is exhausted, we append the remaining part (say $w$) of the other string to $z$.

## *An Example of Zero-Major Encoding*

Consider strings $s = 1001101$ and $t = 0110100$. Algorithm Zero-Major produces a common subsequence $u = 0010$, by the following comparisons:

```
s =               10  01101
comparisons       *|**||*|*
t =                01101 0 0
```

The above encoding scheme yields $y = 101100101$ and $z = 01100$. Here, $w = 0$.

*Some Properties of the Coding Strings*

It is easy to see that the strings $y$ and $z$ uniquely encode $s$ and $t$ and, $l(y) + l(z) = 2n$. Since

$$C(yz) \geq C(x) - 2\log n \geq 2n - 3\log n - O(1)$$

and $C(z) \leq l(z) + O(1)$, we have

$$C(y) \geq l(y) - 3\log n - O(1)$$

Similarly, we can obtain

$$C(z) \geq l(z) - 3\log n - O(1)$$

and

$$C(w) \geq l(w) - 3\log n - O(1)$$

where $w$ is the string appended to $z$ at the end of the above encoding.

## *Proving the Lower Bound*

Let us estimate the length of the common subsequence $u$. Let $\#zeroes(s)$ and $\#zeroes(t)$ be the number of 0's contained in $s$ and $t$. Clearly, $u$ contains $\min\{\#zeroes(s), \#zeroes(t)\}$ 0's. Since both $s$ and $t$ are $\log n$-incompressible, we know

$$n/2 - O(\sqrt{n \log n}) \leq \#zeroes(s) \leq n/2 + O(\sqrt{n \log n})$$

$$n/2 - O(\sqrt{n \log n}) \leq \#zeroes(t) \leq n/2 + O(\sqrt{n \log n})$$

Hence, $w$ has at most $O(\sqrt{n \log n})$ 0's. Combining with the fact that $C(w) \geq l(w) - 3 \log n - O(1)$, we claim

$$l(w) \leq O(\sqrt{n \log n}).$$

Hence, $w$ has at most $O(\sqrt{n \log n})$ 0's. Since $l(z) - l(w) = l(u)$, we have a lower bound on $l(u)$:

$$l(u) \geq l(z) - O(\sqrt{n \log n}).$$

## *Proving the Lower Bound*

On the other hand, since every bit 0 in the string $y$ corresponds to a unique bit in the common subsequence $u$, we have

$$l(u) \geq \#zeroes(y)$$

Since $C(y) \geq l(y) - 2\log n - O(1)$,

$$l(u) \geq \#zeroes(y) \geq l(y)/2 - O(\sqrt{n \log n}).$$

Hence,

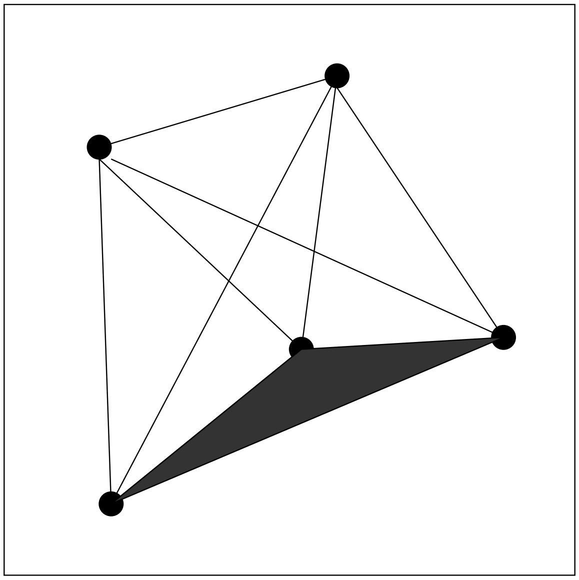
$$3l(u) \geq l(y) + l(z) - O(\sqrt{n \log n}) \geq 2n - O(\sqrt{n \log n}).$$

That is,

$$l(u) \geq 2n/3 - O(\sqrt{n \log n}) \approx 0.66666n - O(\sqrt{n \log n})$$

*Heilbronn's Triangles*

Consider $n$ points $x_1, \ldots, x_n$ in the unit square in the plane. Denote by $\Delta(x_1, \ldots, x_n)$ the smallest area of any triangle formed by three points.



