

Optimal Dual-Pivot Quicksort

Exact Comparison Count

Martin Dietzfelbinger

Technische Universität Ilmenau

Based on joint work with
Martin Aumüller, Daniel Krenn, Clemens Heuberger, Helmut Prodinger

FCT, Bordeaux, September 11, 2017

Classical Quicksort

3 2 8 5 1 4 7 6

Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

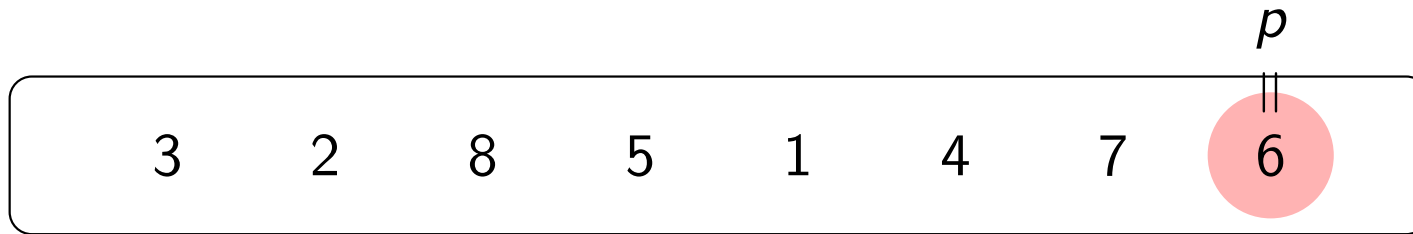
Classical Quicksort

3 2 8 5 1 4 7 6

Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.

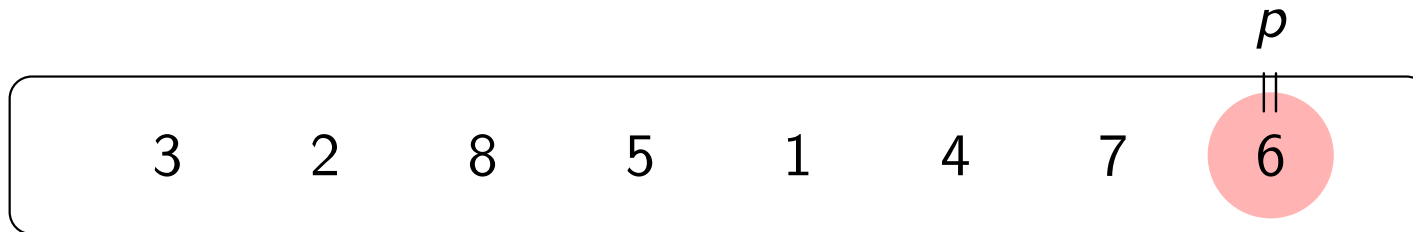
Classical Quicksort



Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.

Classical Quicksort

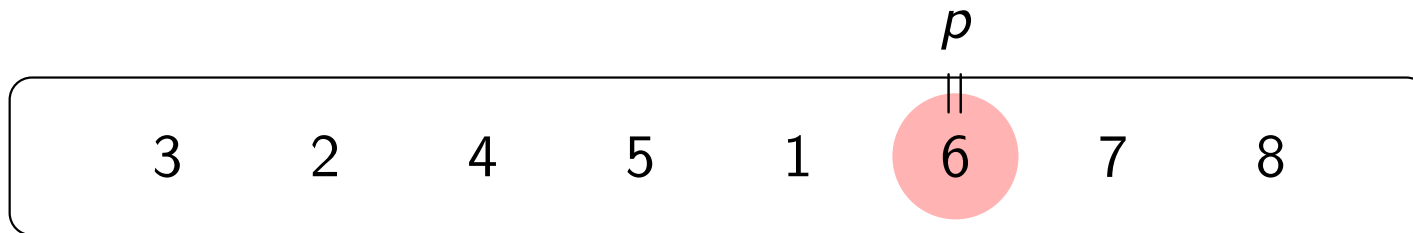


Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.
2. Partition, i.e., re-arrange elements;
cost: $n - 1$ **comparisons**.



Classical Quicksort

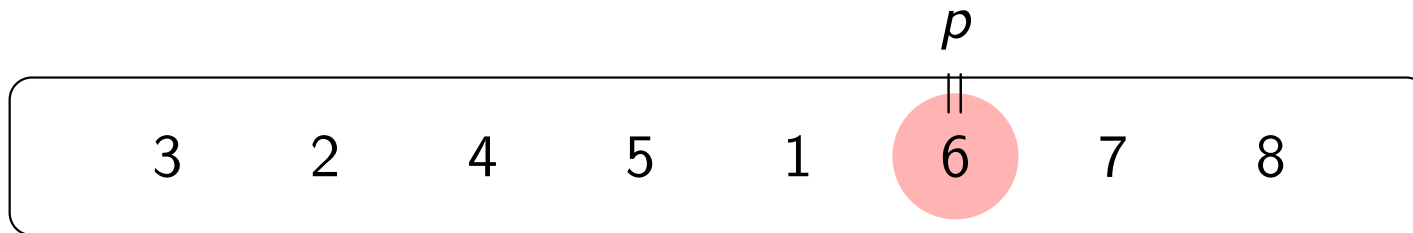


Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.
2. Partition, i.e., re-arrange elements;
cost: $n - 1$ **comparisons**.



Classical Quicksort



Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.
2. Partition, i.e., re-arrange elements;
cost: $n - 1$ **comparisons**.



3. Sort the two subarrays recursively.

Classical Quicksort

1 2 3 4 5 6 7 8

Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.
2. Partition, i.e., re-arrange elements;
cost: $n - 1$ **comparisons**.



3. Sort the two subarrays recursively. Done.

Classical Quicksort

1 2 3 4 5 6 7 8

Input: Distinct numbers a_1, \dots, a_n (**gross** simplification!).

1. Choose a pivot p from $\{a_1, \dots, a_n\}$.
2. Partition, i.e., re-arrange elements;
cost: $n - 1$ **comparisons**.



3. Sort the two subarrays recursively. Done.

Expected number of comparisons: $2n \ln n - \Theta(n)$.

Dual-Pivot Quicksort

3 2 8 5 1 4 7 6

Input: Distinct numbers a_1, \dots, a_n .

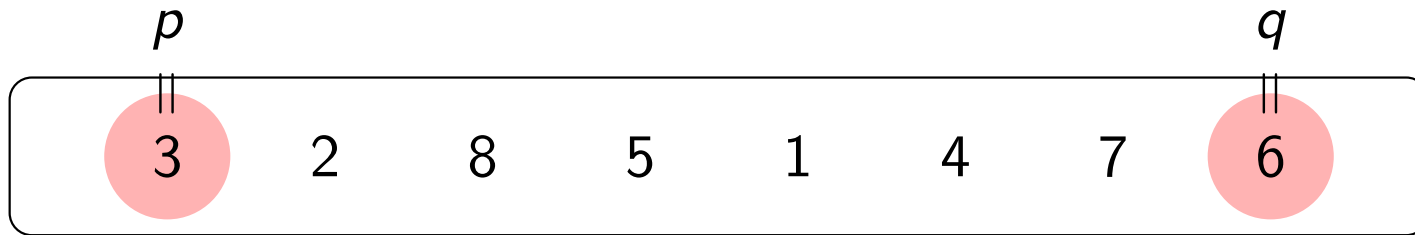
Dual-Pivot Quicksort

3 2 8 5 1 4 7 6

Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.

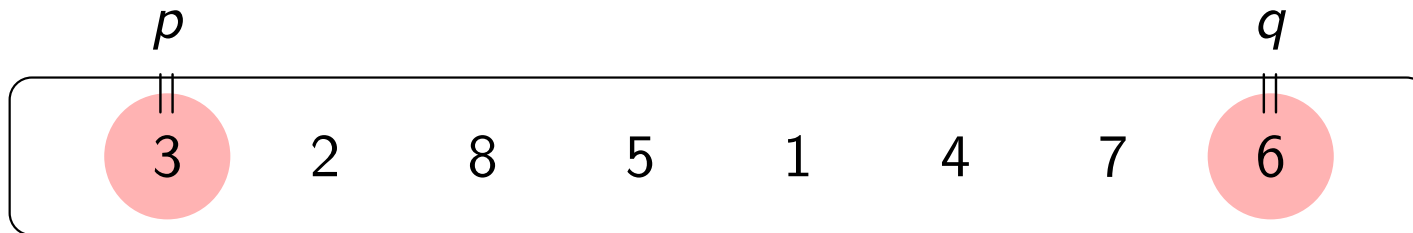
Dual-Pivot Quicksort



Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.

Dual-Pivot Quicksort



Input: Distinct numbers a_1, \dots, a_n .

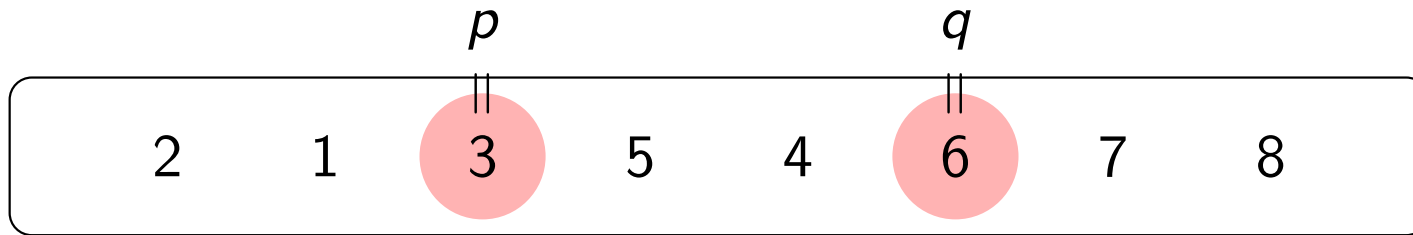
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



Details: in the [program](#).

Dual-Pivot Quicksort



Input: Distinct numbers a_1, \dots, a_n .

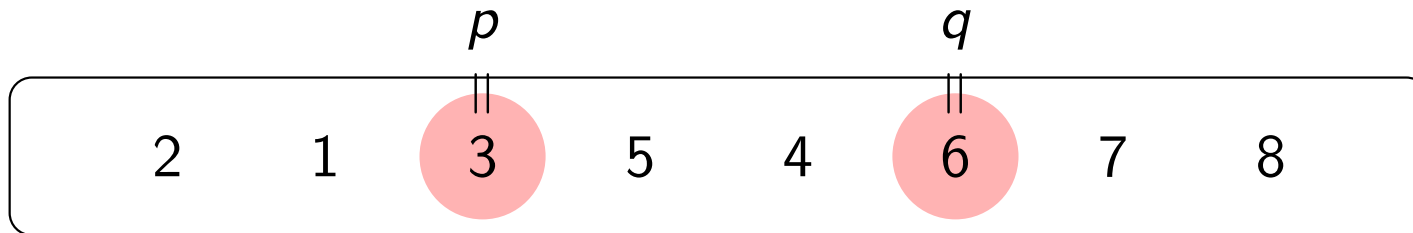
1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



Details: in the [program](#).

