Analysis of Algorithms: Towards a More Realistic Model

Habilitation à Diriger des Recherches

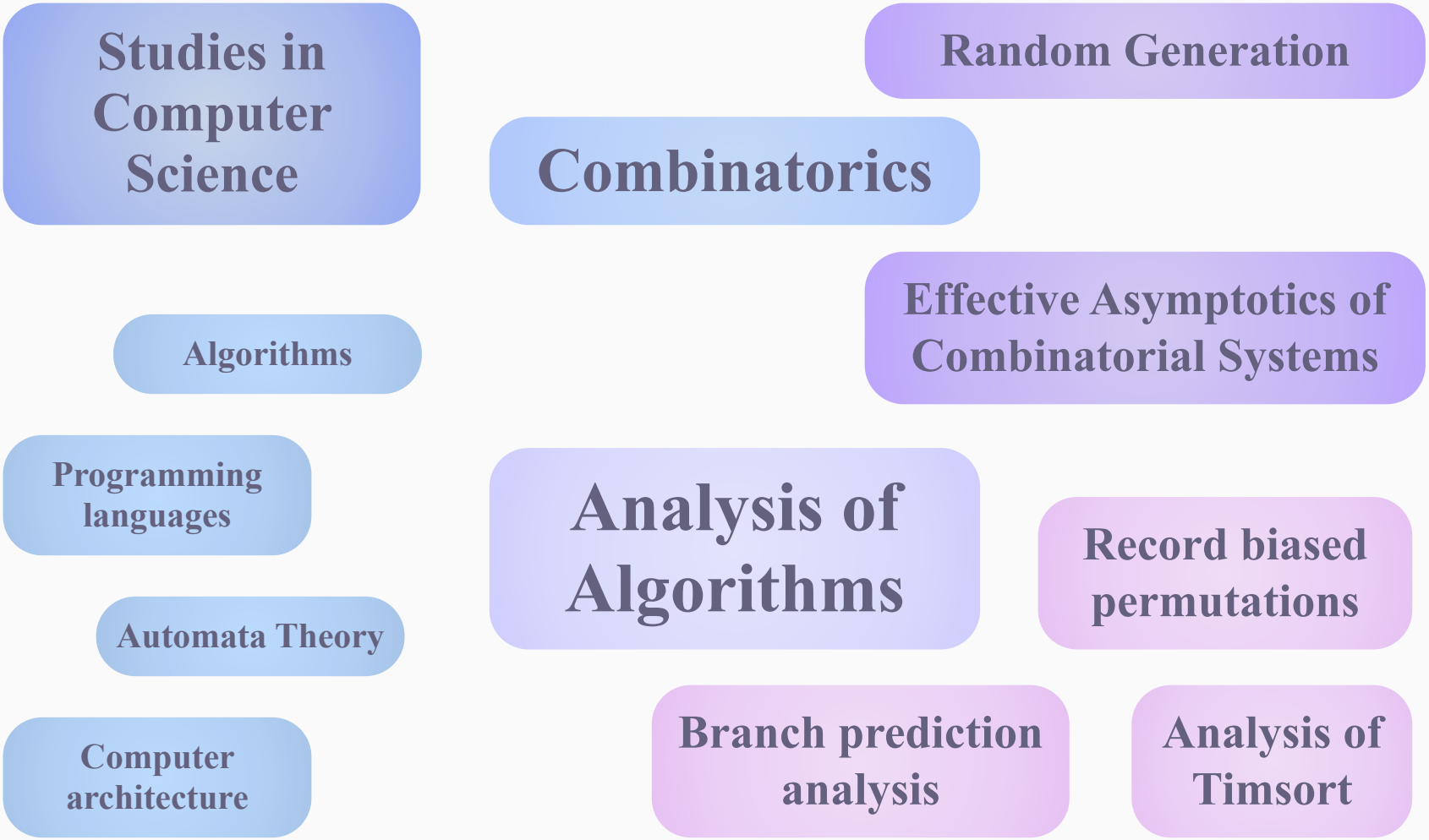
Carine Pivoteau

January 26, 2026

LIGM - Université Gustave Eiffel

Introduction

Background and overview



Analysis of algorithms: a more realistic model

Analysis of algorithms aims to estimate the number of elementary operations required to process an input of a specific size.

Analysis of algorithms: a more realistic model

Analysis of algorithms aims to estimate the number of elementary operations required to process an input of a specific size.

Worst-case analysis provides upper bounds on running time.

- ▷ However, it does not necessarily reflect practical performance.

Analysis of algorithms: a more realistic model

Analysis of algorithms aims to estimate the number of elementary operations required to process an input of a specific size.

Worst-case analysis provides upper bounds on running time.

- ▷ However, it does not necessarily reflect practical performance.

Average-case analysis offers insight into *typical* behavior.

- ▷ Historically, we assumes uniformly distributed random inputs and unit-cost operations within the RAM model.

Analysis of algorithms: a more realistic model

Analysis of algorithms aims to estimate the number of elementary operations required to process an input of a specific size.

Worst-case analysis provides upper bounds on running time.

- ▷ However, it does not necessarily reflect practical performance.

Average-case analysis offers insight into *typical* behavior.

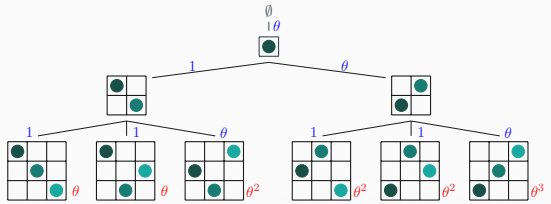
- ▷ Historically, we assumes uniformly distributed random inputs and unit-cost operations within the RAM model.

Towards a more realistic model: examine *real-world* implementations to better align theory with practice.

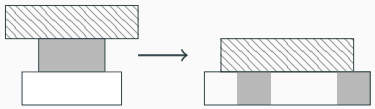
- ▷ Consider non-uniform random distributions for inputs.
- ▷ Refine the analysis with other parameters.
- ▷ Incorporate hardware-aware cost measures.

Real world data and implementations

- Record biased permutations (AofA '16, CPC 2025)
  - generative processes and random samplers
  - full combinatorial study
  - permuton limit



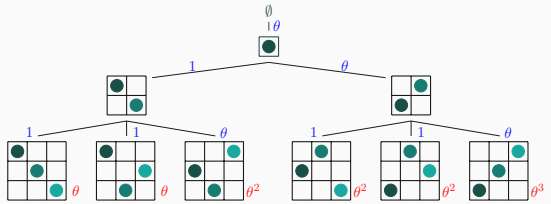
- Additional parameters: complexity of TimSort (ESA '18)
  - algorithm designed to perform on almost sorted inputs
  - worst case complexity: $\mathcal{O}(n \log n)$
  - with a refined parameter: $\mathcal{O}(n\mathcal{H} + n)$



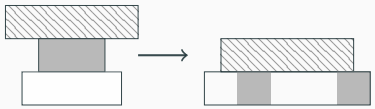
Results

Real world data and implementations

- Record biased permutations (AofA '16, CPC 2025)
  - generative processes and random samplers
  - full combinatorial study
  - permuton limit

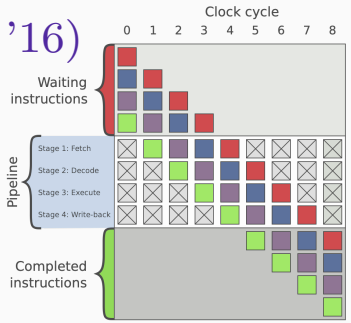


- Additional parameters: complexity of TimSort (ESA '18)
  - algorithm designed to perform on almost sorted inputs
  - worst case complexity: $\mathcal{O}(n \log n)$
  - with a refined parameter: $\mathcal{O}(n\mathcal{H} + n)$



Enhancing the model with branch prediction analysis

- Analysis of algorithms with surprising trade-offs (STACS '16)
- Average case analysis of pattern matching algorithms
  - the sliding window algorithm
  - Morris-Pratt and Knuth-Morris-Pratt (CPM '25)



Real World Data and Implementations

Record biased permutations (with N. Auger, M. Bouvel and C. Nicaud)

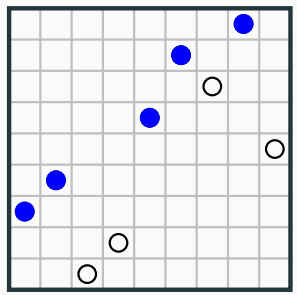
Goal: non-uniform models with good balance between **simplicity**, so that we can study it, and **accuracy**, to model realistic data.

Ewens model (1972): $\mathbb{P}(\sigma)$ is proportional to $\theta^{\text{number of cycles of } \sigma}$

Definition

A record is an element larger than all those preceding it.

For example: **3** **4** 1 2 **6** **8** 7 **9** 5 has 5 records.

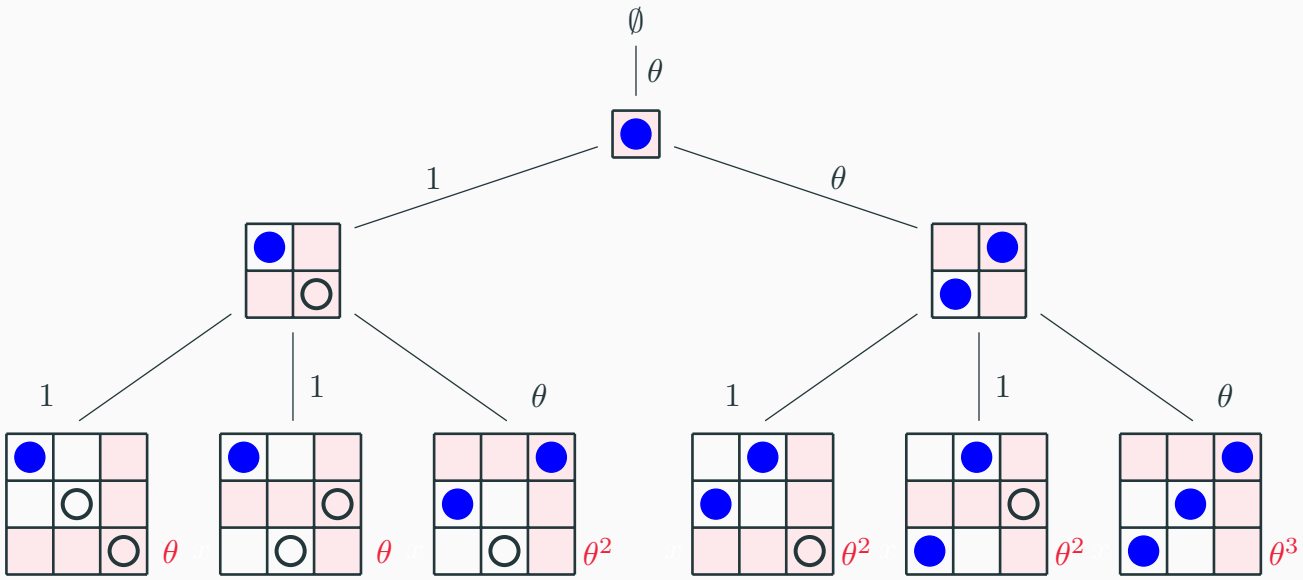


$$\mathbb{P}(\sigma) = \frac{\theta^{\mathrm{rec}(\sigma)}}{\sum_{\rho \in \mathfrak{S}_n} \theta^{\mathrm{rec}(\rho)}} = \frac{\theta^{\mathrm{rec}(\sigma)}}{\theta^{(n)}},$$

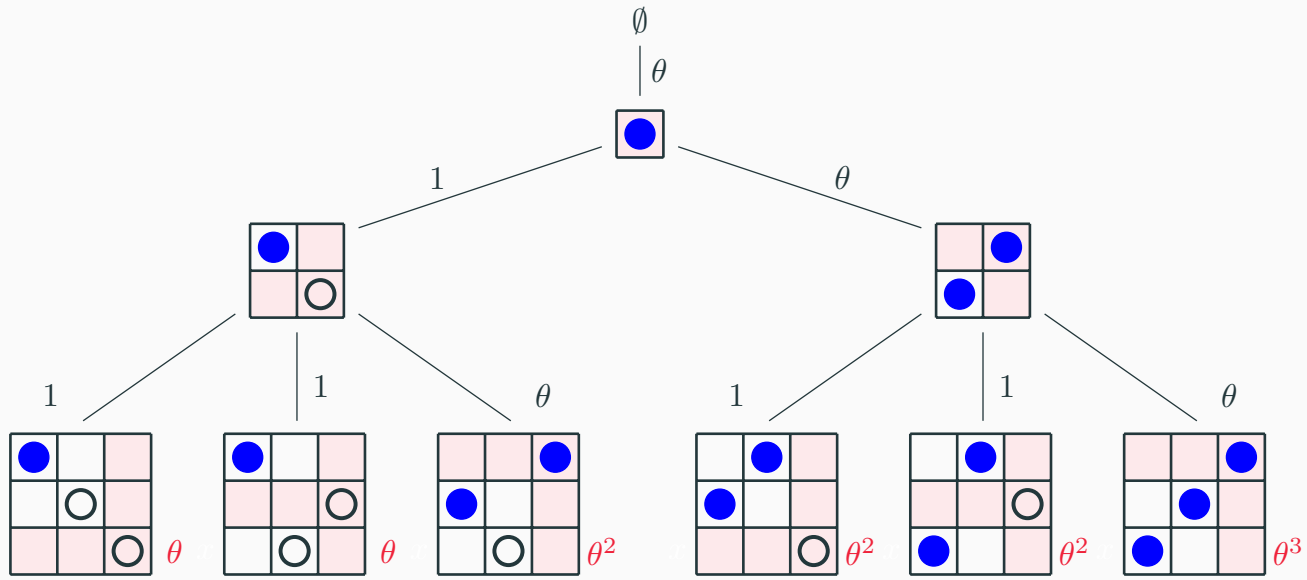
where $\mathrm{rec}(\sigma)$ is the number of records of σ and $\theta^{(n)} = \theta(\theta+1)\dots(\theta+n-1)$.

For $\theta > 1$, higher probabilities for "almost sorted" permutations.

Generative processes and random samplers



Generative processes and random samplers



- Different generative processes for other representations.
- Easy access to various statistics.
- Each step can be performed in $\mathcal{O}(1)$ time using appropriate data structures, which yields a linear random sampler.

Four statistics: expectation and distribution

- Number of records, number of descents, number of inversions, first value.
- Formulas for their expectation, when θ is fixed or $\theta = \theta(n)$.
- Asymptotic behavior of these expectations when θ is fixed (for various regimes).