Define

$$\Delta = \max_{x_1, \ldots, x_n} \Delta(x_1, \ldots, x_n)$$

**Question:** (Heilbronn'1950)
How large is $\Delta$?

*History of Heilbronn’s Triangles*

1. Heilbronn’50: $O(1/n^2)$ ?

2. Erdős’50: $\Omega(1/n^2)$

3. Roth’51: $O(1/(n\sqrt{\log logn})$

4. Schmidt’72: $O(1/(n\sqrt{\log n})$

5. Roth’72: $O(1/n^{1.105})$

6. Roth’72: $O(1/n^{1.117})$

7. Komlós,Pintz,Szemerédi’81: $O(1/n^{1.142})$

8. Komlós,Pintz,Szemerédi’82: $\Omega(\log n/n^2)$

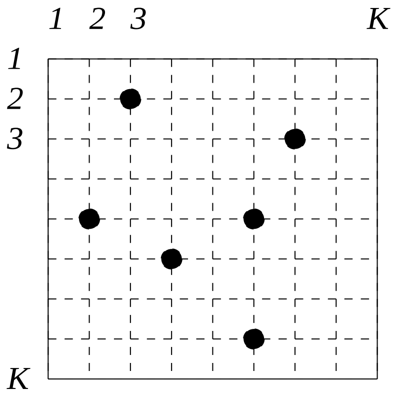
9. Bertram-Kretzberg,Hofmeister,Lefmann’97: $\Omega(\log n/n^2)$

**Theorem.** (Jiang,Li,Vitanyi’98)
Assuming uniform distribution of points $\{x_1, \ldots, x_n\}$,

$$\Delta_{expected} = \mathbf{E}(\Delta(x_1, \ldots, x_n)) = \Theta(1/n^3)$$

## *Proving the Lower Bound*

Consider a $K \times K$ grid on the unit square, where $K \gg n^3$, and distribute points at grid intersections.



Each point is fully described by a pair of coordinates $(x, y)$, where $1 \leq x, y \leq K$.

There are $\binom{K^2}{n}$ ways to put $n$ points.

Fix a $\delta$-incompressible distribution $\{x_1, \ldots, x_n\}$.

$$C(x_1, \ldots, x_n | n, K) \geq \log \binom{K^2}{n} - \delta$$

## *Warm-up*

**Lemma.** No three points can be collinear, and thus the smallest triangle area is $\Omega(1/(2(K-1)^2))$.

**Proof.** Suppose points $x_i$, $x_j$ and $x_k$ are on a straight line. Observe that given $x_i$ and $x_j$, we can describe $x_k$ in just $\log K$ bits. So we can encode $x_1, \ldots, x_n$ by specifying

- this encoding scheme in $O(1)$ bits;
- the locations of $x_i, x_j, x_k$ in $5 \log K$ bits; and
- the distribution of the other $n-3$ points in $\log \binom{K^2}{n-3}$ bits.

Altogether this description takes

$$\begin{aligned} & \log \binom{K^2}{n-3} + 5\log K + O(1) \\ = \; & (n-3)\log \frac{K^2}{n-3} + (n-3)\log e - \frac{1}{2}\log(n-3) \\ & \quad + 5\log K + O(1) \\ = \; & \log \binom{K^2}{n} + 3\log n - \log K + O(1) \\ < \; & \log \binom{K^2}{n} - \delta \end{aligned}$$

Contradiction since $x_1, \ldots, x_n$ is $\delta$-incompressible.

*Warm-up*

**Lemma.** No two pebbles can be on the same (horizontal or vertical) grid line.

**Proof.** Otherwise, we can form a description of the arrangement using

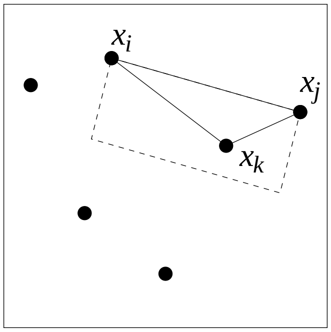
$$\begin{aligned}
& \log \binom{K^2}{n-2} + 3\log K + O(1) \\
= \;& (n-2)\log \frac{K^2}{n-2} + (n-2)\log e - \frac{1}{2}\log(n-2) \\
& \quad + 3\log K + O(1) \\
= \;& \log \binom{K^2}{n} + 2\log n - \log K + O(1) \\
< \;& \log \binom{K^2}{n} - \delta
\end{aligned}$$

With fixed $n$ and $K \to \infty$, we obtain $2\log n \geq \log K - \delta + O(1)$, which is a contradiction.