Dual-Pivot Quicksort



Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

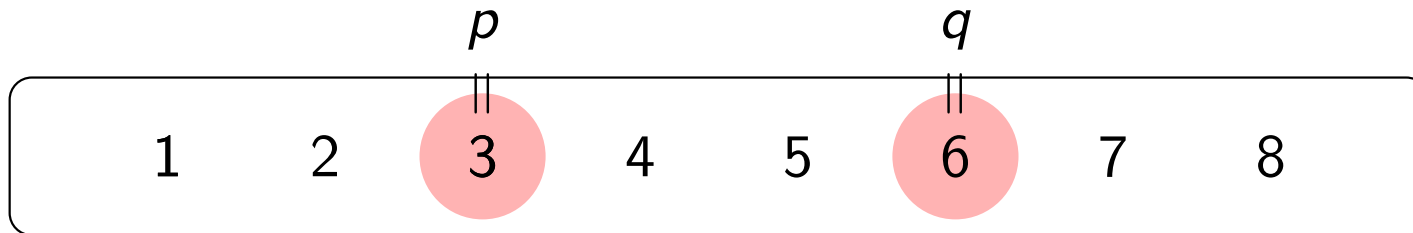
Partition:



Details: in the [program](#).

3. Sort the **three** subarrays recursively.

Dual-Pivot Quicksort



Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



Details: in the [program](#).

3. Sort the **three** subarrays recursively. Done.

Dual-Pivot Quicksort

1 2 3 4 5 6 7 8

Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



Details: in the **program**.

3. Sort the **three** subarrays recursively. Done.

Expected number of comparisons: ??

Dual-Pivot Quicksort

1 2 3 4 5 6 7 8

Input: Distinct numbers a_1, \dots, a_n .

1. Choose **two** pivots p, q with $p < q$.
2. Partition, i.e., re-arrange elements.

Partition:



Details: in the **program**.

3. Sort the **three** subarrays recursively. Done.

Expected number of comparisons: ??

Also interesting: Other cost measures like “swaps” or “running time”.

Our focus. . .

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$
- We count **comparisons**.

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$
- We count **comparisons**.
- **Expectations** of this count

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$
- We count **comparisons**.
- **Expectations** of this count
- Minimum (??) expected comparison count.

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$
- We count **comparisons**.
- **Expectations** of this count
- Minimum (??) expected comparison count.
- **Not** on optimizing running time.

Our focus. . .

- Input is a random permutation of $\{1, \dots, n\}$
- We count **comparisons**.
- **Expectations** of this count
- Minimum (??) expected comparison count.
- **Not** on optimizing running time.
- **Not** on concrete (implemented) algorithms.

Outline:

Outline:

Part 1:

Outline:

Part 1:

- A little history

Outline:

Part 1:

- A little history
- Model for “classification”

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Part 2:

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Part 2:

- “Count”: An absolutely optimal partitioning strategy

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Part 2:

- “Count”: An absolutely optimal partitioning strategy
- Exact analysis of comparisons in “count”

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Part 2:

- “Count”: An absolutely optimal partitioning strategy
- Exact analysis of comparisons in “count”
- Other cost measures (for two and more pivots)

Outline:

Part 1:

- A little history
- Model for “classification”
- Rough analysis, nearly optimal strategy

Part 2:

- “Count”: An absolutely optimal partitioning strategy
- Exact analysis of comparisons in “count”
- Other cost measures (for two and more pivots)
- Some open problems

Dual-Pivot Quicksort: Some History

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (PhD Thesis, 1975):**
Analyzed a dual-pivot algorithm (given as **program**),
found it makes many more **swaps** (and comparisons)
than classical QS → no further investigation.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (PhD Thesis, 1975):**
Analyzed a dual-pivot algorithm (given as **program**), found it makes many more **swaps** (and comparisons) than classical QS → no further investigation.
- **P. Hennequin (PhD Thesis, 1991):**
Thorough analysis of quicksort with $k \geq 1$ pivots (given as **program**).
 - ▶ for $k = 2$, no improvements over $2n \ln n$ found.
 - ▶ for $k \geq 3$, slight improvements over $2n \ln n$, partitioning considered “too complicated” to give improvements.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (PhD Thesis, 1975):**
Analyzed a dual-pivot algorithm (given as **program**), found it makes many more **swaps** (and comparisons) than classical QS → no further investigation.
- **P. Hennequin (PhD Thesis, 1991):**
Thorough analysis of quicksort with $k \geq 1$ pivots (given as **program**).
 - ▶ for $k = 2$, no improvements over $2n \ln n$ found.
 - ▶ for $k \geq 3$, slight improvements over $2n \ln n$, partitioning considered “too complicated” to give improvements.

Topic went to sleep.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (PhD Thesis, 1975):**
Analyzed a dual-pivot algorithm (given as **program**), found it makes many more **swaps** (and comparisons) than classical QS → no further investigation.
- **P. Hennequin (PhD Thesis, 1991):**
Thorough analysis of quicksort with $k \geq 1$ pivots (given as **program**).
 - ▶ for $k = 2$, no improvements over $2n \ln n$ found.
 - ▶ for $k \geq 3$, slight improvements over $2n \ln n$, partitioning considered “too complicated” to give improvements.

Topic went to sleep.

Java 7 (2009):

Classical quicksort is replaced by a dual-pivot quicksort variant, proposed (and carefully engineered) by **Yaroslavskiy, Bentley, and Bloch (YBB)**.

Dual-Pivot Quicksort: Some History

- **R. Sedgewick (PhD Thesis, 1975):**
Analyzed a dual-pivot algorithm (given as **program**), found it makes many more **swaps** (and comparisons) than classical QS → no further investigation.
- **P. Hennequin (PhD Thesis, 1991):**
Thorough analysis of quicksort with $k \geq 1$ pivots (given as **program**).
 - ▶ for $k = 2$, no improvements over $2n \ln n$ found.
 - ▶ for $k \geq 3$, slight improvements over $2n \ln n$, partitioning considered “too complicated” to give improvements.

Topic went to sleep.

Java 7 (2009):

Classical quicksort is replaced by a dual-pivot quicksort variant, proposed (and carefully engineered) by **Yaroslavskiy, Bentley, and Bloch (YBB)**.

Experiments: YBB algorithm around 10% faster than classical QS (keys are integers or reals).

YBB partitioning as program

```
1: procedure Y-Partition(A, p, q, left, right, posp, posq)
2:  $l \leftarrow left + 1; g \leftarrow right - 1; k \leftarrow 1;$  // pointers
3: while  $k \leq g$  do
4:   if  $A[k] < p$  then // small pivot first
5:     swap  $A[k]$  and  $A[l]; l \leftarrow l + 1;$ 
6:   else
7:     if  $A[k] > q$  then // large pivot later
8:       while  $A[g] > q$  do // small pivot first
9:          $g \leftarrow g - 1;$ 
10:      if  $k < g$  then
11:        if  $A[g] < p$  then // large pivot later
12:          rotate3( $A[g], A[k], A[l]$ );  $l \leftarrow l + 1;$ 
13:        else
14:          swap  $A[k]$  and  $A[g];$ 
15:           $g \leftarrow g - 1;$ 
16:       $k \leftarrow k + 1;$ 
17: swap  $A[left]$  and  $A[l - 1];$ 
18: swap  $A[right]$  and  $A[g + 1]; pos_p \leftarrow l - 1; pos_q \leftarrow g + 1;$ 
```

Dual-Pivot Quicksort: More History

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgwick: $2.13n \ln n + O(n)$

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS
(+ exact analysis of bytecode count).

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about constants:

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about constants:

- Why different?

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about **constants**:

- Why different?
- Other possibilities?

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about **constants**:

- Why different?
- Other possibilities?
- Best possible?

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about **constants**:

- Why different?
- Other possibilities?
- Best possible? ←

Dual-Pivot Quicksort: More History

Analysis of some dual-pivot algorithms by Wild and Nebel (2012), regarding **average comparison count**:

- YBB (simplified, omits pivot sampling): $1.9n \ln n + O(n)$
- Sedgewick: $2.13n \ln n + O(n)$

Wild, Nebel, Neininger (2015):

Distributional analysis of comparisons in YBB Dual-Pivot QS (+ exact analysis of bytecode count).

Sebastian Wild's Thesis (2016): Wealth of analysis of quicksort with two and more pivots, **program-based**.

Our questions about **constants**:

- Why different?
- Other possibilities?
- Best possible? ←

Part 1

Part 1

- Model to capture comparison count in **all** dual-pivot algorithms

Part 1

- Model to capture comparison count in **all** dual-pivot algorithms
- Unified analysis

Part 1

- Model to capture comparison count in **all** dual-pivot algorithms
- Unified analysis
- “Asymptotically optimal” algorithms

Part 1

- Model to capture comparison count in **all** dual-pivot algorithms
- Unified analysis
- “Asymptotically optimal” algorithms

Reduce Sorting Cost C_n to Partitioning Cost P_n

Reduce Sorting Cost C_n to Partitioning Cost P_n

Dual-pivot quicksort recurrence: Let

C_n = number of comparisons for sorting n elements

P_n = number of comparisons for partitioning n elements

Reduce Sorting Cost C_n to Partitioning Cost P_n

Dual-pivot quicksort recurrence: Let

C_n = number of comparisons for sorting n elements

P_n = number of comparisons for partitioning n elements

Then:

Reduce Sorting Cost C_n to Partitioning Cost P_n

Dual-pivot quicksort recurrence: Let

C_n = number of comparisons for sorting n elements

P_n = number of comparisons for partitioning n elements

Then:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k) \mathbb{E}(C_k).$$

Reduce Sorting Cost C_n to Partitioning Cost P_n

Dual-pivot quicksort recurrence: Let

C_n = number of comparisons for sorting n elements

P_n = number of comparisons for partitioning n elements

Then:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k) \mathbb{E}(C_k).$$

(Recall $C_n = \mathbb{E}(P_n) + \frac{2}{n} \sum_{k=1}^{n-1} \mathbb{E}(C_k)$ for one pivot.)

Reduce Sorting Cost C_n to Partitioning Cost P_n

Dual-pivot quicksort recurrence: Let

C_n = number of comparisons for sorting n elements

P_n = number of comparisons for partitioning n elements

Then:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k) \mathbb{E}(C_k).$$

(Recall $C_n = \mathbb{E}(P_n) + \frac{2}{n} \sum_{k=1}^{n-1} \mathbb{E}(C_k)$ for one pivot.)

Hennequin (1991) solves recurrence for “toll function” $\mathbb{E}(P_n) = an + b$ (and much more).

Wild/Nebel/Neininger (2012/2015) solve it for P_n induced by (simplified version of) YBB.

Reduce Sorting Cost C_n to Partitioning Cost P_n

Fact (Hennequin (1991), simplified)

Average partitioning cost of $\mathbb{E}(P_n) = a \cdot n + O(1)$ leads to average sorting cost $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof: Uses generating function techniques. Btw: $\frac{6}{5} = (\frac{1}{2} + \frac{1}{3})^{-1}$.)