	$\theta = 1$ uniform	fixed $\theta > 0$	$\theta = n^\varepsilon$, $0 < \varepsilon < 1$	$\theta = \lambda n$, $\lambda > 0$	$\theta = n^\delta$ $\delta > 1$
$\mathbb{E}_n[rec]$	$\boldsymbol{\log n}$	$\theta \log n$	$(1-\varepsilon)n^\varepsilon \log n$	$\lambda \log(1+1/\lambda)n$	n
$\mathbb{E}_n[desc]$	$n/2$	$n/2$	$n/2$	$(\lambda+1)n/2$	$n^{2-\delta}/2$
$\mathbb{E}_n[inv]$	$n^2/4$	$n^2/4$	$n^2/4$	$f(\lambda)n^2/4$	$n^{3-\delta}/6$
$\mathbb{E}_n[\sigma(1)]$	$n/2$	$n/(\theta+1)$	$n^{1-\varepsilon}$	$(\lambda+1)/\lambda$	1

$$f(\lambda) = 1 - 2\lambda + 2\lambda^2 \log(1 + 1/\lambda)$$

Four statistics: expectation and distribution

- Number of records, number of descents, number of inversions, first value.
- Formulas for their expectation, when θ is fixed or $\theta = \theta(n)$.
- Asymptotic behavior of these expectations when θ is fixed (for various regimes).

Example of application:

InsertionSort complexity

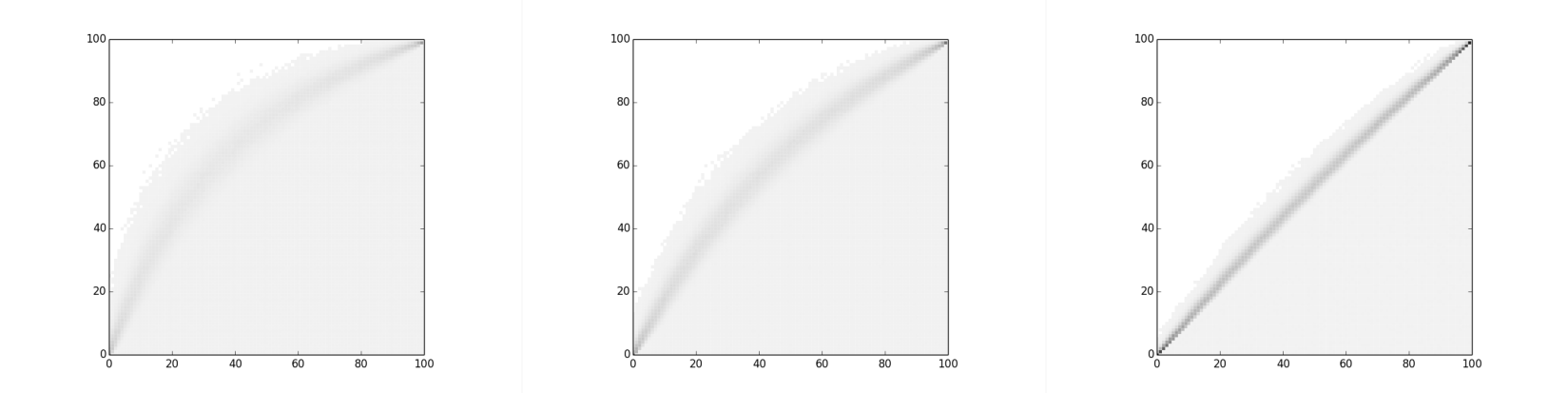
Applied to record-biased permutations of size n for the parameter $\theta = \mathcal{O}(n)$, the expected running time of InsertionSort is $\Theta(n^2)$.

If $\theta = n^\delta$ with $1 < \delta < 2$, it is $\Theta(n^{3-\delta})$. If $\theta = \Omega(n^2)$, it is $\Theta(n)$.

Limit shape of record biased permutations

Full characterization of the limit shape via the definition of a permuton (with M. Bouvel and C. Nicaud).

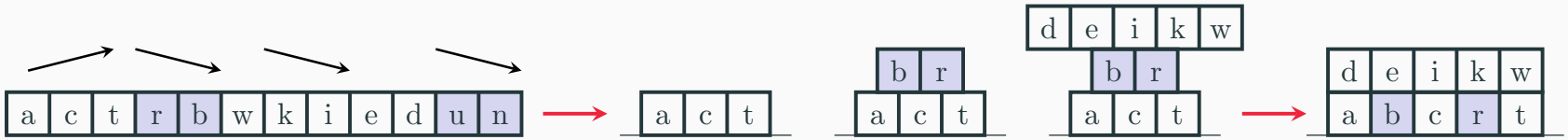
10 000 random permutations of size 100 stacked together, $\theta = \lambda n$.



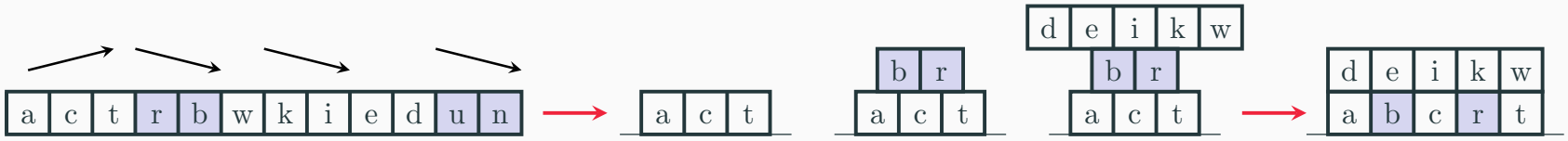
$\theta = \frac{n}{2}$ $\theta = n$ $\theta = 5n$

The darkness is proportional to the number of points at this position.

TimSort: variant of MergeSort, `Python` (2002), `Java` (2011).



TimSort: variant of MergeSort, `Python` (2002), `Java` (2011).

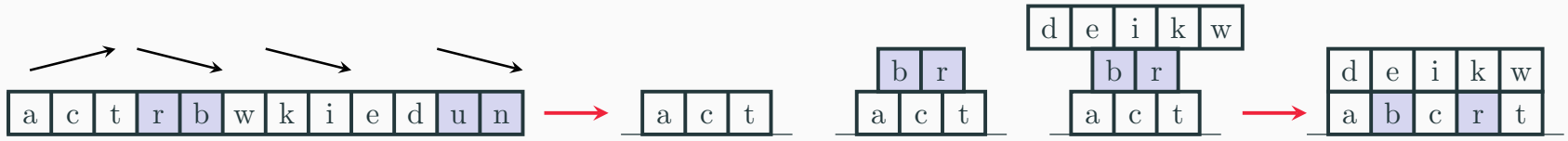


Main results

Proof of the $O(n \log n)$ worst-case complexity.

Refined analysis using the number of runs or the run length entropy: $\mathcal{O}(n \log \rho + n)$ and $\mathcal{O}(n\mathcal{H} + n)$, where $\mathcal{H} = -\sum_{i=1}^{\rho} r_i/n \log_2(r_i/n)$.

TimSort: variant of MergeSort, `Python` (2002), `Java` (2011).



Main results

Proof of the $O(n \log n)$ worst-case complexity.

Refined analysis using the number of runs or the run length entropy: $\mathcal{O}(n \log \rho + n)$ and $\mathcal{O}(n\mathcal{H} + n)$, where $\mathcal{H} = -\sum_{i=1}^{\rho} r_i/n \log_2(r_i/n)$.

Two successive bugs (fixed) in the `Java` Implementation.

Switch to PowerSort in `Python`, *Nearly-Optimal Mergesorts: Fast, Practical Sorting Methods That Optimally Adapt to Existing Runs.* J. I. Munro and S. Wild, ESA '18.

Further work

- **In progress:** Analysis of the run length entropy for record biased permutations.

Further work

- **In progress:** Analysis of the run length entropy for record biased permutations.

- *Galloping in fast-growth natural merge sorts.* E. Ghasemi, V. Jugé, G. Khalighinejad and H. Yazdanyar. Algorithmica 2025.

Further work

- **In progress:** Analysis of the run length entropy for record biased permutations.

- *Galloping in fast-growth natural merge sorts.* E. Ghasemi, V. Jugé, G. Khalighinejad and H. Yazdanyar. Algorithmica 2025.

- Investigating other algorithms:
  - ▷ PatternDefeatingQuicksort, implemented in `Rust` and `C++`.

Further work

- **In progress:** Analysis of the run length entropy for record biased permutations.

- *Galloping in fast-growth natural merge sorts.* E. Ghasemi, V. Jugé, G. Khalighinejad and H. Yazdanyar. Algorithmica 2025.

- Investigating other algorithms:
  - ▷ PatternDefeatingQuicksort, implemented in `Rust` and `C++`.
- Incorporating architecture considerations into the analysis:
  - ▷ cache behavior of HeapSort (see `Java`'s implementation of IntroSort for primitive arrays).

Branch Prediction Analysis

Analysis of algorithms and computer architecture

RAM model: The standard theoretical abstraction for analysis of algorithms. It is a simplified computer model that ignores the specifics of physical hardware.

Main characteristics:

- **Unit cost**: every elementary operation takes 1 unit of time.
- **Sequential**: instructions are executed one after another.

Analysis of algorithms and computer architecture

RAM model: The standard theoretical abstraction for analysis of algorithms. It is a simplified computer model that ignores the specifics of physical hardware.

Main characteristics:

- **Unit cost**: every elementary operation takes 1 unit of time.
- **Sequential**: instructions are executed one after another.

In modern architectures, two main features contradict this model: **cache memories** and **instruction parallelism**.

Analysis of algorithms and computer architecture

RAM model: The standard theoretical abstraction for analysis of algorithms. It is a simplified computer model that ignores the specifics of physical hardware.

Main characteristics:

- **Unit cost**: every elementary operation takes 1 unit of time.
- **Sequential**: instructions are executed one after another.

In modern architectures, two main features contradict this model: **cache memories** and **instruction parallelism**.

Despite these simplifications, an algorithm that is faster in the RAM model is **almost always** faster on real hardware for large inputs.

- ▷ Simple, robust, reliable model.
- ▷ Architectural optimizations can yield **significant speedup**.
- ▷ Theoretical models can be **refined** to incorporate these features.

A case study: simultaneous min/max (with N. Auger and C. Nicaud)

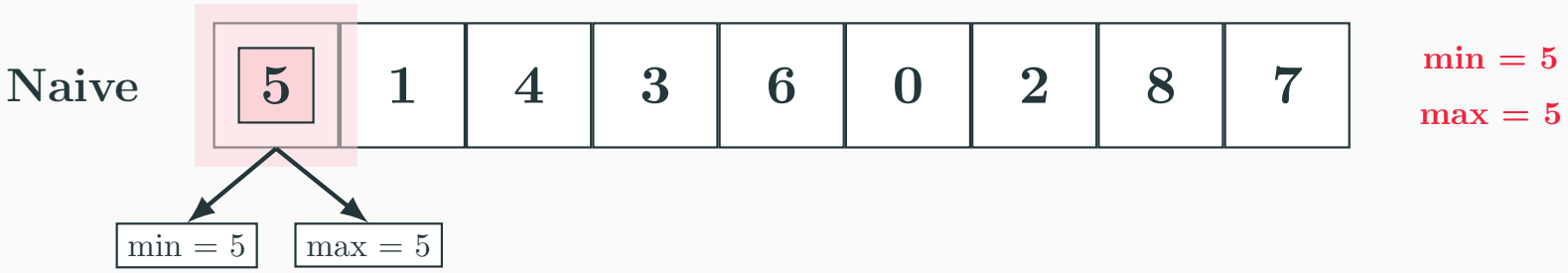
Naive

5	1	4	3	6	0	2	8	7

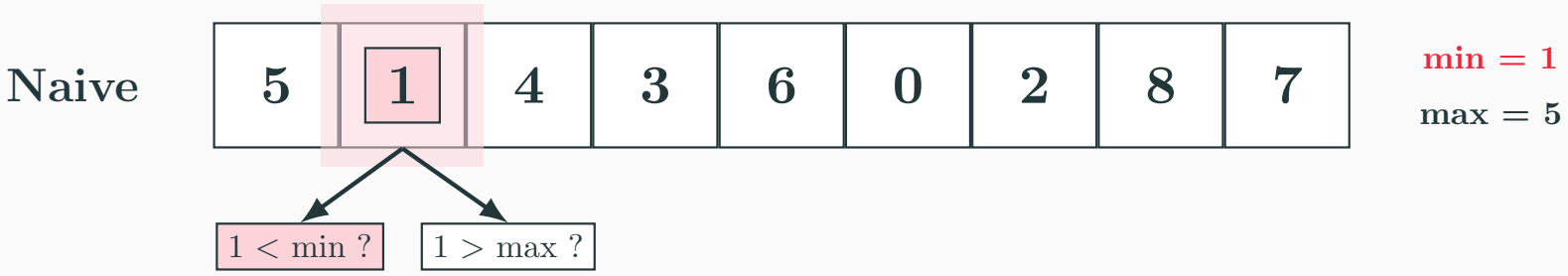
$\min = ?$

$\max = ?$

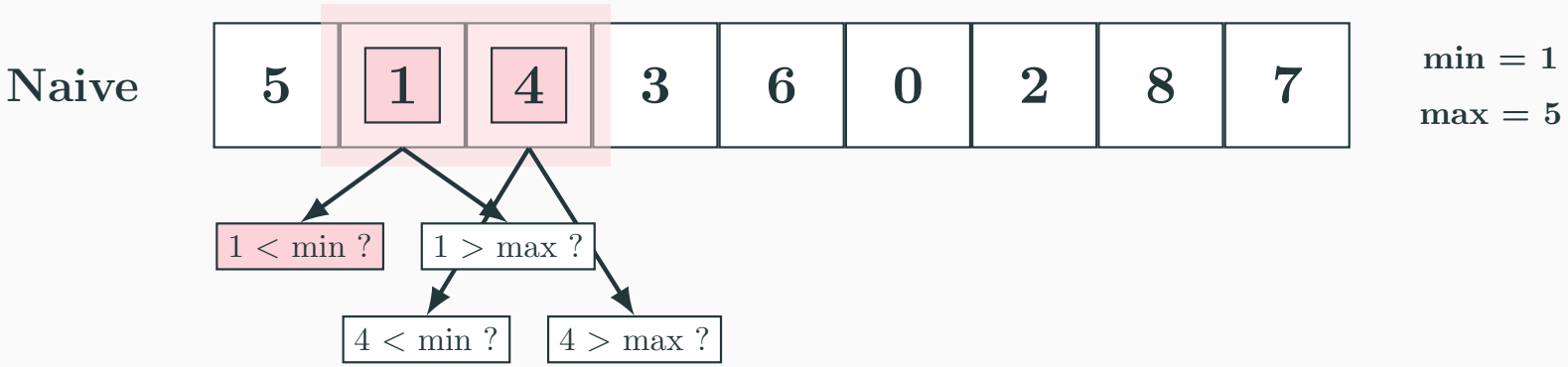
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