# *Proving Lower Bound* $\Omega(1/n^3)$

**Lemma.** $\Delta(x_1, \ldots, x_n) = \Omega(1/n^3)$.

**Proof sketch.** Suppose $\Delta(x_i, x_j, x_k) = 1/f(n) = o(1/n^3)$. Let $(x_i, x_j)$ be the longest edge. Given $x_i, x_j$, we can describe $x_k$ in $\log O(K^2/f(n))$ bits.



Hence we can describe $x_1, \ldots, x_n$ by specifying

- this encoding scheme in $O(1)$ bits;
- the locations of $x_i, x_j, x_k$ in $6 \log K - \log f(n)$ bits;
- the distribution of the other $n-3$ points in $\log \binom{K^2}{n-3}$ bits.

This description is shorter than $\log \binom{K^2}{n} - \delta$ as in the previous lemma since $f(n) \gg n^3$. Q.E.D.

Since most distributions are $\delta$-incompressible,

$$\Delta_{expected} = \frac{\Omega(1/n^3)}{2} = \Omega(1/n^3)$$

## *Proving Upper Bound* $O(1/n^3)$

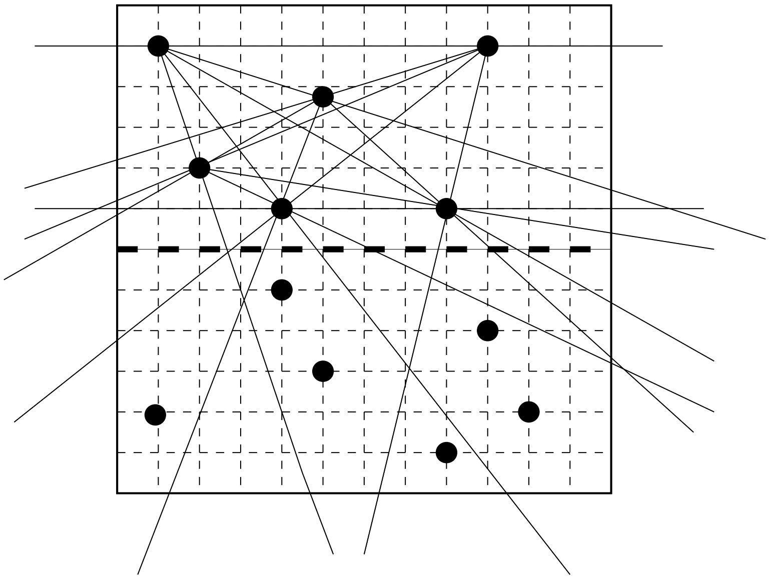
Let $f(n) < (2-\epsilon)\log n$ be a function. Fix an $f(n)$-incompressible distribution $\{x_1, \dots, x_n\}$.

$$C(x_1, \dots, x_n | n, K) \geq \log \binom{K^2}{n} - f(n)$$

**Lemma.** $\Delta(x_1, \dots, x_n) = O(f(n)/n^3)$.

Divide the unit square into upper and lower regions, each containing $n/2 \pm 2$ points.

Connect all $n/2$ points to form $n(n-2)/8$ *upper lines*.



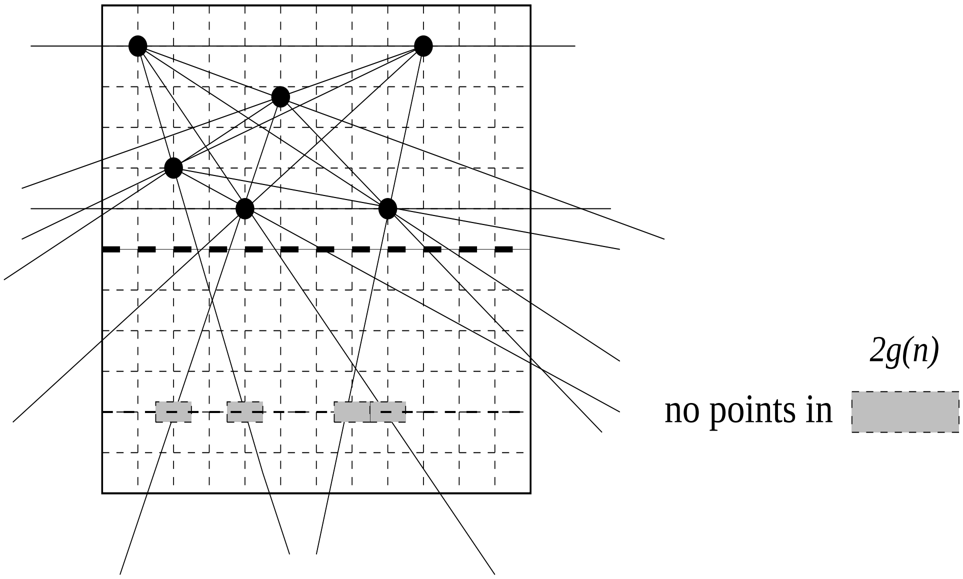
**Claim.** Every horizontal grid line in the lower region intersects $\Omega(n^2)$ upper lines.