Reduce Sorting Cost C_n to Partitioning Cost P_n

Fact (Hennequin (1991), simplified)

Average partitioning cost of $\mathbb{E}(P_n) = a \cdot n + O(1)$ leads to average sorting cost $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof: Uses generating function techniques. Btw: $\frac{6}{5} = (\frac{1}{2} + \frac{1}{3})^{-1}$.)

Slightly more general (also in Martínez, Nebel, Wild (2014)):

If $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

Reduce Sorting Cost C_n to Partitioning Cost P_n

Fact (Hennequin (1991), simplified)

Average partitioning cost of $\mathbb{E}(P_n) = a \cdot n + O(1)$ leads to average sorting cost $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof: Uses generating function techniques. Btw: $\frac{6}{5} = (\frac{1}{2} + \frac{1}{3})^{-1}$.)

Slightly more general (also in Martínez, Nebel, Wild (2014)):

If $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof uses Roura's "Continuous Master Theorem" from 2001.)

Reduce Sorting Cost C_n to Partitioning Cost P_n

Fact (Hennequin (1991), simplified)

Average partitioning cost of $\mathbb{E}(P_n) = a \cdot n + O(1)$ leads to average sorting cost $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof: Uses generating function techniques. Btw: $\frac{6}{5} = (\frac{1}{2} + \frac{1}{3})^{-1}$.)

Slightly more general (also in Martínez, Nebel, Wild (2014)):

If $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then $\mathbb{E}(C_n) = \frac{6}{5}a \cdot n \ln n + O(n)$.

(Proof uses Roura's "Continuous Master Theorem" from 2001.)

So: What is the linear term $a \cdot n$ in $\mathbb{E}(P_n)$?

The Partitioning Cost

Must **classify** $n - 2$ entries x into three parts:



small, medium, or large

The Partitioning Cost

Must **classify** $n - 2$ entries x into three parts:



small, medium, or large

1 or 2 comparisons for x .

The Partitioning Cost

Must **classify** $n - 2$ entries x into three parts:



small, **medium**, or **large**

1 or 2 comparisons for x .

Unavoidable: 1 comparison for small/large x , 2 comparisons for medium x .

The Partitioning Cost

Must **classify** $n - 2$ entries x into three parts:



small, medium, or large

1 or 2 comparisons for x .

Unavoidable: 1 comparison for small/large x , 2 comparisons for medium x .

Extra: small x compared with q first and large x compared with p first.

The Partitioning Cost

Must **classify** $n - 2$ entries x into three parts:



small , medium , or large

1 or 2 comparisons for x .

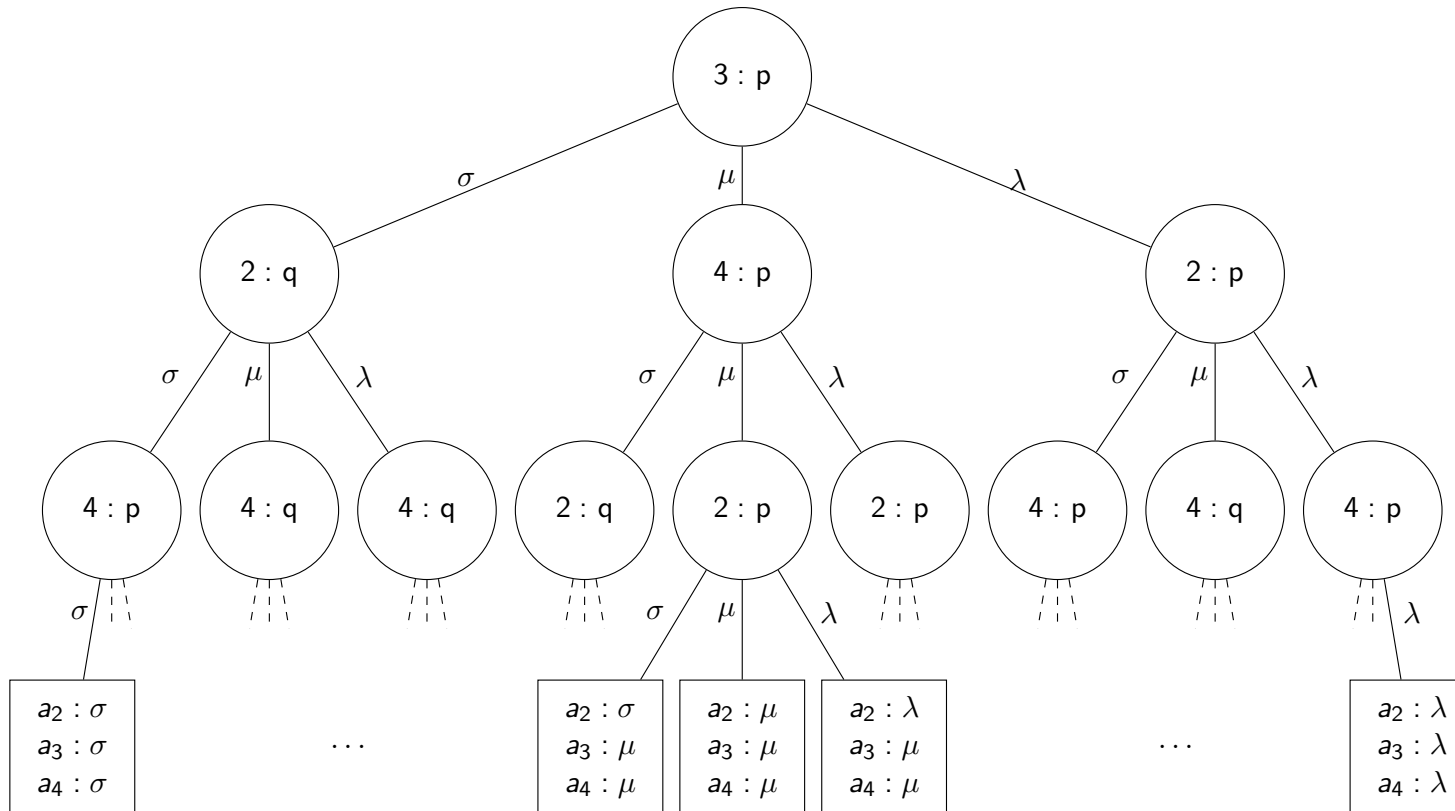
Unavoidable: 1 comparison for small/large x , 2 comparisons for medium x .

Extra: small x compared with q first and large x compared with p first.

Partitioning strategy determines for next element x whether to compare x with p first or with q first.

Program text (YBB/Sedgewick/...) *implicitly* defines strategy.

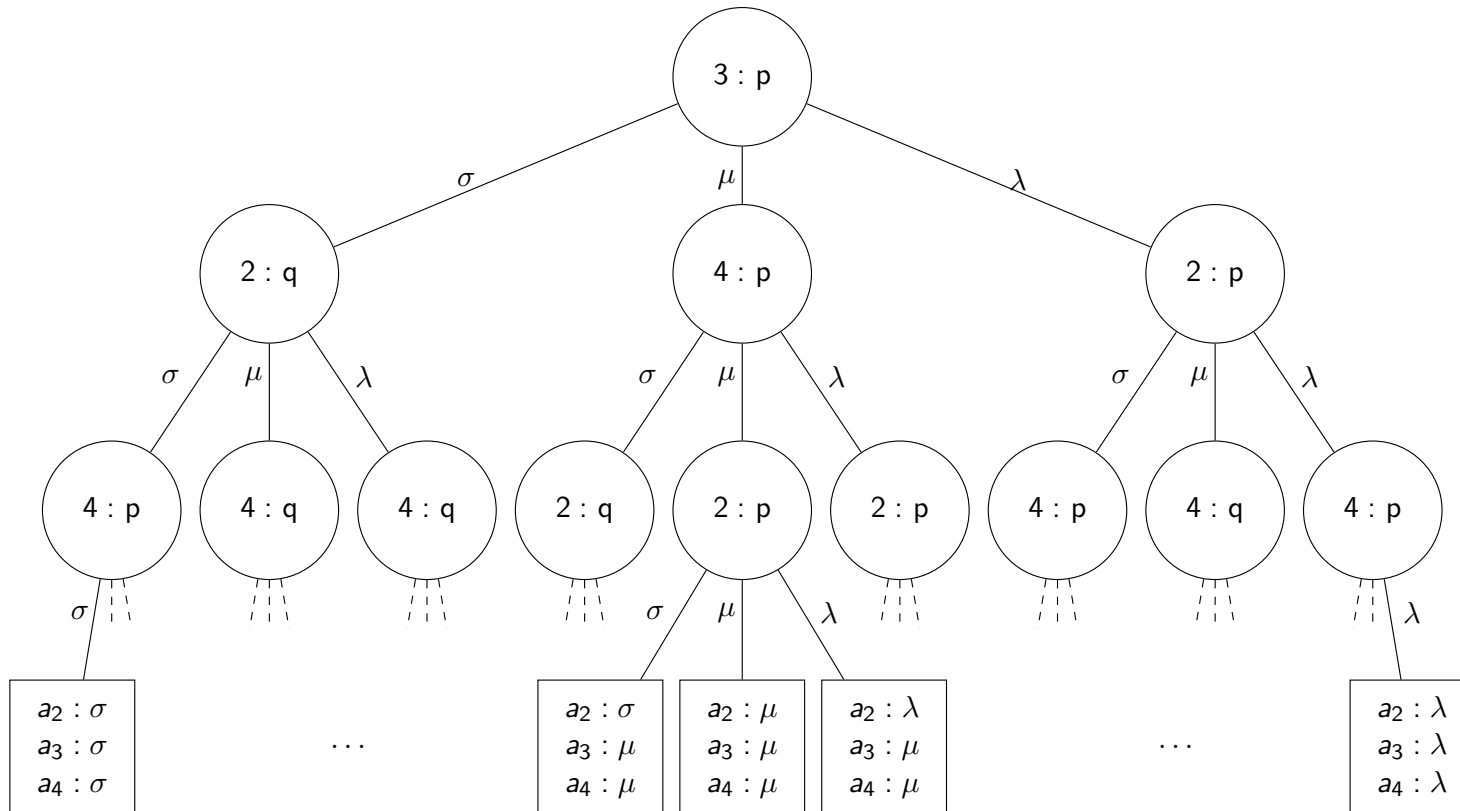
Classification Tree: Models Strategy



Example:

3 4 2 1 5

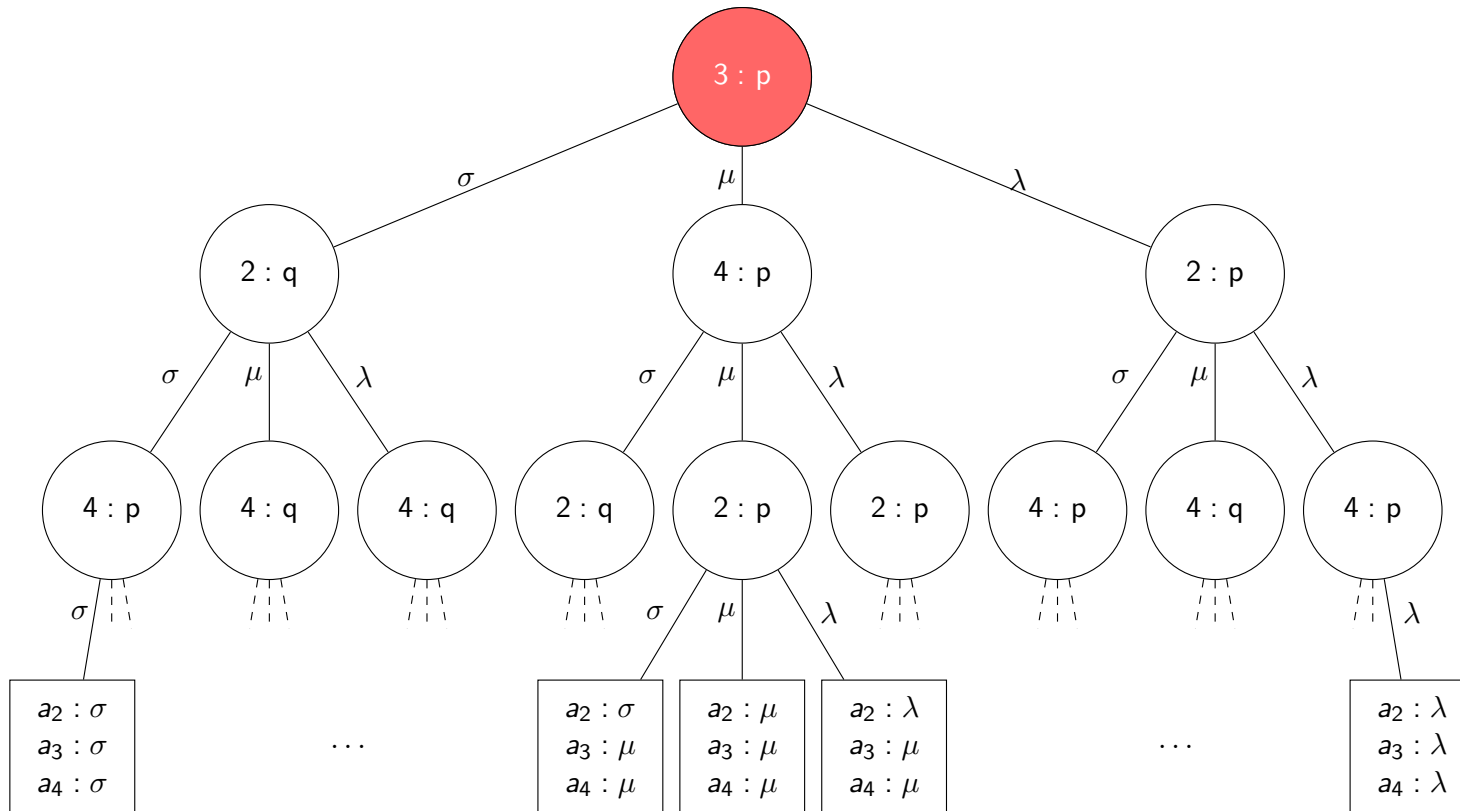
Classification Tree: Models Strategy



Example:

3	4	2	1	5
p				q

Classification Tree: Models Strategy



Example:

3

4

2

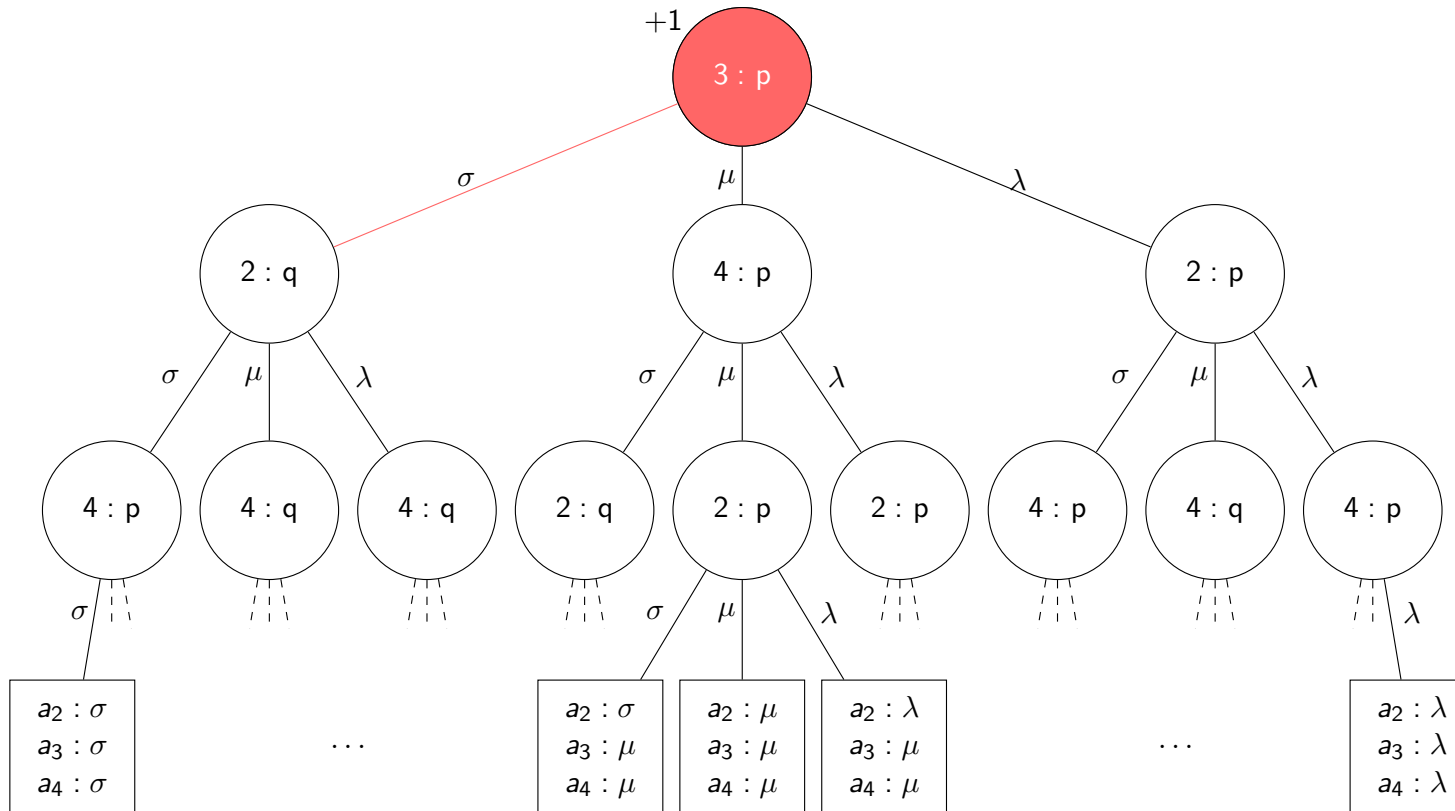
1

5

p

q

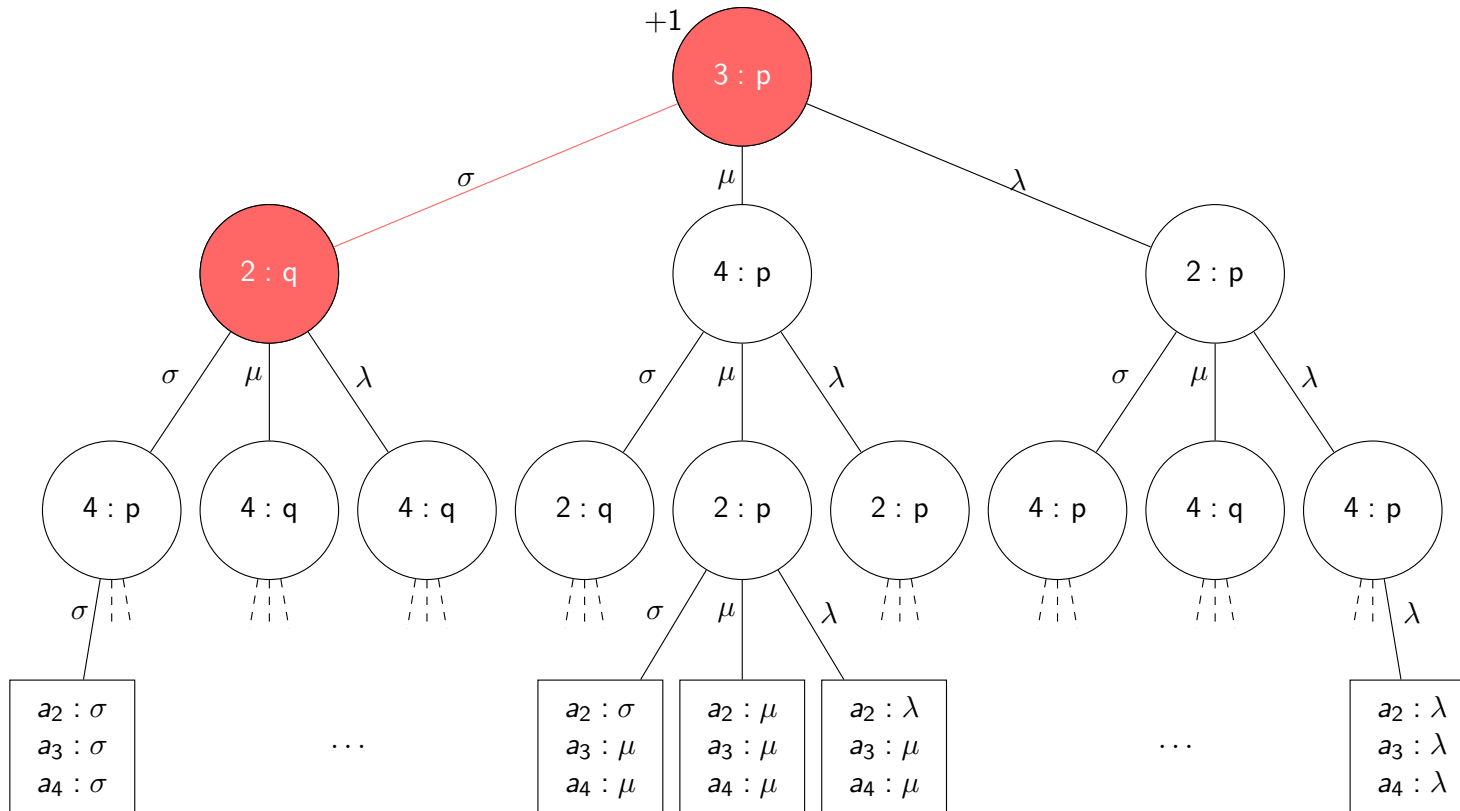
Classification Tree: Models Strategy



Example:

3	4	2	1	5
p		σ		q

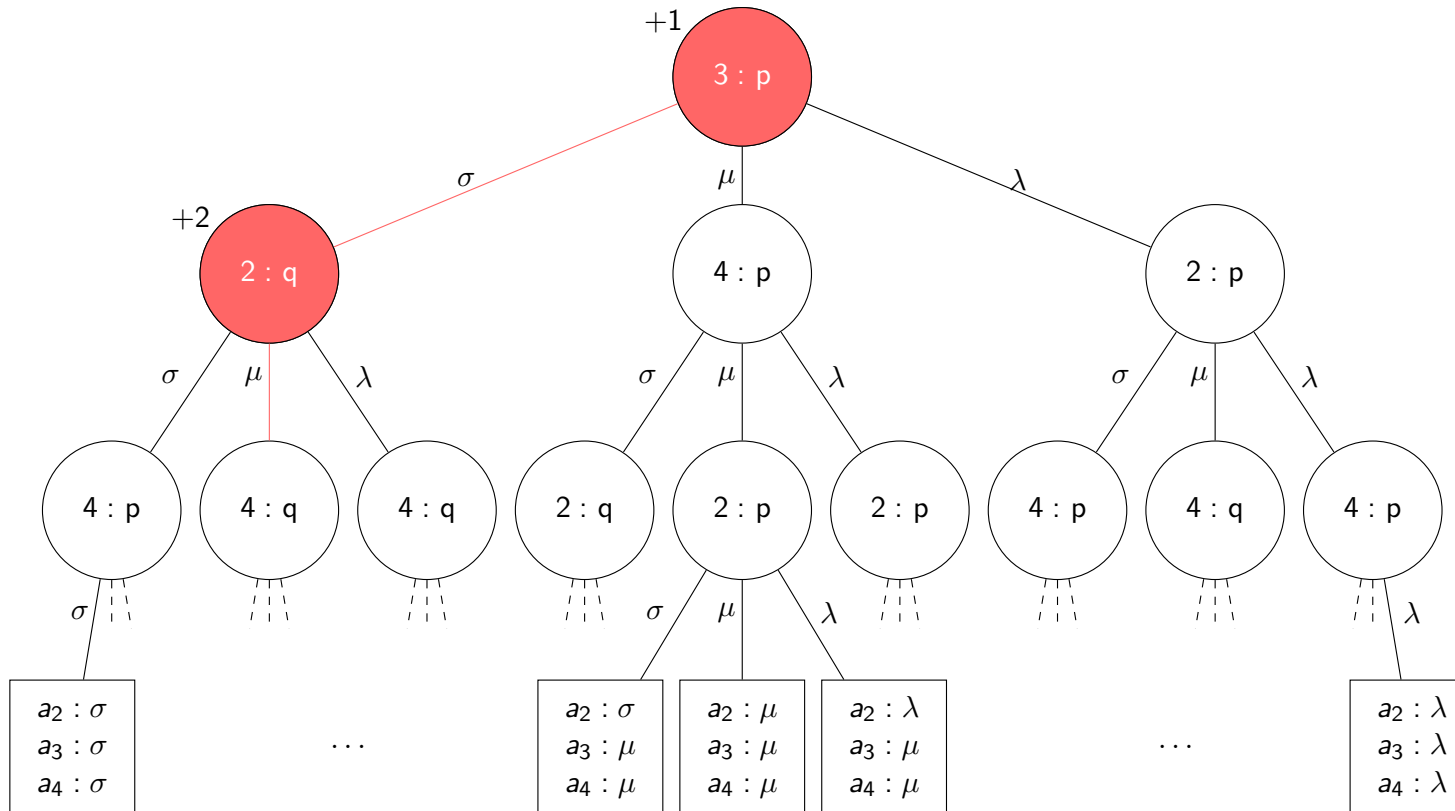
Classification Tree: Models Strategy



Example:

3	4	2	1	5
p		σ		q

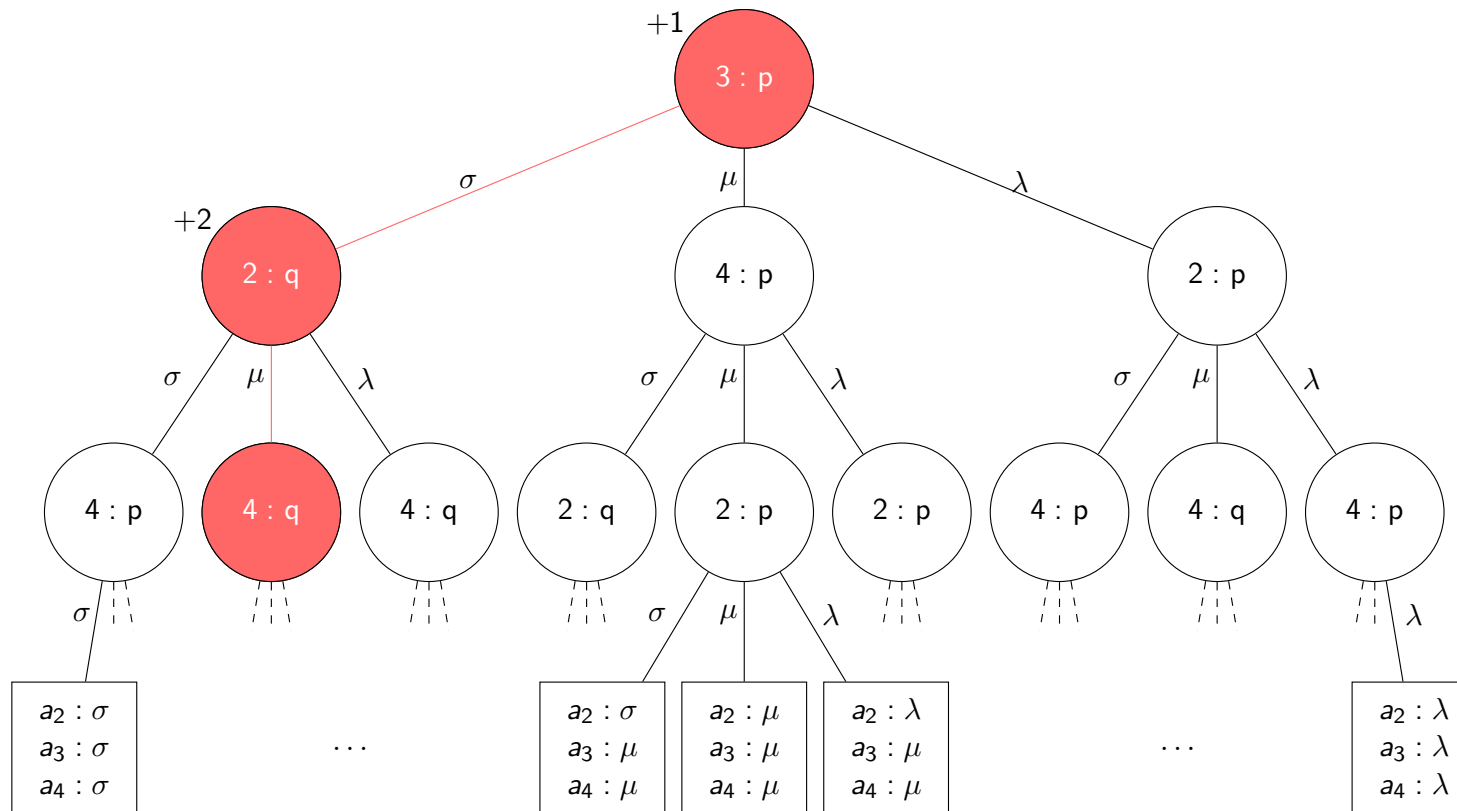
Classification Tree: Models Strategy



Example:

3	4	2	1	5
p	μ	σ		q

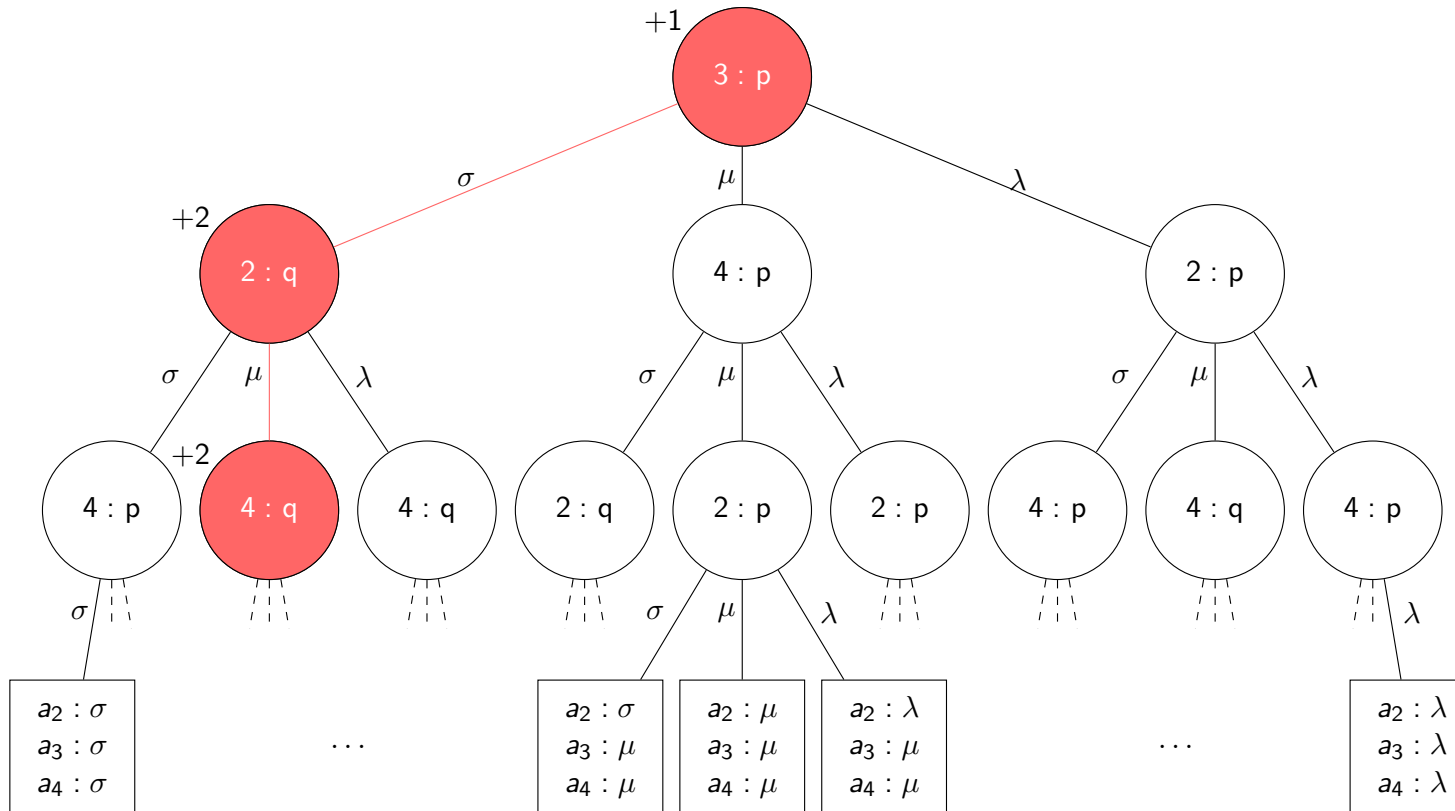
Classification Tree: Models Strategy



Example:

3	4	2	1	5
ρ	μ	σ		q

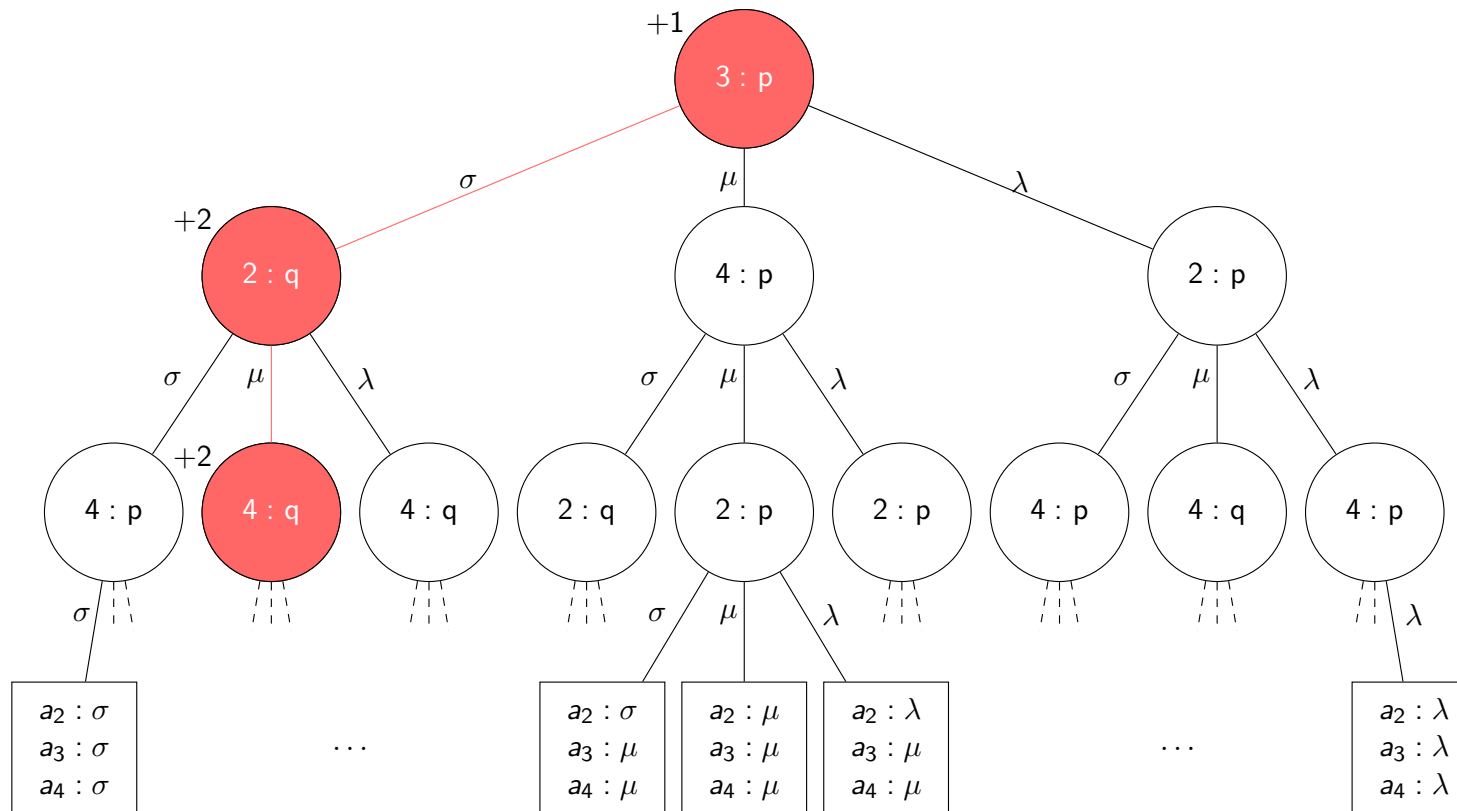
Classification Tree: Models Strategy



Example:

3	4	2	1	5
ρ	μ	σ	σ	q

Classification Tree: Models Strategy



Example:

3	4	2	1	5
ρ	μ	σ	σ	q

\Rightarrow 5 comparisons.

Average Cost

For the average partitioning/classification cost $\mathbb{E}(P_n)$ we get:

$$\mathbb{E}(P_n) \approx \frac{4}{3n} + \text{“average number of extra comparisons”}$$

extra :

- small x compared to q first
- large x compared to p first

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

If v is labelled \mathbf{p} , then
contribution to average number
of extra comparisons is:

$$p_v \cdot \frac{\ell - \ell_v}{n - \text{level}(v)} \approx p_v \cdot \frac{\ell}{n - 2}.$$

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

If v is labelled \mathbf{p} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{\ell - \ell_v}{n - \text{level}(v)} \approx p_v \cdot \frac{\ell}{n - 2}.$$

If v is labelled \mathbf{q} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{s - s_v}{n - \text{level}(v)} \approx p_v \cdot \frac{s}{n - 2}.$$

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

If v is labelled \mathbf{p} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{\ell - \ell_v}{n - \text{level}(v)} \approx p_v \cdot \frac{\ell}{n - 2}.$$

If v is labelled \mathbf{q} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{s - s_v}{n - \text{level}(v)} \approx p_v \cdot \frac{s}{n - 2}.$$

“ \approx ” can be justified up to small error.

Extra Comparisons in the Classification Tree

v : node in classification tree.

Fix s , ℓ , the number of small resp. large elements.

(Apart from that: input random.)

- p_v : probability that node v is reached.
- s_v : number of “small” elements seen on path to v .
- ℓ_v : number of “large” elements seen on path to v .

If v is labelled \mathbf{p} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{\ell - \ell_v}{n - \text{level}(v)} \approx p_v \cdot \frac{\ell}{n - 2}.$$

If v is labelled \mathbf{q} , then contribution to average number of extra comparisons is:

$$p_v \cdot \frac{s - s_v}{n - \text{level}(v)} \approx p_v \cdot \frac{s}{n - 2}.$$

“ \approx ” can be justified up to small error.

Cost of an Arbitrary Classification Tree

Average number of comparisons to larger/smaller pivot first, given s, ℓ :

$$f_{s,\ell}^q = \sum_{v \text{ is q-node}} p_v = \mathbb{E}(\#(\text{q-nodes reached}) \mid s, \ell)$$

$$f_{s,\ell}^p = \sum_{v \text{ is p-node}} p_v = \mathbb{E}(\#(\text{p-nodes reached}) \mid s, \ell).$$

Cost of an Arbitrary Classification Tree

Average number of comparisons to larger/smaller pivot first, given s, ℓ :

$$f_{s,\ell}^q = \sum_{v \text{ is q-node}} p_v = \mathbb{E}(\#(\text{q-nodes reached}) \mid s, \ell)$$

$$f_{s,\ell}^p = \sum_{v \text{ is p-node}} p_v = \mathbb{E}(\#(\text{p-nodes reached}) \mid s, \ell).$$

Lemma

Average comparison cost for classification:

$$\mathbb{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2}(n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + f_{s,\ell}^p \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Cost of an Arbitrary Classification Tree

Average number of comparisons to larger/smaller pivot first, given s, ℓ :

$$f_{s,\ell}^q = \sum_{v \text{ is q-node}} p_v = \mathbb{E}(\#(\text{q-nodes reached}) \mid s, \ell)$$

$$f_{s,\ell}^p = \sum_{v \text{ is p-node}} p_v = \mathbb{E}(\#(\text{p-nodes reached}) \mid s, \ell).$$

Lemma

Average comparison cost for classification:

$$\mathbb{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2}(n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + f_{s,\ell}^p \cdot \ell \right) + O(n^{1-\varepsilon}).$$

(*Proof:* Method of bounded differences:

Behaviour of differences as expected w.h.p. in most levels of the tree.)

Cost of an Arbitrary Classification Tree

Average number of comparisons to larger/smaller pivot first, given s, ℓ :

$$f_{s,\ell}^q = \sum_{v \text{ is q-node}} p_v = \mathbb{E}(\#(\text{q-nodes reached}) \mid s, \ell)$$

$$f_{s,\ell}^p = \sum_{v \text{ is p-node}} p_v = \mathbb{E}(\#(\text{p-nodes reached}) \mid s, \ell).$$

Lemma

Average comparison cost for classification:

$$\mathbb{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2}(n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + f_{s,\ell}^p \cdot \ell \right) + O(n^{1-\varepsilon}).$$

(*Proof:* Method of bounded differences:

Behaviour of differences as expected w.h.p. in most levels of the tree.)

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

$$\mathbb{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \cdot \sum_{s+\ell \leq n-2} (f_n^q \cdot s + (n-2-f_n^q) \cdot \ell) + O(n^{1-\varepsilon})$$

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

$$\mathbb{E}(P_n) = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \cdot \sum_{s+l \leq n-2} (f_n^q \cdot s + (n-2-f_n^q) \cdot l) + O(n^{1-\varepsilon})$$

(symmetry)
=

$$\frac{4}{3}n + \frac{1}{\binom{n}{2} (n-2)} \cdot \sum_{s+l \leq n-2} s(n-2) + O(n^{1-\varepsilon})$$

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

$$\begin{aligned}\mathbb{E}(P_n) &= \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \cdot \sum_{s+l \leq n-2} (f_n^q \cdot s + (n-2-f_n^q) \cdot l) + O(n^{1-\varepsilon}) \\ &\stackrel{\text{(symmetry)}}{=} \frac{4}{3}n + \frac{1}{\binom{n}{2} (n-2)} \cdot \sum_{s+l \leq n-2} s(n-2) + O(n^{1-\varepsilon}) \\ &= \frac{5}{3}n + O(n^{1-\varepsilon}).\end{aligned}$$

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

$$\begin{aligned}\mathbb{E}(P_n) &= \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \cdot \sum_{s+l \leq n-2} (f_n^q \cdot s + (n-2-f_n^q) \cdot l) + O(n^{1-\varepsilon}) \\ &\stackrel{\text{(symmetry)}}{=} \frac{4}{3}n + \frac{1}{\binom{n}{2} (n-2)} \cdot \sum_{s+l \leq n-2} s(n-2) + O(n^{1-\varepsilon}) \\ &= \frac{5}{3}n + O(n^{1-\varepsilon}). \quad \text{Hence:}\end{aligned}$$

Analyzing Strategies

Oblivious Strategies

Ignore results of previous comparisons,
all nodes on one level use the same pivot first.