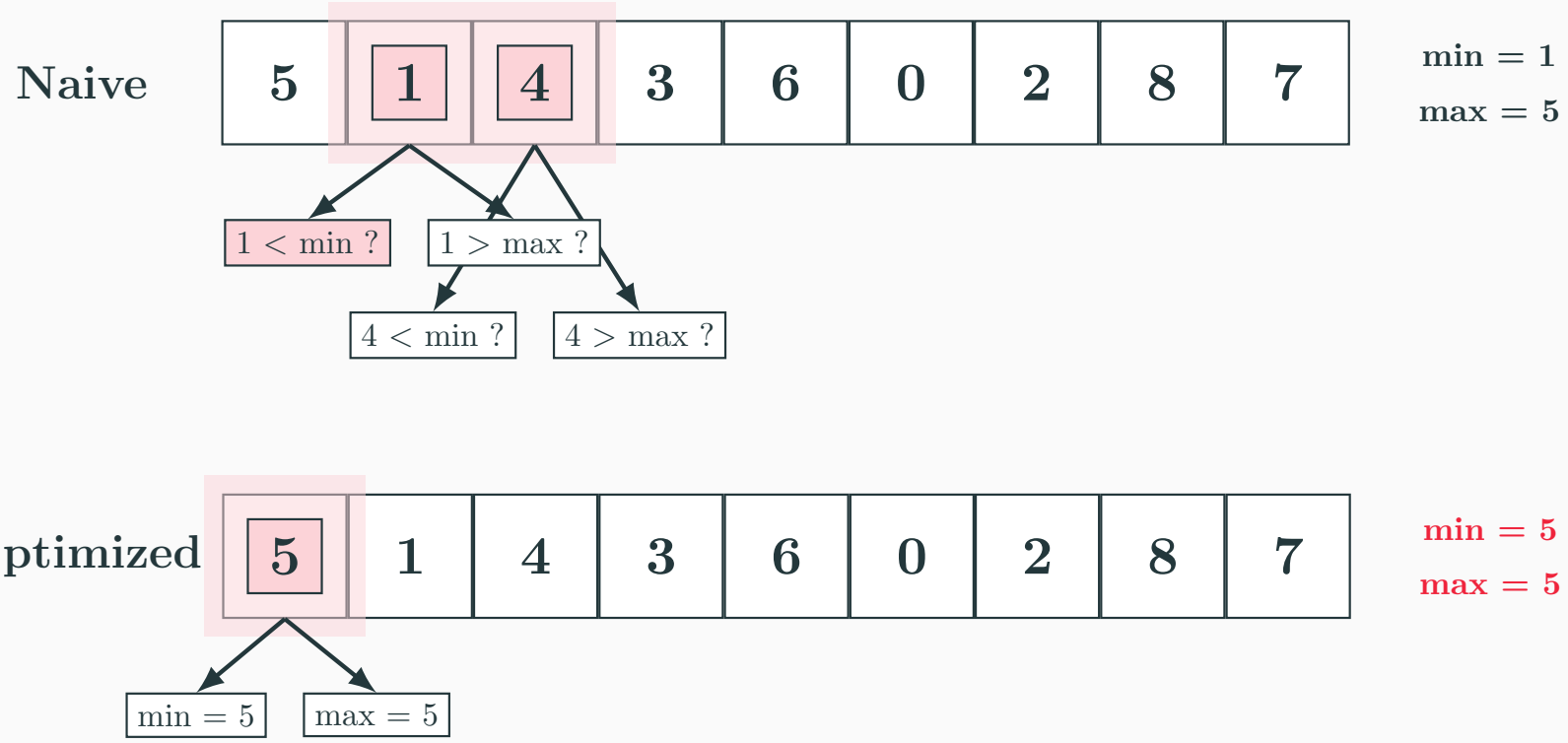
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



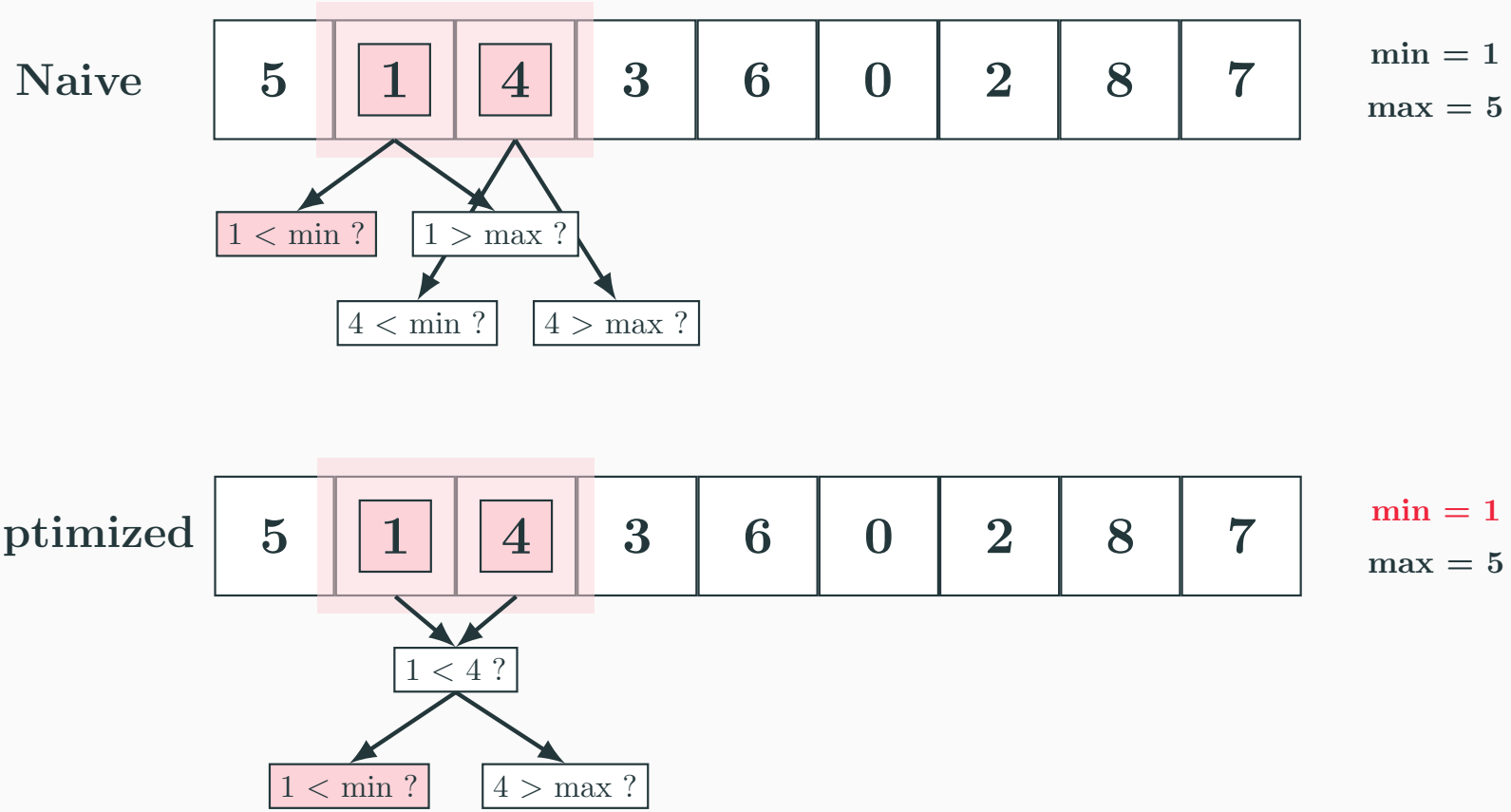
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



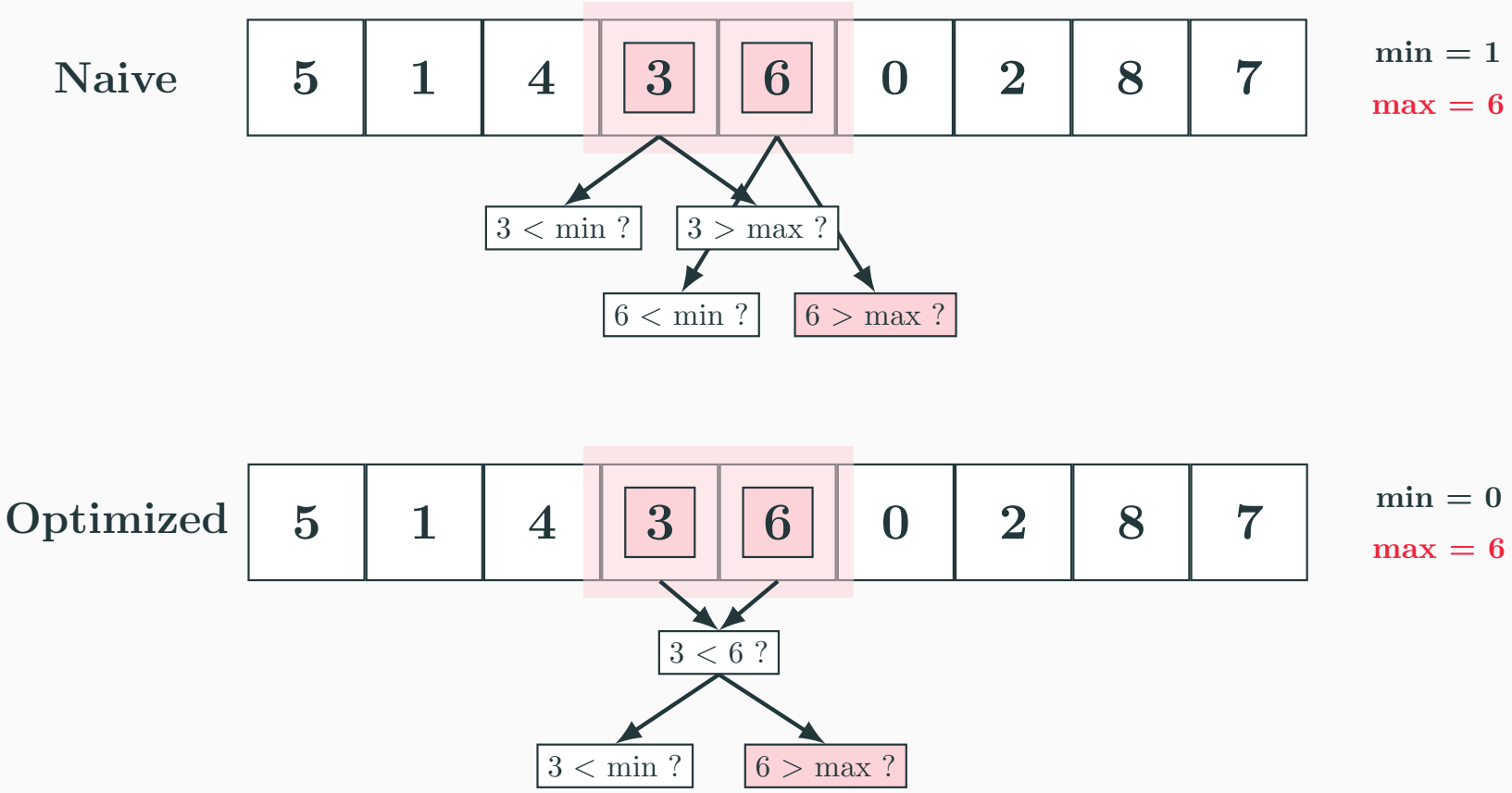
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



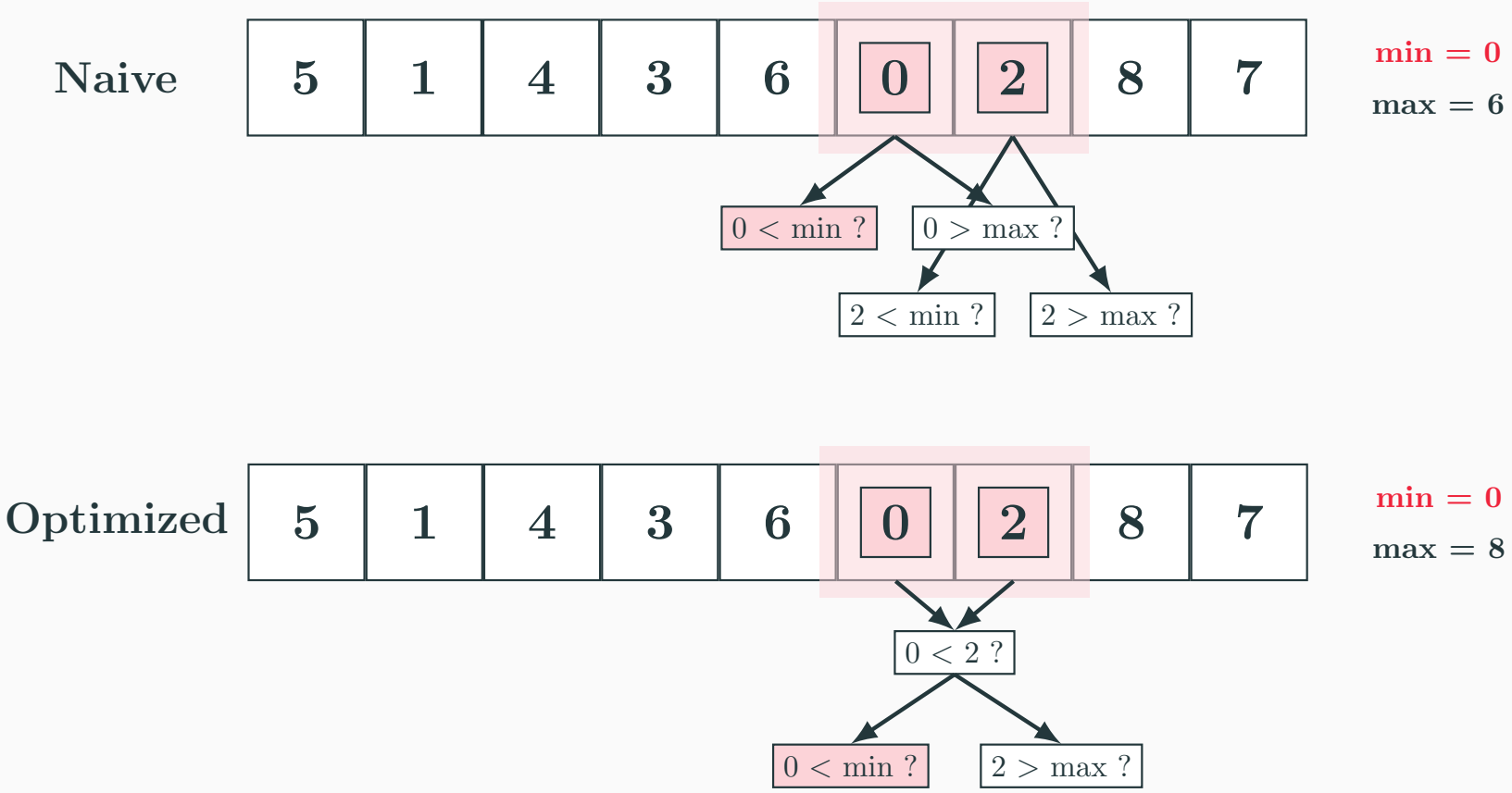
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