# *Proving Upper Bound* $O(1/n^3)$

We can in fact strengthen the claim.

**Claim.** Every horizontal grid line in the lower region intersects $\Omega(n^2)$ upper lines which are *sufficiently far* from each other.

Now let $g(n) = \Delta(x_1, \ldots, x_n)$. Then each intersection eliminates $2g(n)K$ grid intersections that can be used to place the $n/2$ points in the lower region.



This allows us to give a compact description of the lower $n/2$ points.

## *Proving Upper Bound* $O(1/n^3)$

We describe the points $x_1, \ldots, x_n$ by specifying their $x$- and $y$-coordinates separately:

$$\text{WHY?}\quad \log\binom{K^2}{n} - \log\binom{K}{n} = n \log K \pm O(1)$$

- a description of this encoding scheme in $O(1)$ bits;

- a description of the $y$-coordinates in $\log\binom{K}{n}$ bits;

- a description of the $x$-coordinates of the points in the upper region, each using $\log K$ bits, in ascending order of their $y$-coordinates; and

- a description of the $x$-coordinates in the lower region, each using $\log K(1 - \Omega(n^2 g(n)))$ bits, in ascending order of their $y$-coordinates.

Altogether this is at most

$$\log\binom{K}{n} + n \log K + (n/2)\log(1 - \Omega(n^2 g(n))) + O(1)$$

Since this must be at least $\log\binom{K^2}{n} - f(n)$, we obtain

$$g(n) = O(f(n)/n^3)$$

## *Proving Upper Bound* $O(1/n^3)$

Finally, the expected upper bound is computed as:

$$\begin{aligned}\Delta_{expected} &= \sum_{f(n)=1}^{1.9 \log n} \frac{1}{2^{f(n)}} O(\frac{f(n)}{n^3}) + \frac{1}{n^{1.9}} O(\frac{1}{n^{1.142}}) \\ &= O(\frac{1}{n^3}),\end{aligned}$$

since $\Delta = O(1/n^{1.142})$ for all distributions.

Combining both lemmas,

$$\Delta_{expected} = \Theta(1/n^3)$$

**Note.** The above constructions are only rough sketches. For more detailed (and accurate) constructions, see our paper in RSA.

# k One-way Heads Cannot Do String Matching

**(A lower bound or impossibility result by the incompressibility method.)**

Joint work with M. Li.

**String Matching (or Pattern Matching):**
Given pattern $x$ and text $y$, decide if $x$ occurs in $y$.

Sometimes we also want to locate the occurrences of $x$ in $y$.

1. One of the most important problems in computer science.

2. Hundreds of papers written.

3. Many efficient algorithms found.
   *E.g.*, KMP, BM, KR.

**Main features of the algorithms:**

1. Linear time.

2. Constant space
   (*i.e.*, multihead finite automaton).

   A two-way six-head finite automaton can do string matching in linear time (Galil and Seiferas, 1981)

   Note: KMP and BM are not constant-space.

3. No need to back up pointer in the text (*e.g.*, KMP).

**Question 1:** Can we do string matching without backing up the pointers at all?

Note that, the question makes sense only in comparison-based model.

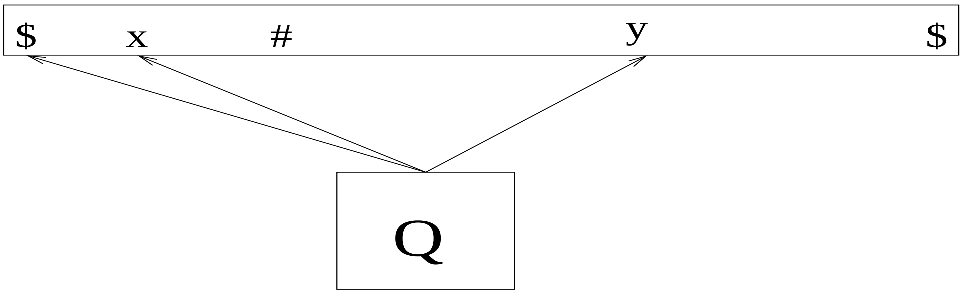
**Question 2:** (Galil and Seiferas, 1981)
Can a *one-way* $k$-head finite automaton do string matching, for any $k$?