Examples: Always q first, Alternate, Random.

$$\begin{aligned}\mathbb{E}(P_n) &= \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \cdot \sum_{s+l \leq n-2} (f_n^q \cdot s + (n-2-f_n^q) \cdot l) + O(n^{1-\varepsilon}) \\ &\stackrel{\text{(symmetry)}}{=} \frac{4}{3}n + \frac{1}{\binom{n}{2}(n-2)} \cdot \sum_{s+l \leq n-2} s(n-2) + O(n^{1-\varepsilon}) \\ &= \frac{5}{3}n + O(n^{1-\varepsilon}). \quad \text{Hence:}\end{aligned}$$

$$\mathbb{E}(C_n) = \frac{6}{5} \cdot \frac{5}{3}n \ln n + O(n) = 2n \ln n + O(n).$$

Analyzing Strategies

Strategy from YBB algorithm

Whenever large entry has been seen, the next comparison is with q first.

$$f_{s,l}^q = l \text{ and } f_{s,l}^p = s + m = n - 2 - l.$$

Analyzing Strategies

Strategy from YBB algorithm

Whenever large entry has been seen, the next comparison is with q first.

$$f_{s,l}^q = l \text{ and } f_{s,l}^p = s + m = n - 2 - l.$$

$$\mathbb{E}(P_n^y) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+l \leq n-2} \left(\frac{sl}{n-2} + \frac{(s+m)l}{n-2} \right) + O(n^{1-\varepsilon})$$

Analyzing Strategies

Strategy from YBB algorithm

Whenever large entry has been seen, the next comparison is with q first.

$$f_{s,l}^q = l \text{ and } f_{s,l}^p = s + m = n - 2 - l.$$

$$\begin{aligned} \mathbb{E}(P_n^y) &= \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+l \leq n-2} \left(\frac{sl}{n-2} + \frac{(s+m)l}{n-2} \right) + O(n^{1-\varepsilon}) \\ &= \frac{19}{12}n + O(n^{1-\varepsilon}). \end{aligned}$$

Analyzing Strategies

Strategy from YBB algorithm

Whenever large entry has been seen, the next comparison is with q first.

$$f_{s,l}^q = l \text{ and } f_{s,l}^p = s + m = n - 2 - l.$$

$$\begin{aligned} \mathbb{E}(P_n^y) &= \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+l \leq n-2} \left(\frac{sl}{n-2} + \frac{(s+m)l}{n-2} \right) + O(n^{1-\varepsilon}) \\ &= \frac{19}{12}n + O(n^{1-\varepsilon}). \end{aligned} \quad \text{Hence:}$$

$$\mathbb{E}(C_n^y) = \frac{6}{5} \cdot \frac{19}{12}n \ln n + O(n) = 1.9n \ln n + O(n).$$

Analyzing Strategies

Strategy from Sedgewick's Algorithm

$$f_{s,\ell}^q = (n - 2) \cdot s / (s + \ell) \text{ and } f_{s,\ell}^p = (n - 2) \cdot \ell / (s + \ell)$$

Analyzing Strategies

Strategy from Sedgewick's Algorithm

$$f_{s,l}^q = (n-2) \cdot s/(s+l) \text{ and } f_{s,l}^p = (n-2) \cdot l/(s+l)$$

$$\mathbb{E}(P_n^S) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+l \leq n-2} \left(\frac{s^2}{s+l} + \frac{l^2}{s+l} \right) + O(n^{1-\varepsilon}) = \frac{16}{9}n + O(n^{1-\varepsilon})$$

Analyzing Strategies

Strategy from Sedgewick's Algorithm

$$f_{s,\ell}^q = (n-2) \cdot s/(s+\ell) \text{ and } f_{s,\ell}^p = (n-2) \cdot \ell/(s+\ell)$$

$$\mathbb{E}(P_n^S) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \left(\frac{s^2}{s+\ell} + \frac{\ell^2}{s+\ell} \right) + O(n^{1-\varepsilon}) = \frac{16}{9}n + O(n^{1-\varepsilon})$$

$$\mathbb{E}(C_n^S) = \frac{6}{5} \cdot \frac{16}{9}n \ln n + O(n) = 2.133.. \cdot n \ln n + O(n). \text{ (Also: (NW 2012).)}$$

Analyzing Strategies

Strategy from Sedgwick's Algorithm

$$f_{s,\ell}^q = (n-2) \cdot s/(s+\ell) \text{ and } f_{s,\ell}^p = (n-2) \cdot \ell/(s+\ell)$$

$$\mathbb{E}(P_n^S) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \left(\frac{s^2}{s+\ell} + \frac{\ell^2}{s+\ell} \right) + O(n^{1-\varepsilon}) = \frac{16}{9}n + O(n^{1-\varepsilon})$$

$$\mathbb{E}(C_n^S) = \frac{6}{5} \cdot \frac{16}{9}n \ln n + O(n) = 2.133.. \cdot n \ln n + O(n). \text{ (Also: (NW 2012).)}$$

Simple improvement (also observed by Wild):

In Sedgwick's algorithm, switch p and q in choice for first pivot.

Analyzing Strategies

Strategy from Sedgwick's Algorithm

$$f_{s,\ell}^q = (n-2) \cdot s/(s+\ell) \text{ and } f_{s,\ell}^p = (n-2) \cdot \ell/(s+\ell)$$

$$\mathbb{E}(P_n^S) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \left(\frac{s^2}{s+\ell} + \frac{\ell^2}{s+\ell} \right) + O(n^{1-\varepsilon}) = \frac{16}{9}n + O(n^{1-\varepsilon})$$

$$\mathbb{E}(C_n^S) = \frac{6}{5} \cdot \frac{16}{9}n \ln n + O(n) = 2.133\dots \cdot n \ln n + O(n). \text{ (Also: (NW 2012).)}$$

Simple improvement (also observed by Wild):

In Sedgwick's algorithm, switch p and q in choice for first pivot.

$$\mathbb{E}(C_n^{S'}) = \frac{6}{5} \cdot \frac{14}{9}n \ln n + O(n) = 1.866\dots \cdot n \ln n + O(n).$$

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + (n-2-f_{s,\ell}^q) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + (n-2-f_{s,\ell}^q) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Strategy

- $\ell > s$: Compare all elements to larger pivot first ($f_{s,\ell}^q = n-2$).

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + (n-2-f_{s,\ell}^q) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Strategy

- $\ell > s$: Compare all elements to larger pivot first ($f_{s,\ell}^q = n-2$).
- $\ell \leq s$: Compare all elements to smaller pivot first ($f_{s,\ell}^q = 0$).

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + (n-2-f_{s,\ell}^q) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Strategy

- $\ell > s$: Compare all elements to larger pivot first ($f_{s,\ell}^q = n-2$).
- $\ell \leq s$: Compare all elements to smaller pivot first ($f_{s,\ell}^q = 0$).

$$\mathbb{E}(P_n^I) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \min\{s, \ell\} + O(n^{1-\varepsilon}) = \frac{3}{2}n + O(n^{1-\varepsilon}).$$

Almost Optimal Strategy (with Oracle)

Assume: Given input and pivots, an oracle tells us whether or not $\ell > s$.

Goal: Minimize

$$p_n = \frac{4}{3}n + \frac{1}{\binom{n}{2} \cdot (n-2)} \sum_{s+\ell \leq n-2} \left(f_{s,\ell}^q \cdot s + (n-2-f_{s,\ell}^q) \cdot \ell \right) + O(n^{1-\varepsilon}).$$

Strategy

- $\ell > s$: Compare all elements to larger pivot first ($f_{s,\ell}^q = n-2$).
- $\ell \leq s$: Compare all elements to smaller pivot first ($f_{s,\ell}^q = 0$).

$$\mathbb{E}(P_n^I) = \frac{4}{3}n + \frac{1}{\binom{n}{2}} \sum_{s+\ell \leq n-2} \min\{s, \ell\} + O(n^{1-\varepsilon}) = \frac{3}{2}n + O(n^{1-\varepsilon}).$$

$$\mathbb{E}(C_n^I) = \frac{6}{5} \cdot \frac{3}{2}n \ln n + O(n) = 1.8n \ln n + O(n).$$

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons on average.

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons on average.

Implementation?

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons on average.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons on average.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

With Chernoff-Hoeffding type bounds: Probability to guess wrong is small.

Idealized, but gives lower bound:

No strategy can use fewer than $1.8n \ln n - O(n)$ comparisons on average.

Implementation?

Random Sampling (read $n^{3/4}$ entries) to estimate if $s > \ell$ or $s \leq \ell$.

With Chernoff-Hoeffding type bounds: Probability to guess wrong is small.

Average cost of random sampling algorithm: $1.8n \ln n + O(n)$.

Part 2

Part 2

- Find a *truly optimal* algorithm.

Part 2

- Find a *truly optimal* algorithm.
- Do an exact analysis.

Part 2

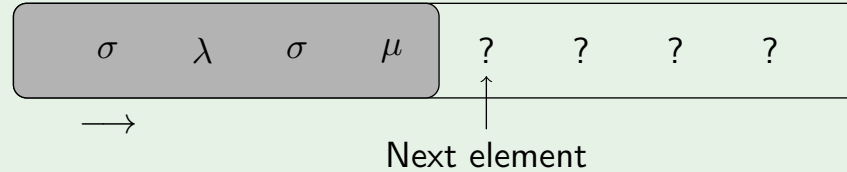
- Find a *truly optimal* algorithm.
- Do an exact analysis.


(From (ADHKP16).)

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i /level i of the tree.

Strategy "Count"

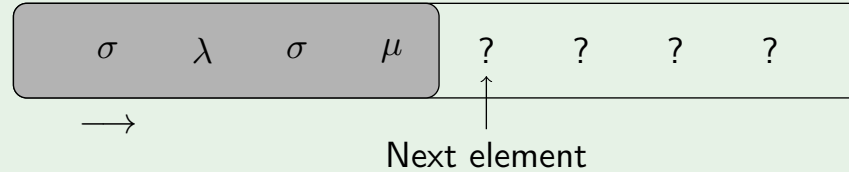



Seen s_{i-1} small and ℓ_{i-1} large elements in 

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i /level i of the tree.

Strategy “Count”



Seen s_{i-1} small and l_{i-1} large elements in 

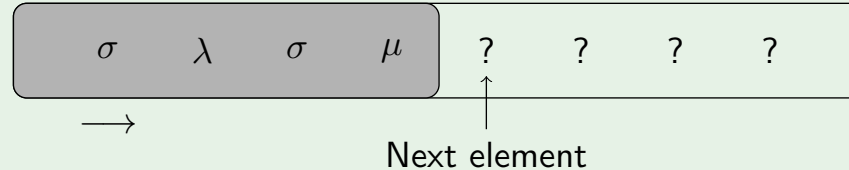
Classification Strategy:


- $l_{i-1} > s_{i-1}$: compare with larger pivot first.
- $l_{i-1} \leq s_{i-1}$: compare with smaller pivot first.

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i /level i of the tree.

Strategy “Count”



Seen s_{i-1} small and l_{i-1} large elements in 

Classification Strategy:

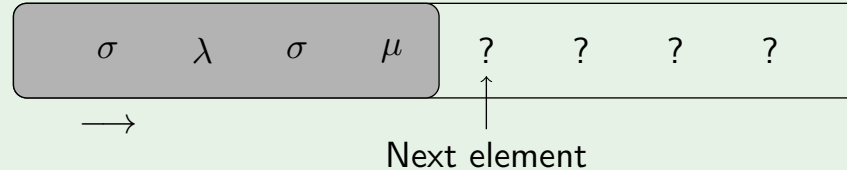
- $l_{i-1} > s_{i-1}$: compare with larger pivot first.
- $l_{i-1} \leq s_{i-1}$: compare with smaller pivot first.