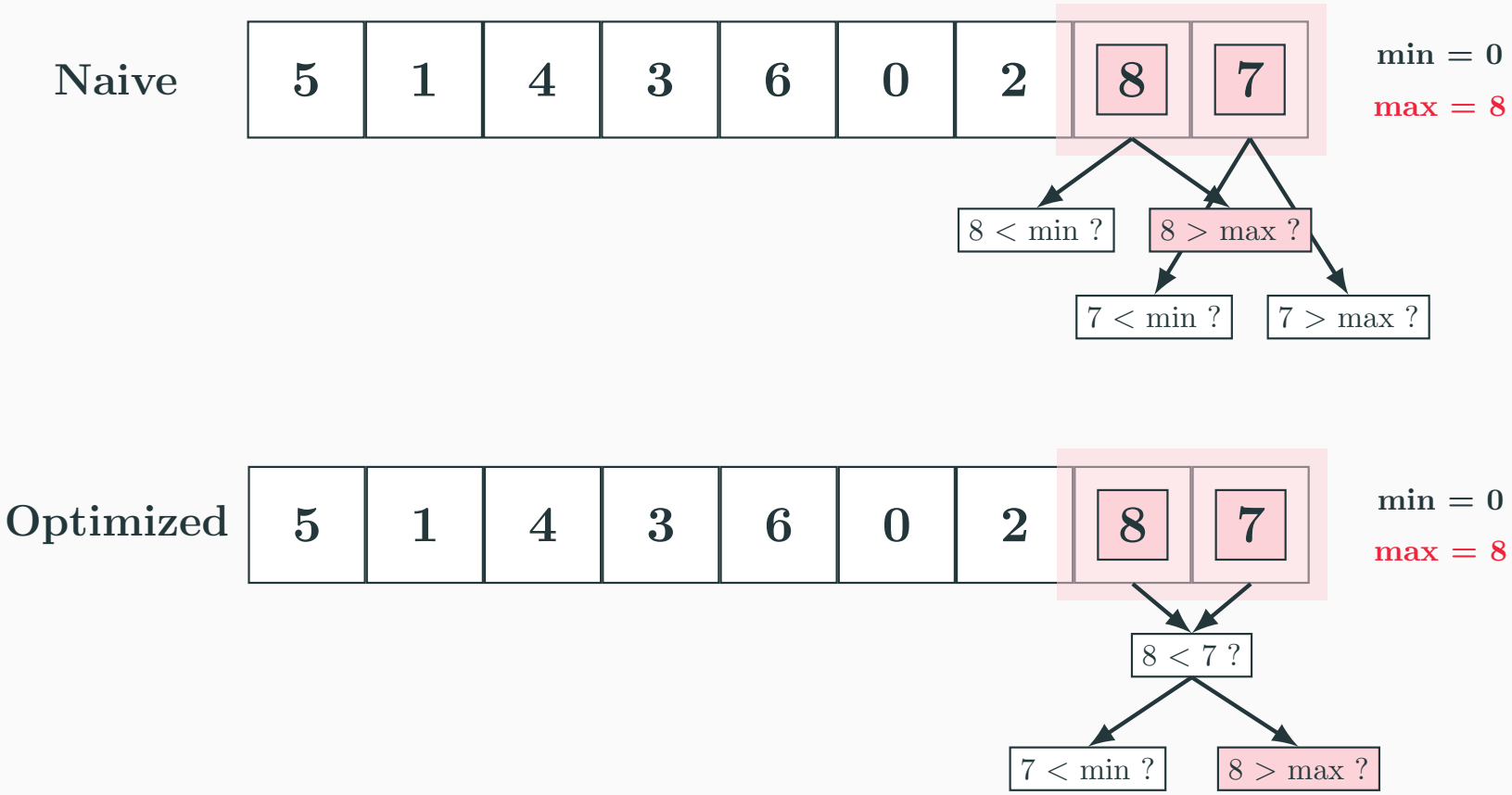
A case study: simultaneous min/max (with N. Auger and C. Nicaud)



A case study: simultaneous min/max (with N. Auger and C. Nicaud)



A case study: simultaneous min/max (with N. Auger and C. Nicaud)

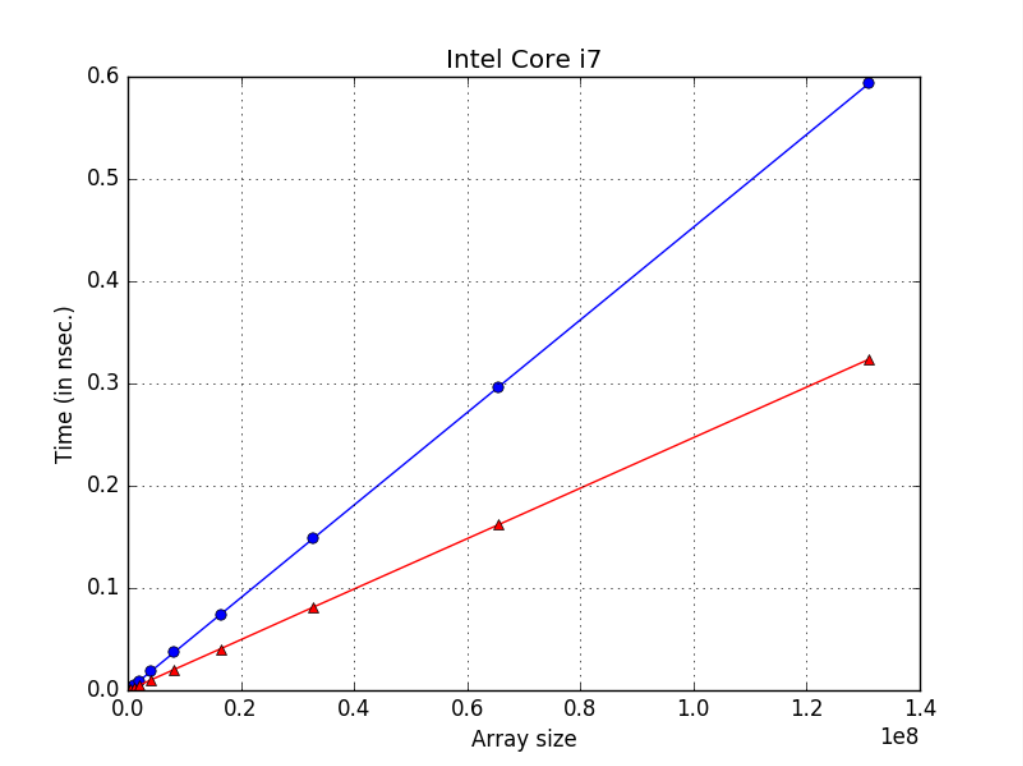


Naive algorithm: $2n$ comparisons

Optimized algorithm: $3n/2$ comparisons (optimal)

In practice, on uniform random data?

A case study: simultaneous min/max (with N. Auger and C. Nicaud)



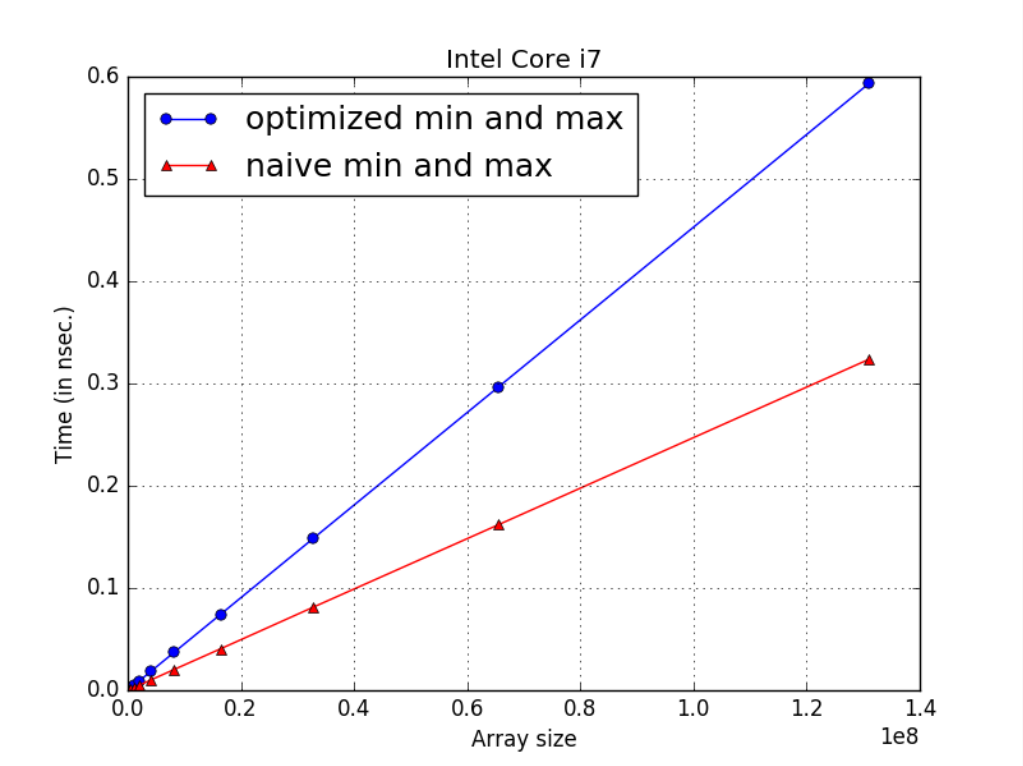
- code in C,
- with `gcc -O0`,
- random floats in $[0, 1]$

Naive algorithm: $2n$ comparisons

Optimized algorithm: $3n/2$ comparisons (optimal)

In practice, on uniform random data?

A case study: simultaneous min/max (with N. Auger and C. Nicaud)



- code in C,
- with `gcc -O0`,
- random floats in $[0, 1]$

Naive algorithm: $2n$ comparisons

Optimized algorithm: $3n/2$ comparisons (optimal)

In practice, on uniform random data?

Pipeline, hazards, and branch prediction

- (Modern) processors use a pipeline to create instruction-level parallelism.

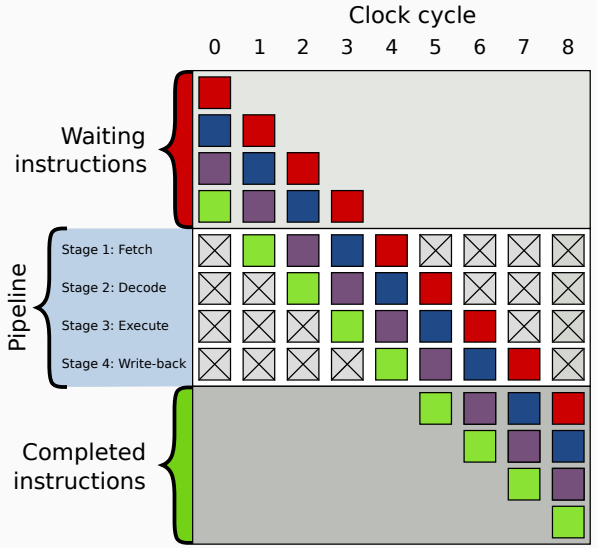


Illustration: Wikipedia

Computer Architecture: A Quantitative Approach (5th ed.), Hennessy & Patterson

Pipeline, hazards, and branch prediction

- (Modern) processors use a pipeline to create instruction-level parallelism.
- Hazards can stall a pipeline: branching instructions (`if`, `while`, ...) often require the preceding one to complete.

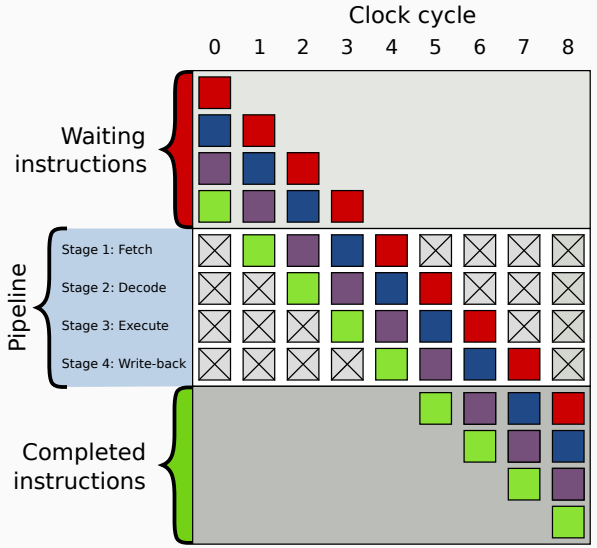


Illustration: Wikipedia

Computer Architecture: A Quantitative Approach (5th ed.), Hennessy & Patterson

Pipeline, hazards, and branch prediction

- (Modern) processors use a pipeline to create instruction-level parallelism.
- Hazards can stall a pipeline: branching instructions (`if`, `while`, ...) often require the preceding one to complete.
- Solution: anticipating the outcome of a branch instruction by using a predictor.

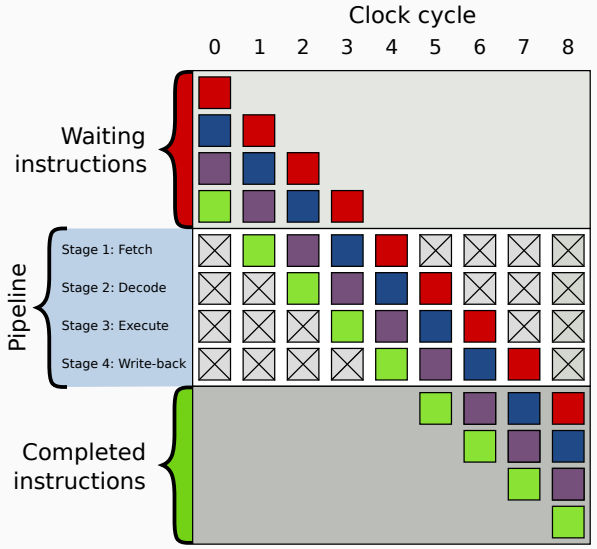


Illustration: Wikipedia

Computer Architecture: A Quantitative Approach (5th ed.), Hennessy & Patterson

Pipeline, hazards, and branch prediction

- (Modern) processors use a pipeline to create instruction-level parallelism.
- Hazards can stall a pipeline: branching instructions (`if`, `while`, ...) often require the preceding one to complete.
- Solution: anticipating the outcome of a branch instruction by using a predictor.