**Previous results.** (Geréb-Graus, Li, Yesha)
Negative for $k = 2, 3$.

**Our result.** Negative for any $k$.

## Model of Computation

Fix a deterministic finite automaton $M = \ < \{0,1\}, Q, \delta, q_0, F >$ with $k$ one-way read-only heads ($k$-DFA).



$M$ *accepts* an input $\$x\#y\$$ if $x$ appears in $y$.

Assume the heads are *non-sensing*,
*i.e.*, $M$ cannot detect if two heads meet.

## A Bit of Kolmogorov Complexity

The Kolmogorov complexity (KC) of $x$, $K(x)$, is the size (in bits) of the smallest description of $x$.

A binary string $x$ is *random* if

$$K(x) \geq |x| - O(\log |x|)$$

KC of $x$ *conditional to* $y$, $K(x|y)$, is the size of the smallest description of $x$, given $y$.

$x$ is *random relative to* $y$ if

$$K(x|y) \geq |x| - O(\log |x|)$$

*Fact 1.* Most strings of a fixed length are random.

*Fact 2.* If string $x = uvw$ is random, then

$$K(v|uw) \geq |v| - O(\log |x|)$$

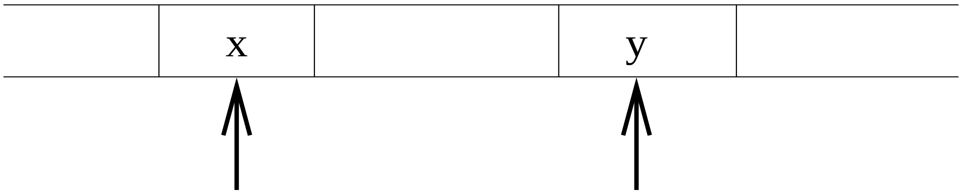
*Fact 3.* (Symmetry of Information)
Up to a logarithmic additive term,

$$K(x) - K(x|y) = K(y) - K(y|x)$$

$$K(x) + K(y|x) = K(y) + K(x|y)$$

## Some Useful Lemmas

**Def.** $x, y$ are two segments of the input. $M$ *matches* $x$ and $y$ if, at some time a head moves a step in $x$ while another head is in $y$ and vice versa.



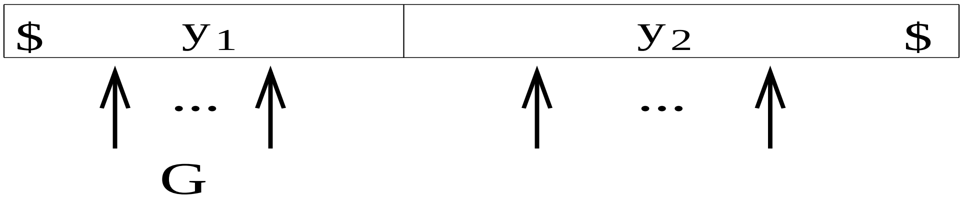
**Matching Lemma.** (Li and Yesha)
Let $M$ accept $I = \$x\#y\$$. Suppose

$$K(x|y-x) \geq |x| - \log|I|$$

Then $M$ must match the pattern $x$ with some occurrence of $x$ in $y$.

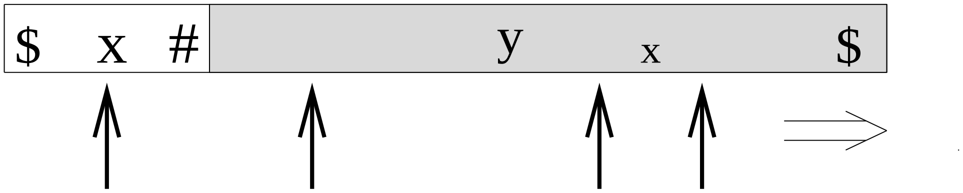
Otherwise $M$ will accept a wrong input with all occurrences of $x$ in $y$ being replaced by some non-random string.