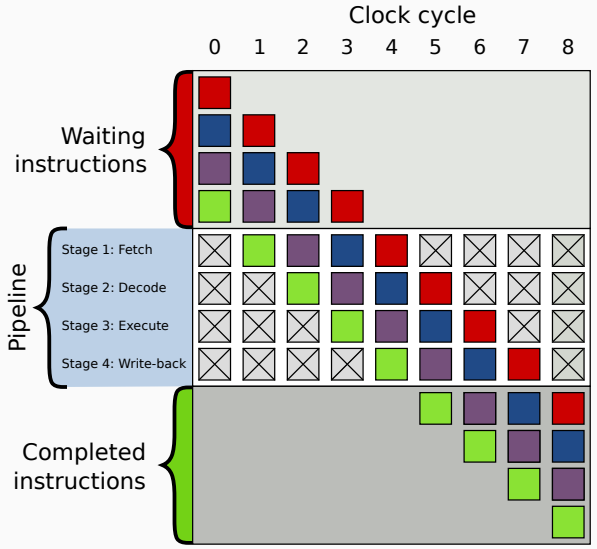


Illustration: Wikipedia

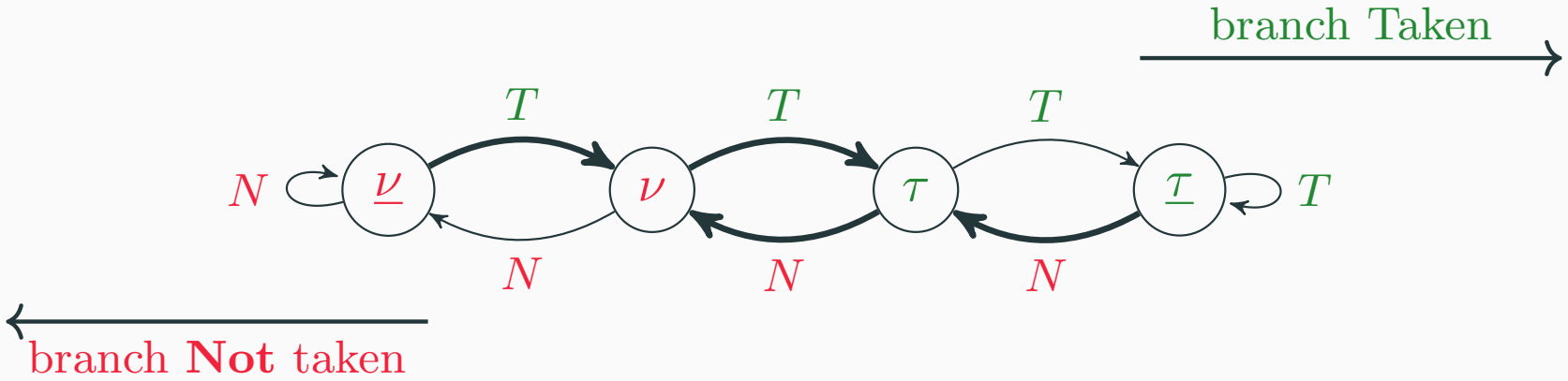
A **dynamic** branch predictor uses runtime information (the branch execution history) for accuracy.

- **local**: independent history for each conditional instructions
- **global**: shared history of all conditional instructions
- Since the 2000s, processors use a combination of both.

Computer Architecture: A Quantitative Approach (5th ed.), Hennessy & Patterson

Main model: standard local branch predictor

2-bit saturated counter for each conditional branching instruction.



State = prediction:

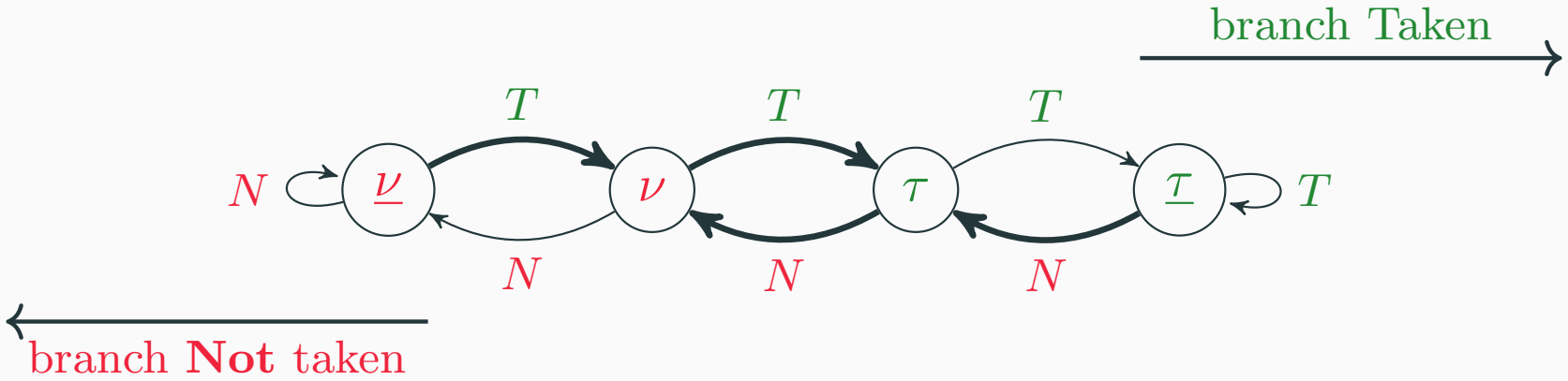
- $\underline{\nu}$ and ν predicts not taken (*false*)
- τ and $\underline{\tau}$ predicts taken (*true*)

Transition = actual outcome: the branch is **T**aken or **N**ot taken

Bold transition = misprediction

Main model: standard local branch predictor

2-bit saturated counter for each conditional branching instruction.



State = prediction:

- $\underline{\nu}$ and ν predicts not taken (*false*)
- τ and $\underline{\tau}$ predicts taken (*true*)

Transition = actual outcome: the branch is **T**aken or **N**ot taken

Bold transition = misprediction

▷ How does it work during the execution of an algorithm?

Back to simultaneous min/max

Expected number of mispredictions, uniform distribution

- **Optimized**: $n/4 + \mathcal{O}(\log n)$
- **Naive**: $2 \log n$

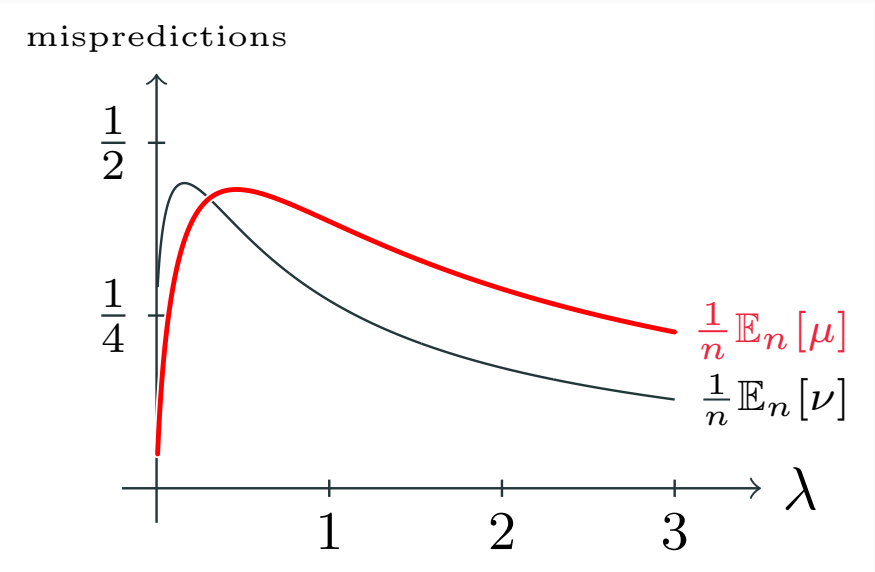
Proof: expectation of the number of records in uniform permutations.

Back to simultaneous min/max

Expected number of mispredictions, uniform distribution

- **Optimized**: $n/4 + \mathcal{O}(\log n)$
- **Naive**: $2 \log n$

Proof: expectation of the number of records in uniform permutations.



With record-biased permutations as input, when $\theta = \lambda n$.

μ: naive algorithm
ν: optimized algorithm

$\mathbb{E}_n[\mu] \sim \mathbb{E}_n[\nu]$ for $\lambda_0 \approx 0.305$.

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting

Tradeoffs Between Branch Mispredictions and Comparisons for Sorting Algorithms

Gerth Stølting Brodal[1,⋆] and Gabriel Moruz[1]

BRICS**, Department of Computer Science, University of Aarhus, IT Parken, Åbogade 34, DK-8200 Århus N, Denmark. E-mail: {gerth,gabi}@daimi.au.dk

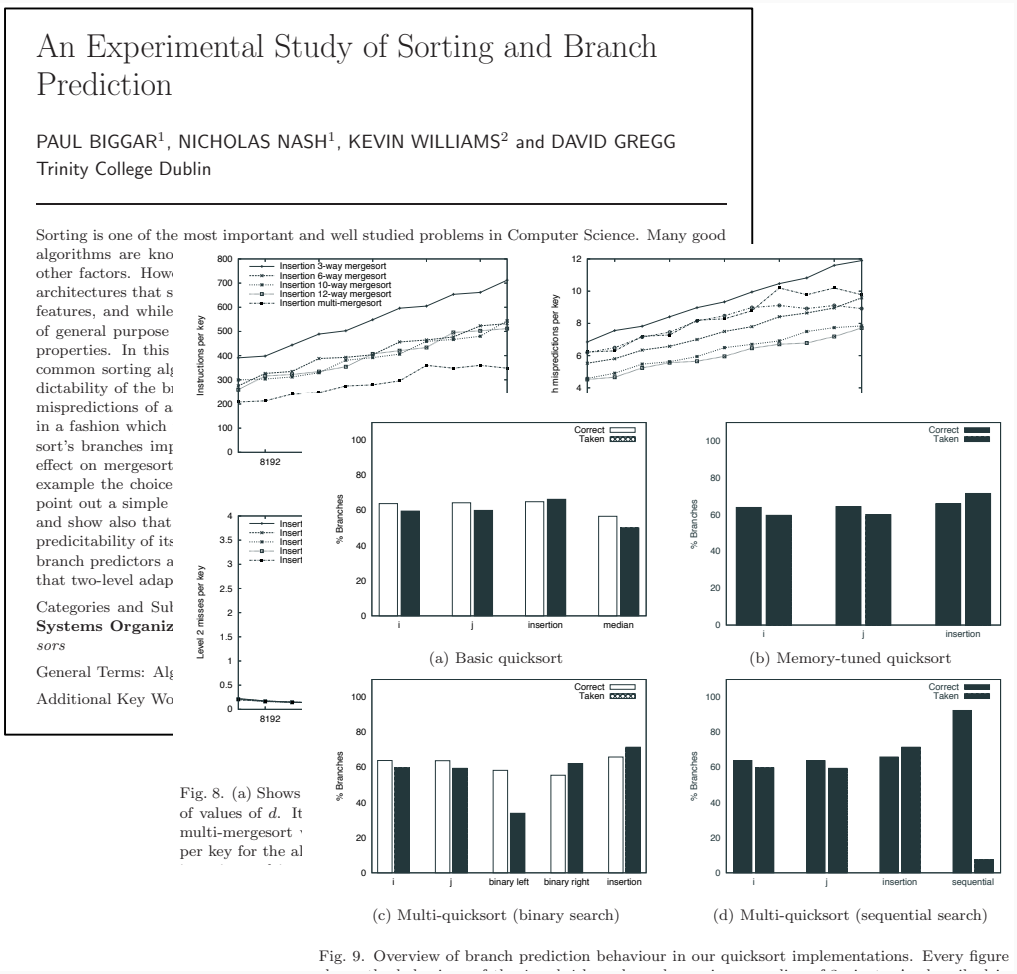
Abstract. Branch mispredictions is an important factor affecting the running time in practice. In this paper we consider tradeoffs between the number of branch mispredictions and the number of comparisons for sorting alg
algorithm
mispredicti
by adoptin
tions. For
rithm perf
$\Omega(n \log_d(1$
inversions
Estivill-Ca
a multiway
tions.

Measure	Comparisons	Branch mispredictions
Dis	$O(dn(1+\log(1+\text{Dis})))$	$\Omega(n \log_d(1+\text{Dis}))$
Exc	$O(dn(1+\text{Exc}\log(1+\text{Exc})))$	$\Omega(n\text{Exc} \log_d(1+\text{Exc}))$
Enc	$O(dn(1+\log(1+\text{Enc})))$	$\Omega(n \log_d(1+\text{Enc}))$
Inv	$O(dn(1+\log(1+\text{Inv}/n)))$	$\Omega(n \log_d(1+\text{Inv}/n))$
Max	$O(dn(1+\log(1+\text{Max})))$	$\Omega(n \log_d(1+\text{Max}))$
Osc	$O(dn(1+\log(1+\text{Osc}/n)))$	$\Omega(n \log_d(1+\text{Osc}/n))$
Reg	$O(dn(1+\log(1+\text{Reg})))$	$\Omega(n \log_d(1+\text{Reg}))$
Rem	$O(dn(1+\text{Rem}\log(1+\text{Rem})))$	$\Omega(n\text{Rem} \log_d(1+\text{Rem}))$
Runs	$O(dn(1+\log(1+\text{Runs})))$	$\Omega(n \log_d(1+\text{Runs}))$
SMS	$O(dn(1+\log(1+\text{SMS})))$	$\Omega(n \log_d(1+\text{SMS}))$
SUS	$O(dn(1+\log(1+\text{SUS})))$	$\Omega(n \log_d(1+\text{SUS}))$

Fig. 4. Lower bounds on the number of branch mispredictions for deterministic comparison based adaptive sorting algorithms for different measures of presortedness, given the upper bounds on the number of comparisons.

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting
- Biggar *et al*, 2008: experimental, branch prediction and sorting



An Experimental Study of Sorting and Branch Prediction

PAUL BIGGAR[1], NICHOLAS NASH[1], KEVIN WILLIAMS[2] and DAVID GREGG
Trinity College Dublin

Sorting is one of the most important and well studied problems in Computer Science. Many good
algorithms are kno
other factors. How
architectures that s
features, and while
of general purpose
properties. In this
common sorting alg
dictability of the bi
mispredictions of a
in a fashion which
sort's branches imp
effect on mergesort
example the choice
point out a simple
and show also that
predicitability of its
branch predictors a
that two-level adap

Categories and Sub
Systems Organiz
sors

General Terms: Alg

Additional Key Wo

(a) Basic quicksort

(b) Memory-tuned quicksort

(c) Multi-quicksort (binary search)

(d) Multi-quicksort (sequential search)

Fig. 8. (a) Shows
of values of *d*. It
multi-mergesort
per key for the al

Fig. 9. Overview of branch prediction behaviour in our quicksort implementations. Every figure

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting
- Biggar *et al*, 2008: experimental, branch prediction and sorting
- Kaligosi & Sanders, 2006: mispredictions and quicksort (worst-case)

How Branch Mispredictions Affect Quicksort

Kanela Kaligosi[1] and Peter Sanders[2]

[1] Max Planck In
Saarbrüc
kaligosi@
[2] Universität K
sanders

Abstract. We explain the cou
"good" pivots (close to the medi
not impro
pivot *imp*
count dec
direction
fect of sim
hardware.

Table 1. Number of branch mispredictions

	random pivot	α-skewed pivot
static predictor	$\frac{\ln 2}{2} n \lg n + \mathcal{O}(n)$, $\frac{\ln 2}{2} \approx 0.3466$	$\frac{\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha < 1/2$ $\frac{1-\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha \geq 1/2$
1-bit predictor	$\frac{2 \ln 2}{3} n \lg n + \mathcal{O}(n)$, $\frac{2 \ln 2}{3} \approx 0.4621$	$\frac{2\alpha(1-\alpha)}{H(\alpha)} n \lg n + \mathcal{O}(n)$
2-bit predictor	$\frac{28 \ln 2}{45} n \lg n + \mathcal{O}(n)$, $\frac{28 \ln 2}{45} \approx 0.4313$	$\frac{2\alpha^4 - 4\alpha^3 + \alpha^2 + \alpha}{(1-\alpha(1-\alpha))H(\alpha)} n \lg n + \mathcal{O}(n)$

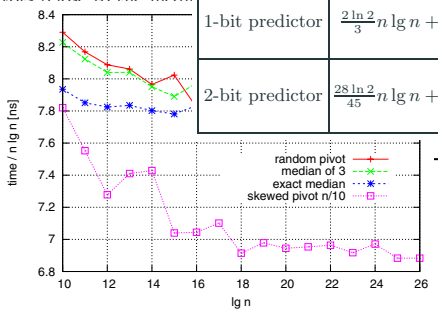


Fig. 3. Time / $n \lg n$ for random pivot, median of 3, exact median, 1/10-skewed pivot

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting
- Biggar *et al*, 2008: experimental, branch prediction and sorting
- Kaligosi & Sanders, 2006: mispredictions and quicksort (worst-case)
- Martínez, Nebel & Wild, 2014: mispredictions and quicksort (average)

Analysis of Branch Misses in Quicksort*

Conrado Martínez† Markus E. Nebel‡§ Sebastian Wild‡

November 11, 2014

Abstract

The analysis of algorithms mostly relies on counting classic elementary operations like additions, multiplications, comparisons, swaps etc. This approach is of
efficiency. I
ern process
and memor
running tin
get a reliab
sort: It has
under certa
sically opti
dian of a s
mispredicte
dominating
pivots are chosen from a sample of the input. We conclude that the difference in branch misses is too small to explain the superiority of the dual-pivot algorithm.

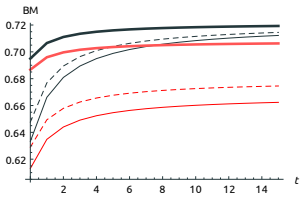


Figure 5: Branch mispredictions, as a function of t, in CQS (black) and YQS (red) with 1-bit branch prediction (fat), 2-bit saturating counter (thin solid) and 2-bit flip-consecutive (dashed) using symmetric sampling: $\mathbf{t}_{\mathrm{CQS}} = (3t+2, 3t+2)$ and $\mathbf{t}_{\mathrm{YQS}} = (2t+1, 2t+1, 2t+1)$

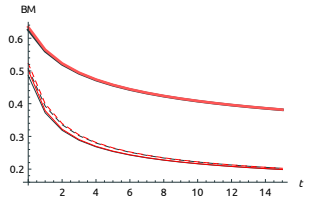
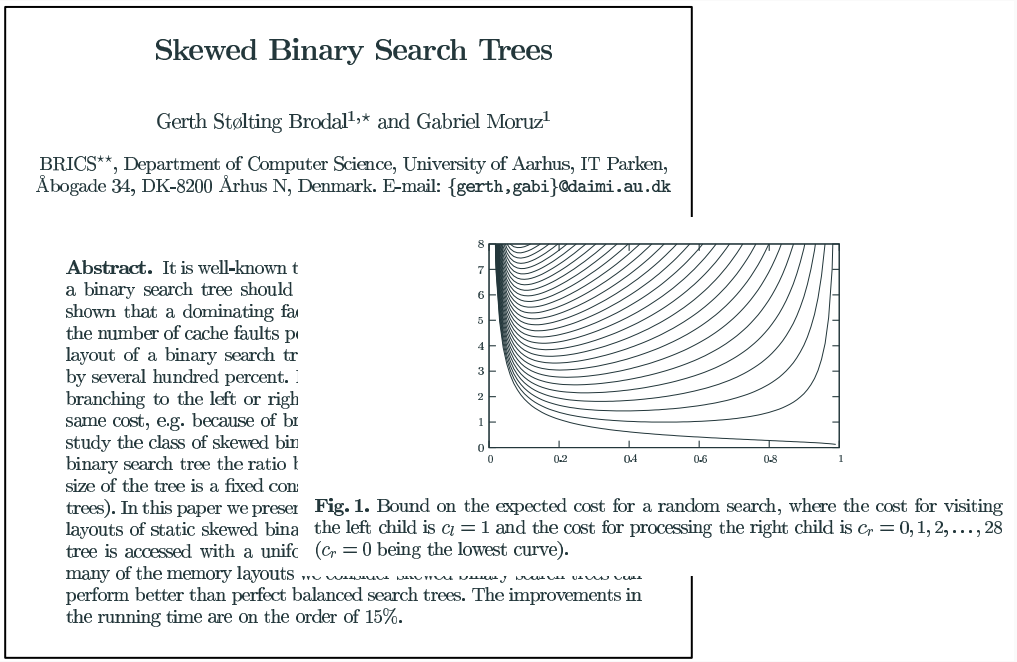


Figure 6: Branch mispredictions, as a function of t, in CQS (black) and YQS (red) with 1-bit (fat), 2-bit sc (thin solid) and 2-bit fc (dashed) predictors, using extremely skewed sampling: $\mathbf{t}_{\mathrm{CQS}} = (0, 6t+4)$ and $\mathbf{t}_{\mathrm{YQS}} = (0, 6t+3, 0)$

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting
- Biggar *et al*, 2008: experimental, branch prediction and sorting
- Kaligosi & Sanders, 2006: mispredictions and quicksort (worst-case)
- Martínez, Nebel & Wild, 2014: mispredictions and quicksort (average)
- Brodal & Moruz, 2006: skewed binary search trees



Skewed Binary Search Trees

Gerth Stølting Brodal[1,*] and Gabriel Moruz[1]

BRICS**, Department of Computer Science, University of Aarhus, IT Parken, Åbogade 34, DK-8200 Århus N, Denmark. E-mail: {gerth,gabi}@daimi.au.dk

Abstract. It is well-known t
a binary search tree should
shown that a dominating fa
the number of cache faults p
layout of a binary search tr
by several hundred percent.
branching to the left or righ
same cost, e.g. because of br
study the class of skewed bir
binary search tree the ratio b
size of the tree is a fixed con
trees). In this paper we preser
layouts of static skewed bina
tree is accessed with a unifc
many of the memory layouts
perform better than perfect balanced search trees. The improvements in the running time are on the order of 15%.

Fig. 1. Bound on the expected cost for a random search, where the cost for visiting the left child is $c_l = 1$ and the cost for processing the right child is $c_r = 0, 1, 2, \ldots, 28$ ($c_r = 0$ being the lowest curve).

Analysis of mispredictions for various algorithms

- Brodal & Moruz, 2005: mispredictions and (adaptive) sorting
- Biggar *et al*, 2008: experimental, branch prediction and sorting
- Kaligosi & Sanders, 2006: mispredictions and quicksort (worst-case)
- Martínez, Nebel & Wild, 2014: mispredictions and quicksort (average)
- Brodal & Moruz, 2006: skewed binary search trees
- Auger, Nicaud & Pivoteau, 2016: average (trade-off) analysis for min/max, exponentiation and binary search

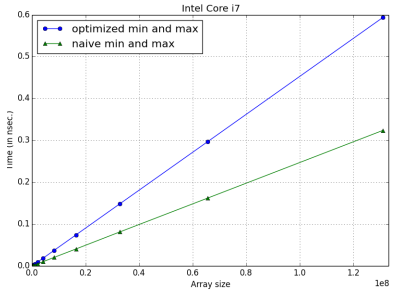
Good Predictions Are Worth a Few Comparisons

Nicolas Auger, Cyril Nicaud, and Carine Pivoteau

Université Paris-Est, LIGM (UMR 8049), F77454 Marne-la-Vallée, France

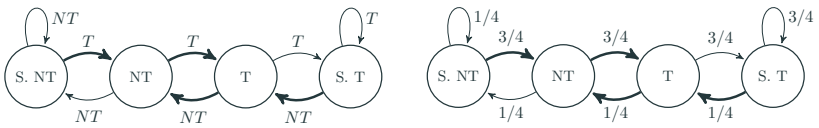
Abstract

Most modern processors are heavily parallelized and use predictor
conditional branches, in order to avoid costly stalls in their pipelir
friendly versions of two classical algorithms: exponentiation by squa
sorted array. These variants result in less mispredictions on average,
number of operations. These theoretical results are supported by e
that our algorithms perform significantly better than the standard on



1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms ε

Keywords and phrases branch misses, binary search, exponentiation by squaring, Markov chains



Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)
    r = r * x;
  n /= 2;
  x = x * x;
}
```

`x` is a floating-point number, `n` is an integer (uniform random bits) and `r` is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)  ← 1/2
    r = r * x;
  n /= 2;
  x = x * x;
}
```

`x` is a floating-point number, `n` is an integer (uniform random bits) and `r` is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)  ← 1/2
    r = r * x;
  n /= 2;
  x = x * x;
}
```

x is a floating-point number, n is an integer (uniform random bits) and r is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  # last 2 bits are not 00
  if (n & 3){
    if (n & 1) # last bit is 1
      r = r * x;
    if (n & 2)# next bit is 1
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```

Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)  ← 1/2
    r = r * x;
  n /= 2;
  x = x * x;
}
```

x is a floating-point number, n is an integer (uniform random bits) and r is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  # last 2 bits are not 00
  if (n & 3){
    if (n & 1) # last bit is 1
      r = r * x;
    if (n & 2)# next bit is 1
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```

- The number of multiplications is the same.
- GUIDED exponential performs more comparisons.

Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)  ← 1/2
    r = r * x;
  n /= 2;
  x = x * x;
}
```

x is a floating-point number, n is an integer (uniform random bits) and r is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  # last 2 bits are not 00
  if (n & 3){  ← 3/4
    if (n & 1) # last bit is 1   ← 2/3
      r = r * x;
    if (n & 2)# next bit is 1   ← 2/3
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```

- The number of multiplications is the same.
- GUIDED exponential performs more comparisons.

Exponentiation by squaring

STANDARD(x,n)

```
r = 1;
while (n > 0) {
  # n is odd
  if (n & 1)  ← 1/2
    r = r * x;
  n /= 2;
  x = x * x;
}
```

x is a floating-point number, n is an integer (uniform random bits) and r is the result.

$x^n = (x^2)^{\lfloor n/2 \rfloor} x^{n_0}$

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  # last 2 bits are not 00
  if (n & 3){  ← 3/4
    if (n & 1) # last bit is 1   ← 2/3
      r = r * x;
    if (n & 2)# next bit is 1   ← 2/3
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```

- The number of multiplications is the same.
- GUIDED exponential performs more comparisons.
- Yet, in practice, GUIDED exponential is faster !

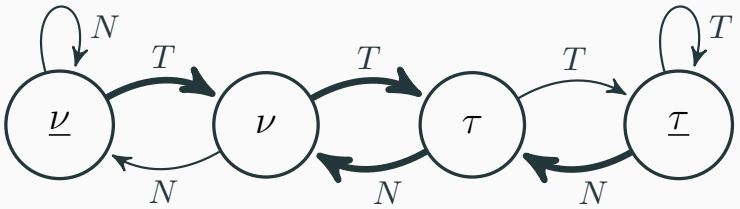
Average number of mispredictions

Computation of x^n, for random n in $[0, N-1]$

- **Expected nb. of conditionals**
  - $\sim \log_2 N$ for STANDARD
  - $\sim$ **5/4** $\log_2 N$ for GUIDED
- **Expected nb. of mispredictions**
  - $\sim$ **1/2** $\log_2 N$ for STANDARD
  - $\sim$ **9/20** $\log_2 N$ for GUIDED (2-bit pred.)

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  if (n & 3) {  ←
    if (n & 1)
      r = r * x;
    if (n & 2)
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```



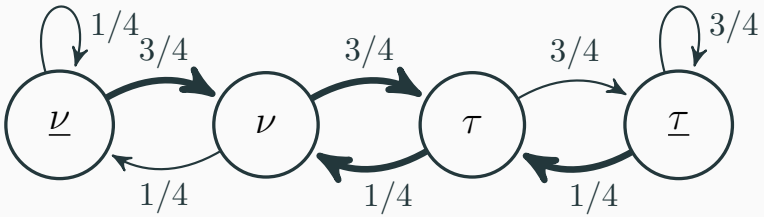
Average number of mispredictions

Computation of x^n, for random n in $[0, N-1]$

- **Expected nb. of conditionals**
  - $\sim \log_2 N$ for STANDARD
  - $\sim$ **5/4** $\log_2 N$ for GUIDED
- **Expected nb. of mispredictions**
  - $\sim$ **1/2** $\log_2 N$ for STANDARD
  - $\sim$ **9/20** $\log_2 N$ for GUIDED (2-bit pred.)

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  if (n & 3) {  ←
    if (n & 1)
      r = r * x;
    if (n & 2)
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```



$$\mu(p) = \sum_{\substack{(i,j) \text{ is} \\ \text{mispred.}}} \hat{\pi}_p(i) M_p(i,j)$$

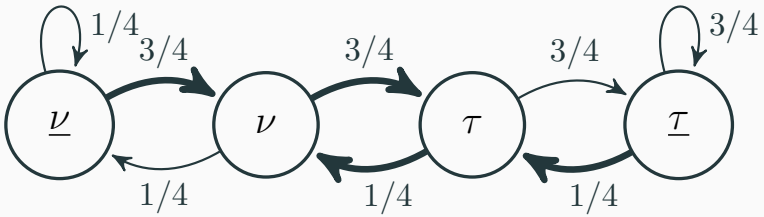
Average number of mispredictions

Computation of x^n, for random n in $[0, N-1]$

- **Expected nb. of conditionals**
  - $\sim \log_2 N$ for STANDARD
  - $\sim$ **5/4** $\log_2 N$ for GUIDED
- **Expected nb. of mispredictions**
  - $\sim$ **1/2** $\log_2 N$ for STANDARD
  - $\sim$ **9/20** $\log_2 N$ for GUIDED (2-bit pred.)

GUIDED(x,n)