**Def.** Let $G$ be a set of heads, and \$$y_1 y_2$\$ an input. Suppose the leading head is at the last bit of $y_1$. $M$ is *conscious of* $G$ if there exists a $y_2$ that can make some head in $G$ move before any head reaches the right endmarker \$.



Otherwise, $M$ is *unconscious of* $G$. That is, some heads will blindly move to the right.

**Moving Lemma.** Let $\alpha > 0$ be any constant and $\$x\#y\$$ an input, where $y$ is a sufficiently long random string. If some heads move $|y|^\alpha$ steps in $y$ while the others remain completely stationary, these heads will move unconsciously of the others until some reaches the right endmarker \$.



*I.e.*, a random text is like a desert: if some heads move too much in it, they will get lost and move blindly to the right.

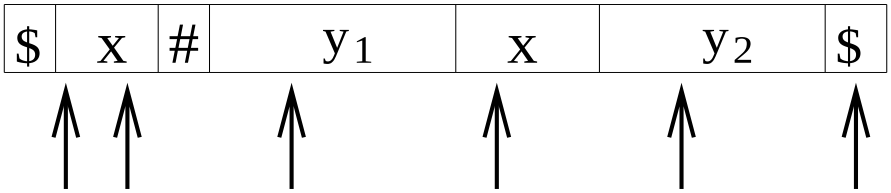
**Theorem.** $M$ cannot do string matching.

Input: \$$x\#y_1xy_2$\$, where $x, y_1$ are random and $|y_1| > |x|^3$.

Either all heads shift out of $x$ in $k|x|^2$ steps or Moving Lemma will apply (with $\alpha = 1/3$)!

The former case is prohibited by Matching Lemma. Thus, some heads will move blindly to the right.

The head reaches \$ first becomes *dead*.



What if some moving heads are near the copy of $x$?

## A More Sophisticated Construction.

Choose a large $n$ and

$$l = 1 + \sum_{i=1}^{k-1} i = (k^2 - k + 2)/2$$

Fix a random string $z = xy_1 \cdots y_l$, where $|x| = n$, $|y_i| = n^{3i} - n^{3i-3}$.

Let $y = y_1 x y_2 x \cdots y_l$. Simulate $M$ on input \$$x\#y$\$ until a head dies. At most $k-1$ copies of $x$ are near some head.

Find an $x'$ relatively random to $x, y$ and, replace the $k-1$ copies of $x$ with $x'$ to get $y'$. Make sure $M$ ends up in the same configuration on \$$x\#y'$\$.

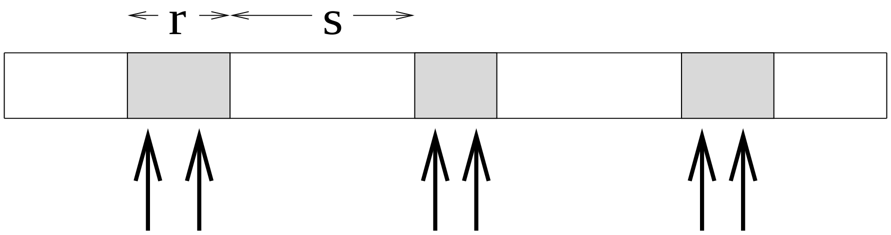
Repeat above until $k-1$ heads have died.

## Proof of Moving Lemma

Want: If some heads move $|y|^\alpha$ steps in $y$ *consciously* while the others remain still, we can compress $y$ by at least $|y|^\epsilon$ bits, for $\epsilon > 0$ depending only on $\alpha$ and $M$.

**Def.** An $(r, s)$-grouping is a partition of heads into groups such that

1. heads in each group are within $r$ bits,
2. groups are separated by at least $s$ bits.

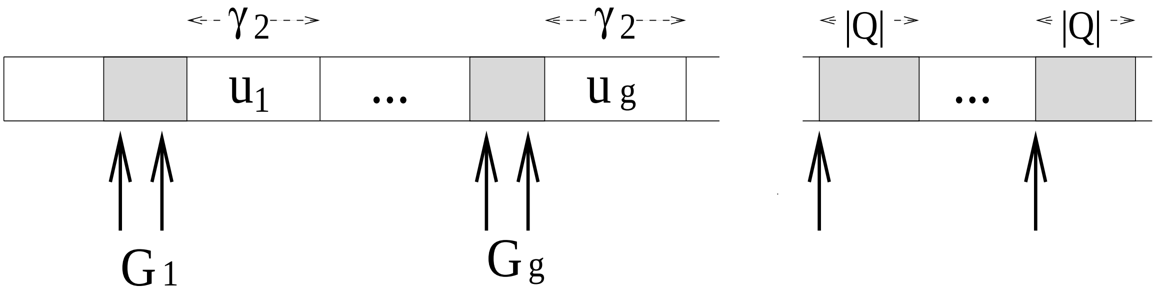


**Local Compression Lemma.**
For every $\gamma_1$, there is $\gamma_2$ validating next claim:

**Claim.** Suppose $G = (G_1, \ldots, G_g)$ is a $(\gamma_1, \gamma_2)$-grouping of some heads at time $t$. Let $u_i$ be the substring of length $\gamma_2$ to the right of $G_i$. If the heads not in $G$ remain stationary till a head in some $G_i$ finishes $u_i$, then

$$u_1 \cdots u_g \in S$$

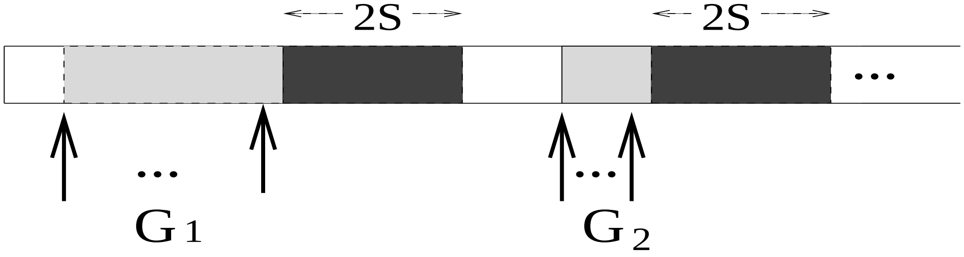
for some $S$ of size at most $2^{g\gamma_2} - 1$, depending only on the configuration of $M$ at time $t$.



Roughly, we can predict $u_1 \cdots u_g$ and compress it by $g\gamma_2 - \log(2^{g\gamma_2} - 1)$ bits.

If $|G_i| = 1$ for all $i$, $\gamma_1 = 1$ and $\gamma_2 = |Q|$.

**Great Compression Lemma.**
Consider a grouping $G = (G_1, \ldots, G_g)$. Suppose the groups are at least $2s$ bits apart, for some $s \geq \phi(G)$. If the heads not in $G$ remain stationary in the next $s$ steps, we can compress the portions of input scanned by the heads in $G$ during the $s$ steps by at least $s^{\epsilon(G)}$ bits, given initial positions of the heads.

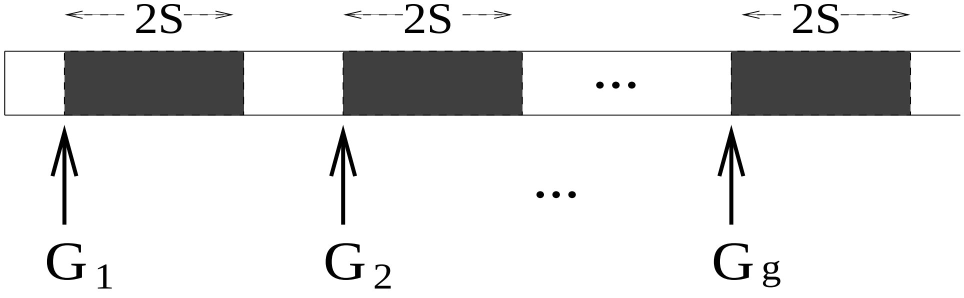


The proof is by induction on $l(G)$, where

$$l(G) = \sum_{i=1}^{g} |G_i|^2 (\sum_{i=1}^{g} |G_i|^2 - |G_i|)$$

Note, $l(G)$ decreases if some $G_i$ decreases in size or splits into more smaller groups.

**Claim.** The lemma holds when $l(G) = 0$.



Proof by induction on $|G| = g$.

*Case 1:* During the next $s$ steps, some head $h \in G$ pauses for $s^{1/2}$ steps.

Consider the $g - 1$ moving heads for $s^{1/2}$ steps.

*Case 2:* No head pause for $s^{1/2}$ steps.

Do local compression (with $\gamma_1 = 1, \gamma_2 = |Q|$) $s^{1/2}/(g|Q|)$ times to disjoint sections.
(once every $g|Q|s^{1/2}$ steps)

**Inductive Step.**

*Case 1:* Some $G_i$ pauses for $s^{1/3}$ steps during the next $s$ steps.