```
r = 1;
while (n > 0) {
  t = x * x;
  if (n & 3) {  ←
    if (n & 1)
      r = r * x;
    if (n & 2)
      r = r * t;
  }
  n /= 4;
  x = t * t;
}
```



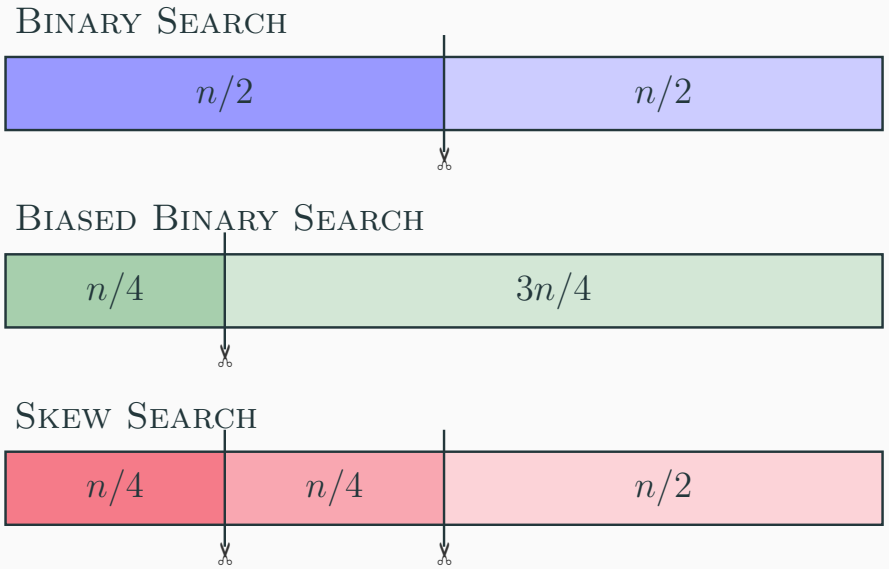
$$\mu(p) = \sum_{\substack{(i,j) \text{ is} \\ \text{mispred.}}} \hat{\pi}_p(i) M_p(i,j)$$

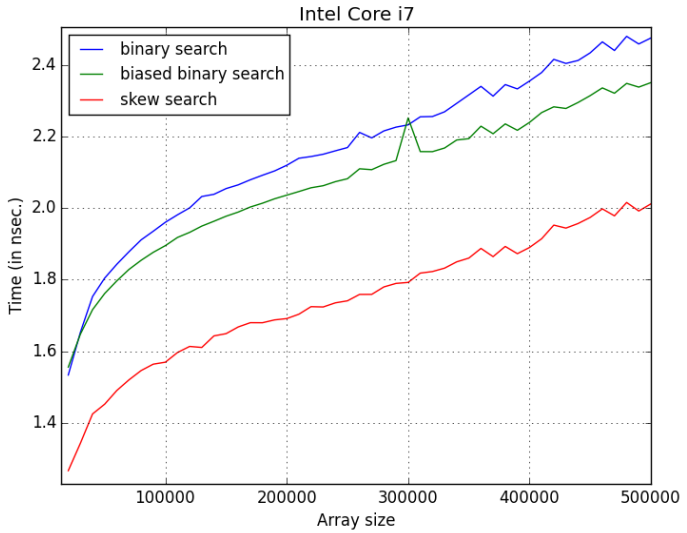
Number of mispredictions (Ergodic Th.):

$$\mathbb{E}[M_n] \sim \mathbb{E}[L_n] \times \mu(p)$$

L_n is the length of the path in the Markov chain, $\mu(\frac{3}{4}) = \frac{3}{10}$ and $\mu(\frac{2}{3}) = \frac{2}{5}$.

Binary search



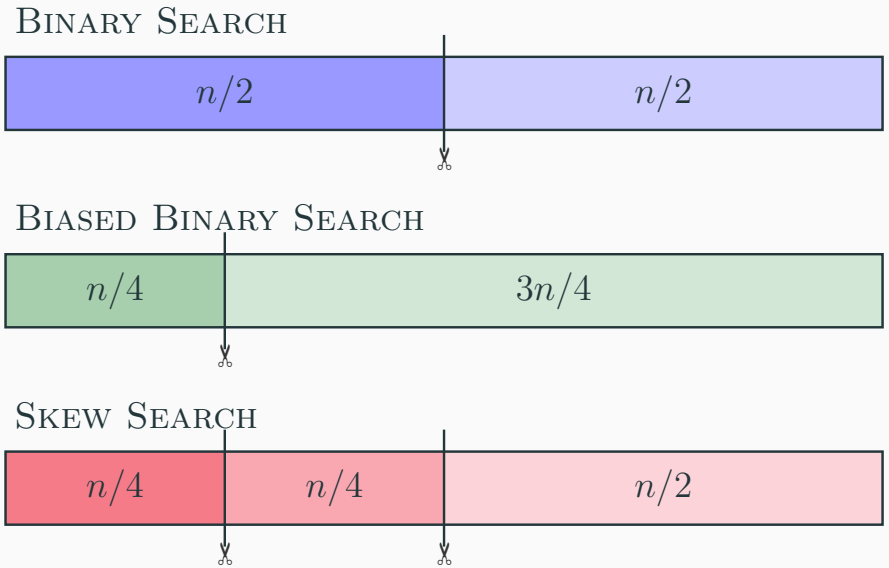


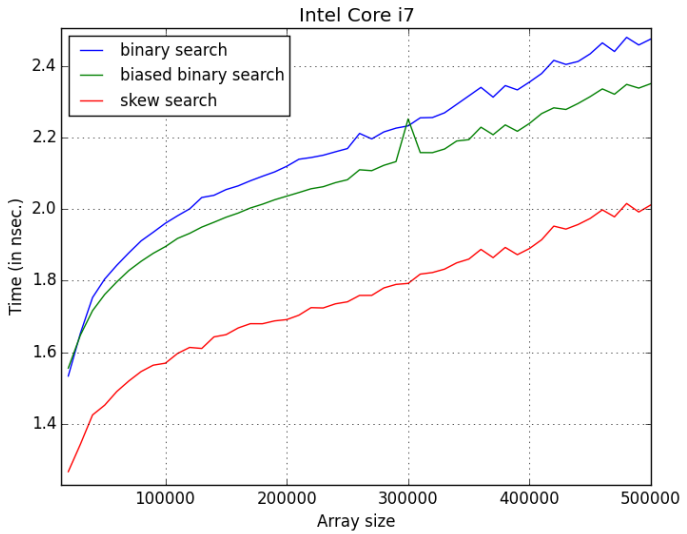
Theorem (2-bit saturated counter)

For arrays of size n filled with random uniform integers. C_n is the number of comparisons and M_n the number of mispredictions.

	BinarySearch	BiasedBinarySearch	SkewSearch
$\mathbb{E}[C_n]$	$\frac{\log n}{\log 2}$	$\frac{4 \log n}{(4 \log 4 - 3 \log 3)}$	$\frac{7 \log n}{(6 \log 2)}$
$\mathbb{E}[M_n]$	$\frac{\log n}{(2 \log 2)}$	$\mu(\frac{1}{4})\mathbb{E}[C_n]$	$(\frac{4}{7}\mu(\frac{1}{4}) + \frac{3}{7}\mu(\frac{1}{3}))\mathbb{E}[C_n]$

Binary search





Theorem (2-bit saturated counter)

For arrays of size n filled with random uniform integers. C_n is the number of comparisons and M_n the number of mispredictions.

	BinarySearch	BiasedBinarySearch	SkewSearch
$\mathbb{E}[C_n]$	$1.44 \log n$	$1.78 \log n$	$1.68 \log n$
$\mathbb{E}[M_n]$	$0.72 \log n$	$0.53 \log n$	$0.58 \log n$

Beyond independent branches (with C. Nicaud and S. Vialette)

Previous algorithms rely on branching instructions, but these are mostly independent. What happens when branches are correlated?

Beyond independent branches (with C. Nicaud and S. Vialette)

Previous algorithms rely on branching instructions, but these are mostly independent. What happens when branches are correlated?

This is the case for **pattern matching** algorithms.

- How can we study the **impact of branch prediction** on them?
- Can we observe it on the **execution time**?

Beyond independent branches (with C. Nicaud and S. Vialette)

Previous algorithms rely on branching instructions, but these are mostly independent. What happens when branches are correlated?

This is the case for **pattern matching** algorithms.

- ▷ How can we study the **impact of branch prediction** on them?
- ▷ Can we observe it on the **execution time**?

- Sliding window algorithm
  - Complexity in number of comparisons: $\mathcal{O}(mn)$
  - `String.indexOf()` in `Java`
  - For some patterns, the misprediction rate is over 50%.
- Today: Focus on **MP** and **KMP** algorithms.

Expected number of mispredictions for MP/KMP

Input: **random text** W of length n, **fixed pattern** X of length m

Expected number of mispredictions for MP/KMP

Input: **random text** W of length n, **fixed pattern** X of length m

while $j < n$ **do**
    while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
        $i \leftarrow B[i]$ `# precomputed border table, ≠ for MP/KMP`
    $i, j \leftarrow i + 1, j + 1$
    if $i = m$ **then**
        $i \leftarrow B[i]$
        $nb \leftarrow nb + 1$

Expected number of mispredictions for MP/KMP

Input: **random text** W of length n, **fixed pattern** X of length m

while $j < n$ **do** ← super easy: at most 3
  while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
    $i \leftarrow B[i]$ # precomputed border table, $\neq$ for MP/KMP
  $i, j \leftarrow i + 1, j + 1$
  if $i = m$ **then** ← almost easy: $\sim$ nb. of occurrences of X
    $i \leftarrow B[i]$
    $nb \leftarrow nb + 1$

Expected number of mispredictions for MP/KMP

Input: **random text** W of length n, **fixed pattern** X of length m

while $j < n$ **do**
  while $i \geq 0$ **and** $X[i] \neq W[j]$ **do** ← focus on this one
    $i \leftarrow B[i]$ # precomputed border table, $\neq$ for MP/KMP
  $i, j \leftarrow i + 1, j + 1$
  if $i = m$ **then**
    $i \leftarrow B[i]$
    $nb \leftarrow nb + 1$

Analysis of the mispredictions caused by letter comparisons:

- depends on the pattern X
- probability measure on A such that for all $\alpha \in A$, $0 < \pi(\alpha) < 1$
- transducer for the (mis)predictions + Markov chain

Expected number of mispredictions for MP/KMP

Input: **random text** W of length n, **fixed pattern** X of length m

while $j < n$ **do**
    while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
        $i \leftarrow B[i]$ `# precomputed border table, ≠ for MP/KMP`
    $i, j \leftarrow i + 1, j + 1$
    if $i = m$ **then**
        $i \leftarrow B[i]$
        $nb \leftarrow nb + 1$