*Case 2:* Some $G_i$ splits, *i.e.*, at a time two adjacent heads in $G_i$ are $2s^{1/3}$ bits apart.

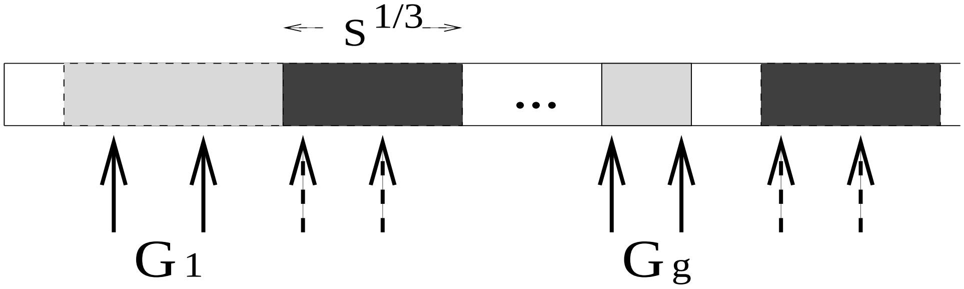
*Case 3:* Neither Case 1 nor Case 2.

*Case 3:* Every group moves at least once in every $s^{1/3}$ steps and no group split.

The heads in $G_i$ are always within distance

$$2|G_i|s^{1/3} \leq 2ks^{1/3}$$

Dynamically partition input into *sections* of at most $s^{1/3}$ bits, and compress each
$g$-tuple of sections by at least 1 bit.

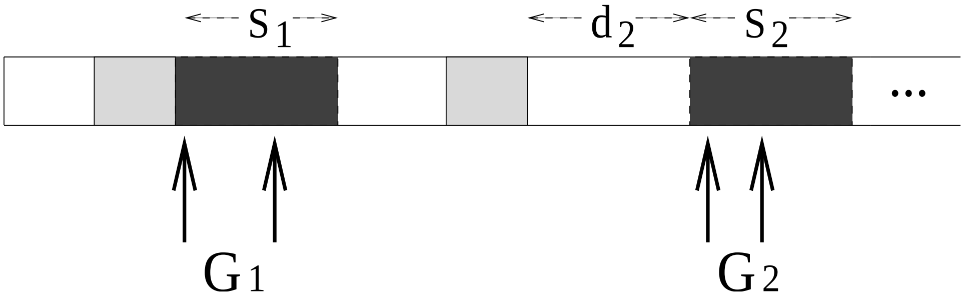


We can compress at least $s^{1/3}$ $g$-tuples.

## Two Subcases of A g-tuple

*Subcase 1:* Some $G_i$ split by a large constant.

Record head positions and compress recursively for $s_1$ steps.



How to compensate for the overhead

$$\log s_1 + \log d_2 + \log s_2?$$

*Subcase 2:* Heads in each $G_i$ are within a constant distance.

**Open problem:** What if the heads are sensing?

The current Moving Lemma doesn’t hold since with 7 heads one can easily make some heads moving consciously for a long time.

## Concluding Remarks

1. We have demonstrated the usefulness and simplicity of the incompressibility method.

2. It seems that the method has advantage over other methods (such as the probabilistic method) especially when dealing with algorithmic problems.

3. We expect more applications to come.

4. Can Shellsort achieve $O(n \log n)$ on the average?

5. Give a simple analysis of Quicksort by incompressibility.

# References

1. M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 2nd Edition, 1997.

2. T. Jiang and M. Li. k one-way heads cannot do string-matching. *Journal of Computer and System Sciences*, 1996.

3. T. Jiang, J. Seiferas and P. Vitanyi. Two heads are better than two tapes. *Journal of the ACM*, 1997.

4. T. Jiang, M. Li, and P. Vitanyi. New applications of the incompressibility method. *Computer Journal*, 1999.

5. T. Jiang, M. Li and P. Vitanyi. A lower bound on the average-case complexity of Shellsort. *Journal of the ACM*, 2000.

6. H. Buhrman, T. Jiang, M. Li, and P. Vitanyi. New applications of the incompressibility method: Part II. *Theoretical Computer Science*, 2000.

7. T. Jiang, M. Li and P. Vitanyi. Average-case analysis of algorithms using Kolmogorov complexity. *Journal of Computer Science and Technology*, 2000.

8. T. Jiang, M. Li and P. Vitanyi. The average-case area of Heilbronn-type triangles. *Random Structures and Algorithms*, 2002.