Analysis of the mispredictions caused by letter comparisons:

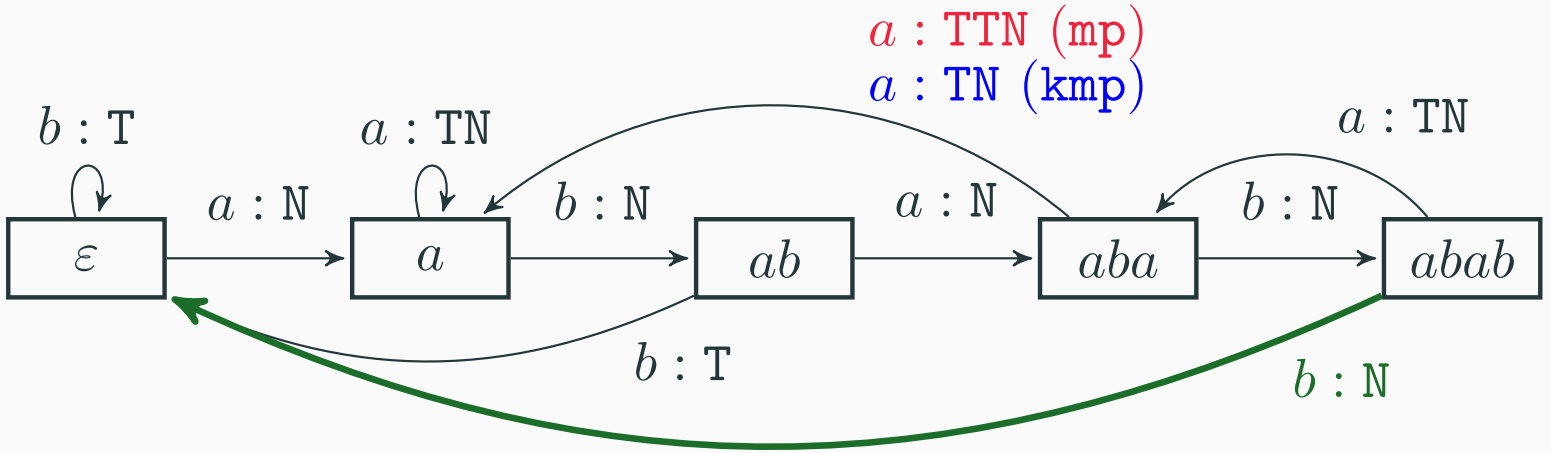
- depends on the pattern X
- probability measure on A such that for all $\alpha \in A$, $0 < \pi(\alpha) < 1$
- transducer for the (mis)predictions + Markov chain
- same kind of ideas for $i \geq 0$

Following letter comparisons on the automaton $X = ababb$

Input: random text W of length n, fixed pattern X of length m

while $j < n$ **do**
  while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
    $i \leftarrow B[i]$
  $i, j \leftarrow i + 1, j + 1$
  if $i = m$ **then**
    $i \leftarrow B[i]$
    $nb \leftarrow nb + 1$

	a	b	a	a	b	a	b	...
	△							
	a	b	a	b	b			

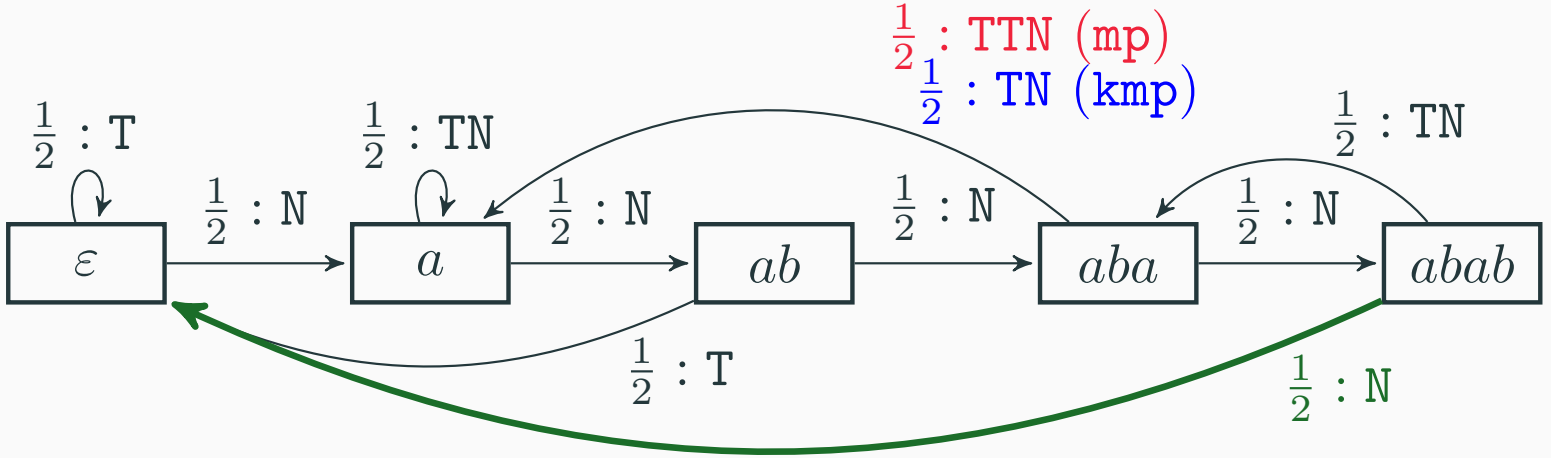


Following letter comparisons on the automaton $X = ababb$

Input: random text W of length n, fixed pattern X of length m

while $j < n$ **do**
  while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
    $i \leftarrow B[i]$
  $i, j \leftarrow i + 1, j + 1$
  if $i = m$ **then**
    $i \leftarrow B[i]$
    $nb \leftarrow nb + 1$

	a	b	a	a	b	a	b	...
	△							
	a	b	a	b	b			

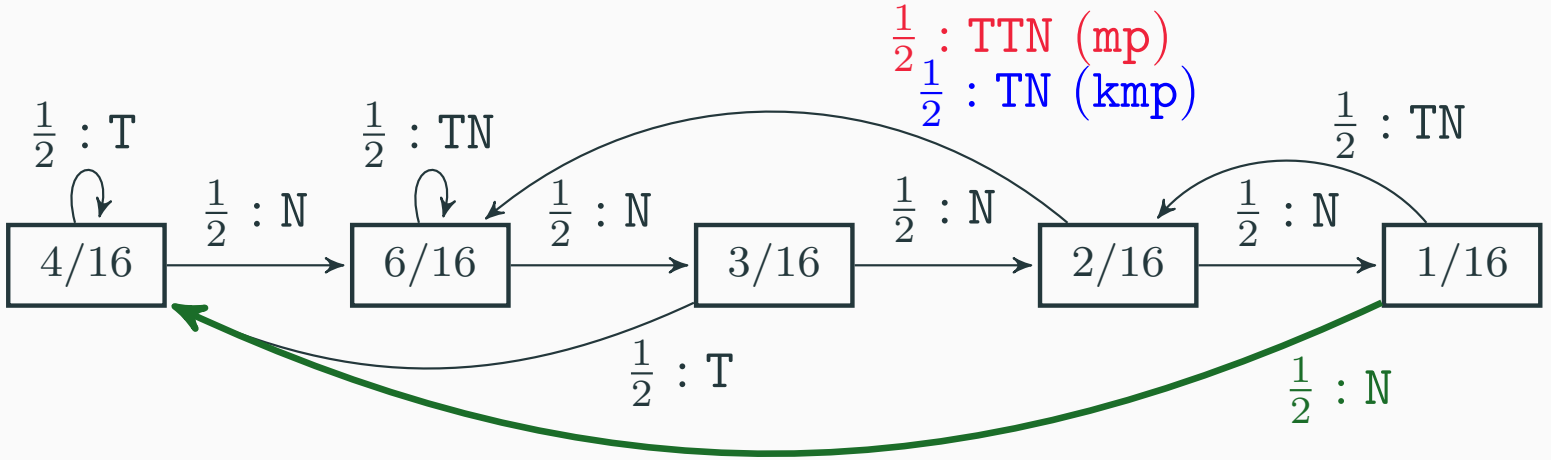


Following letter comparisons on the automaton $X = ababb$

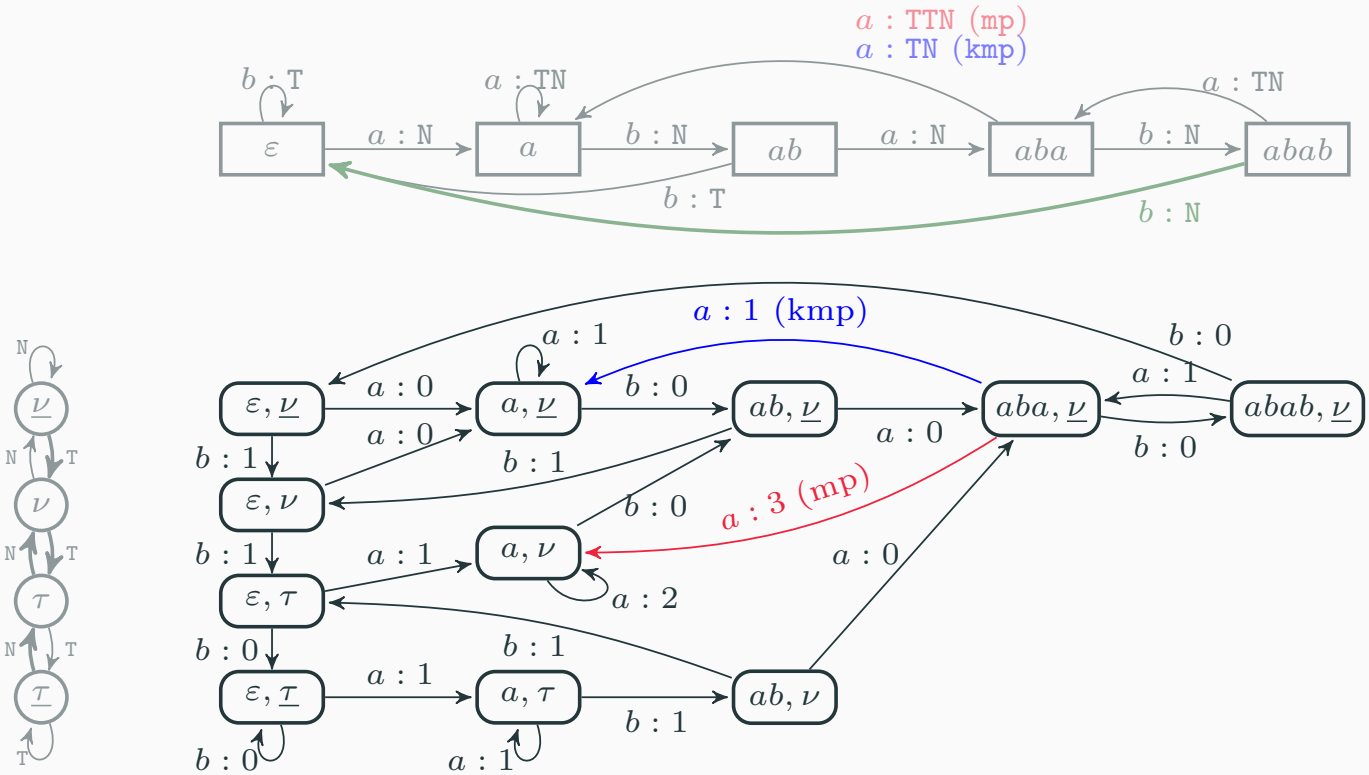
Input: random text W of length n, fixed pattern X of length m

while $j < n$ **do**
  while $i \geq 0$ **and** $X[i] \neq W[j]$ **do**
    $i \leftarrow B[i]$
  $i, j \leftarrow i + 1, j + 1$
  if $i = m$ **then**
    $i \leftarrow B[i]$
    $nb \leftarrow nb + 1$

	a	b	a	a	b	a	b	...
	△							
	a	b	a	b	b			



Computing the number of mispredictions $X = ababb$



Proposition

The expected number of **mispredictions** *caused by letter comparisons in KMP on a random text of length n and a pattern X, is asymptotically equivalent to $L_X \cdot n$, with*

$$L_X = \sum_{u \in Q_X} \sum_{\lambda \in \{\underline{\nu}, \nu, \tau, \underline{\tau}\}} \hat{\pi}(u, \lambda) \times \sum_{\alpha \in A} \pi(\alpha) \cdot output((u, \lambda) \xrightarrow{\alpha})$$

Asymptotic expected number of mispredictions **per symbol** in KMP with $\Sigma = \{a, b\}$ and $p := \pi(a) = 1 - \pi(b)$.

X	$i = m$	$i \geq 0$	$X[i] \neq T[j]$
aa	...	$1 - p$	$\frac{p(1-p)}{1-2p+2p^2}$
ab	$p(1-p)$	$(1-p)^2$	$\frac{p(3-7p+7p^2-2p^3)}{1-p+2p^2-p^3}$
aaa	$p^3(1-p)(1+p)^2$	$1 - p$	$\frac{p(1-p)}{1-2p+2p^2}$
aab	$p^2(1-p)$	$(1-p)^2(1+p)$	$\frac{p(1-2p^2-p^3+5p^4-3p^5+p^6)}{1-2p+3p^2-2p^3+p^4}$
aba	$p^2(1-p)$	$(1-p)^2$	$\frac{p(3-7p+7p^2-2p^3)}{1-p+2p^2-p^3}$
abb	$p(1-p)^2$	$(1-p)^3$	$p(4-13p+21p^2-16p^3+6p^4-p^5)$

Last column for $x = abab$:

$$\frac{\pi_a(-\pi_a^3\pi_b + 2\pi_a^2\pi_b^3 + 4\pi_a^2\pi_b^2 + 3\pi_a^2\pi_b + \pi_a^2 - 5\pi_a\pi_b^2 - 4\pi_a\pi_b - 2\pi_a + 2\pi_b + 1)}{(1-\pi_a)(\pi_a^2\pi_b^2 + \pi_a^2\pi_b - \pi_a\pi_b - \pi_a + 1)}$$

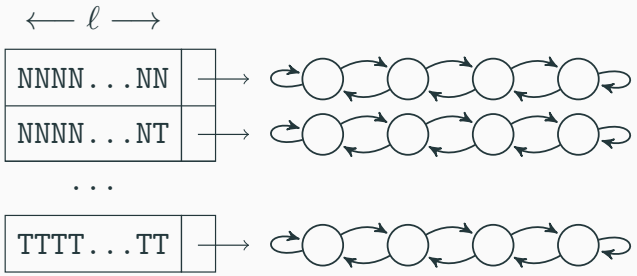
Asymptotic expected number of mispredictions per symbol in a random text, with **uniform** distribution over alphabets of size 2 or 4.

	$\lvert A\rvert = 2$				$\lvert A\rvert = 4$			
X	$i = m$	$i \geq 0$	$X[i] \neq T[j]$	**Total**	$i = m$	$i \geq 0$	$X[i] \neq T[j]$	**Total**
aa	0.283	0.5	0.5	1.283	0.073	0.75	0.3	1.123
ab	0.25	0.25	0.571	1.321	0.062	0.688	0.375	1.186
aaa	0.14	0.5	0.5	1.14	0.018	0.75	0.3	1.068
aab	0.125	0.375	0.542	1.166	0.015	0.734	0.322	1.086
aba	0.125	0.25	0.571	0.946	0.015	0.688	0.375	1.076
abb	0.125	0.125	0.547	0.921	0.015	0.672	0.397	1.098

Perspectives

Branch prediction

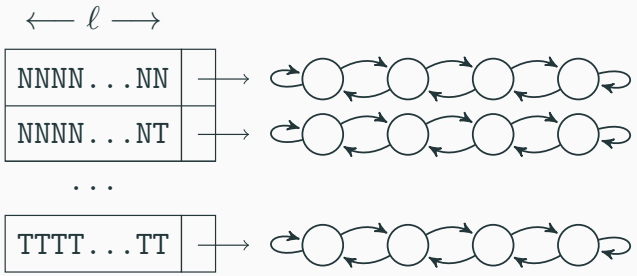
- **In progress:** analysis of a global predictor to capture correlations.
  ▷ Simulations: measured number of mispredictions roughly divided by $|A|$.



Perspectives

Branch prediction

- **In progress:** analysis of a global predictor to capture correlations.
  ▷ Simulations: measured number of mispredictions roughly divided by $|A|$.

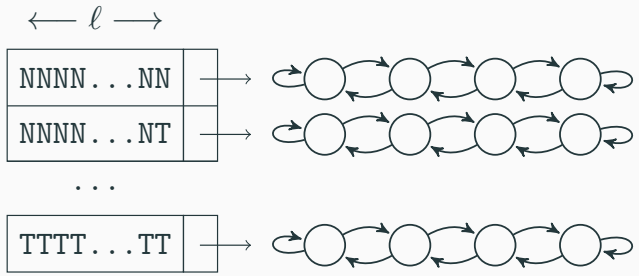


- More involved predictors, i.e., with multiple history sizes.
- Enhanced probabilistic distributions for texts, i.e., Markovian sources (stateful sources).
- A more general framework for the analysis.

Perspectives

Branch prediction

- **In progress:** analysis of a global predictor to capture correlations.
  ▷ Simulations: measured number of mispredictions roughly divided by $|A|$.



- More involved predictors, i.e., with multiple history sizes.
- Enhanced probabilistic distributions for texts, i.e., Markovian sources (stateful sources).
- A more general framework for the analysis.

Other computer architecture features: cache analysis

- Conjugate effects of cache and branch prediction.
- Additional parameters beyond cache and block size.

The End