Can show (AD13/16): Average sorting cost is $1.8n \ln n + O(n)$.
($O(n)$ away from (idealized) optimal strategy.)

The Comparison-Optimal Dual-Pivot Quicksort Algorithm

Assume we are in round i /level i of the tree.

Strategy “Count”



Seen s_{i-1} small and ℓ_{i-1} large elements in 

Classification Strategy:

- $\ell_{i-1} > s_{i-1}$: compare with larger pivot first.
- $\ell_{i-1} \leq s_{i-1}$: compare with smaller pivot first.

Can show (AD13/16): Average sorting cost is $1.8n \ln n + O(n)$.

($O(n)$ away from (idealized) optimal strategy.)

Now:

“Count” is optimal + exact average comparison count (ADHKP16/17).

Proof idea for: “Count” is optimal (among all algorithms)

Distribution on $\{\sigma, \mu, \lambda\}$ -sequences generated by input and two random pivots is given by:

Each fixed sequence with s many σ 's, m many μ 's, ℓ many λ 's, with $s + m + \ell = n - 2$ appears with probability

$$\frac{1}{\binom{n}{2}} \cdot \frac{s!m!\ell!}{(n-2)!}$$

Proof idea for: “Count” is optimal (among all algorithms)

Distribution on $\{\sigma, \mu, \lambda\}$ -sequences generated by input and two random pivots is given by:

Each fixed sequence with s many σ 's, m many μ 's, ℓ many λ 's, with $s + m + \ell = n - 2$ appears with probability

$$\frac{1}{\binom{n}{2}} \cdot \frac{s!m!\ell!}{(n-2)!}$$

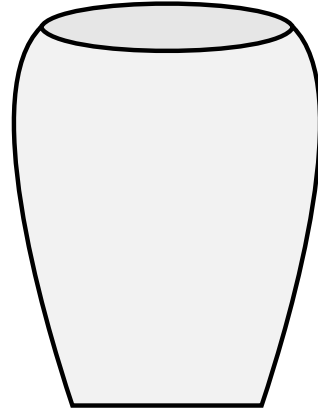
Reason:

Probability to have s many σ 's, m many μ 's, ℓ many λ 's is $1/\binom{n}{2}$.

Every sequence with s many σ 's, m many μ 's, ℓ many λ 's has the same probability $\frac{1}{\binom{n-2}{s,m,\ell}}$.

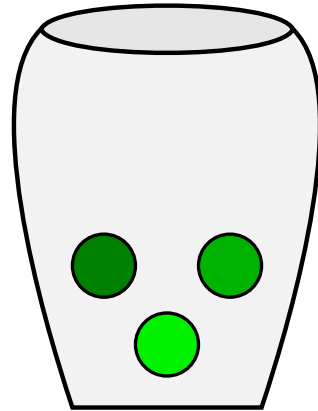
Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

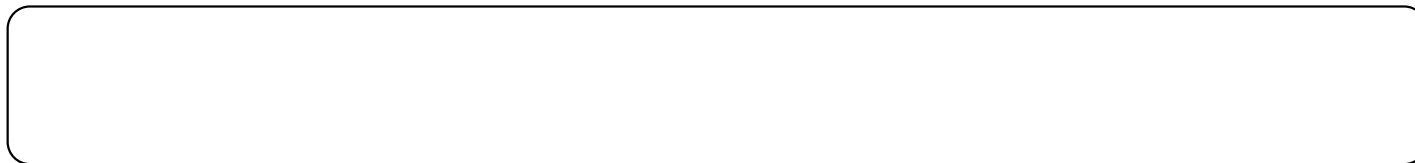


Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

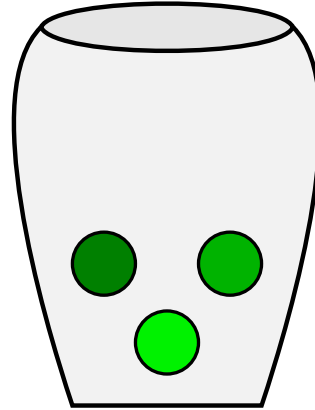


- 1 Put one light green ball, one green ball, one dark green ball in urn.



Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

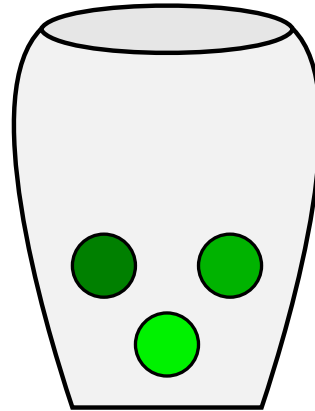


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:



Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

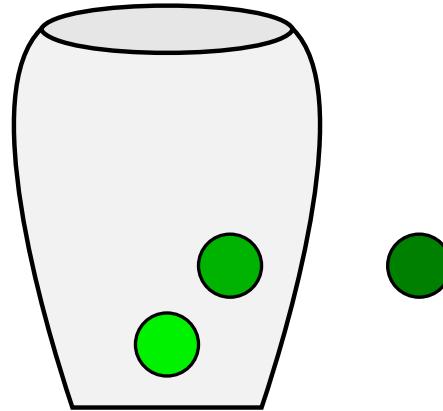


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random.



Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

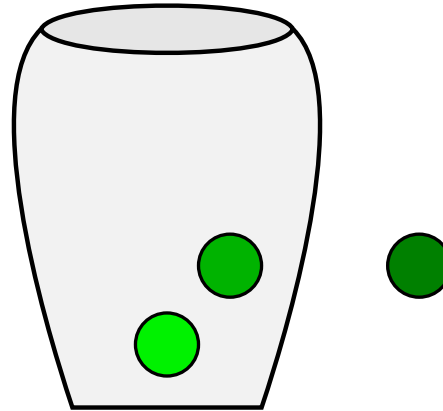


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random.



Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

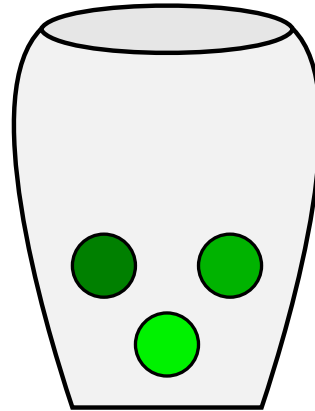


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back,

λ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

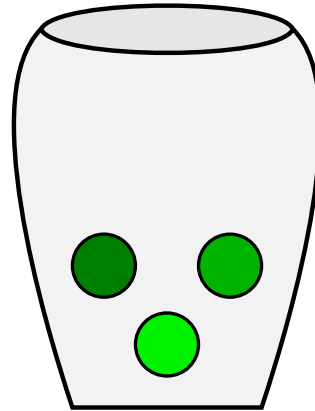


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back,

λ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

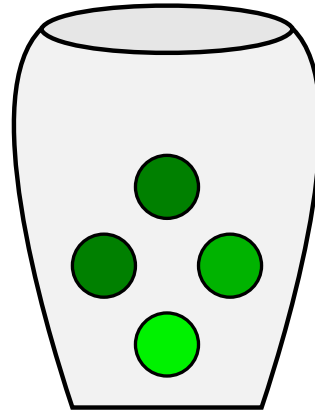


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

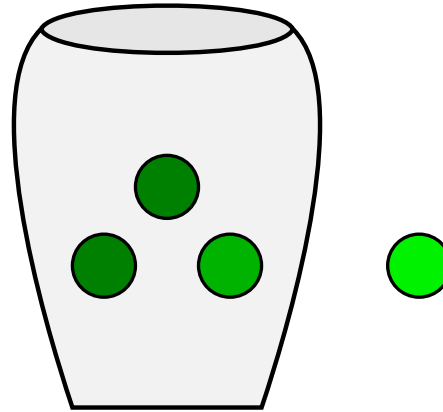


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

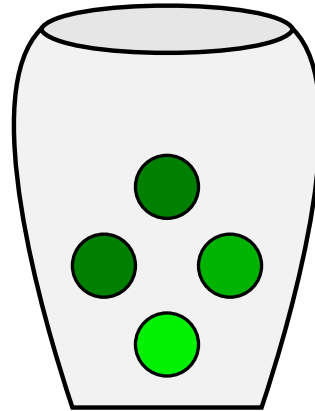


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

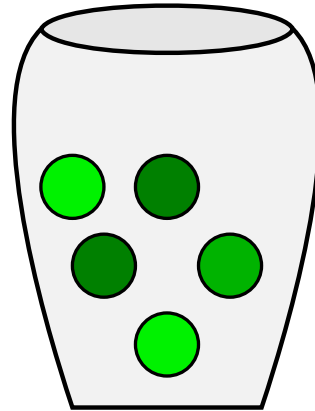


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

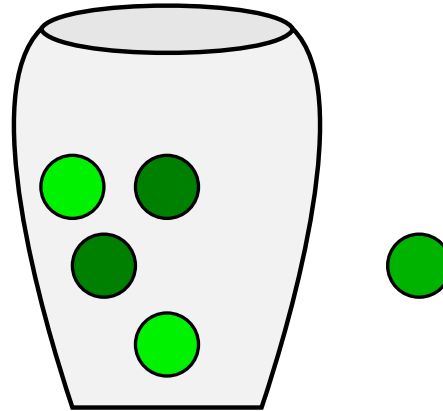


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

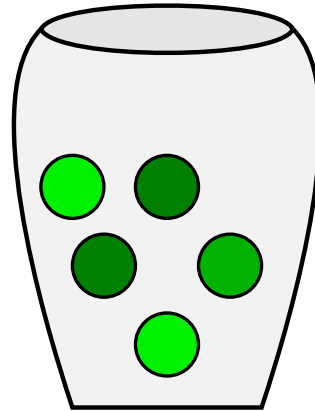


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

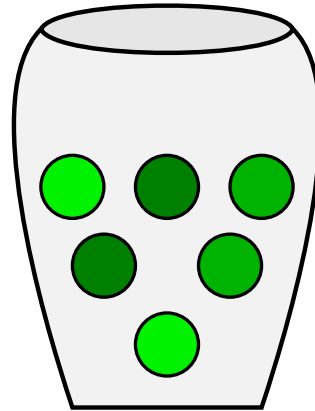


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

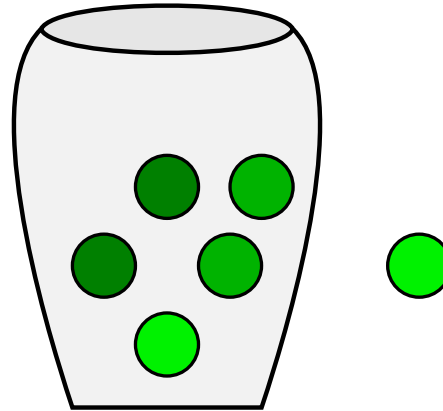


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

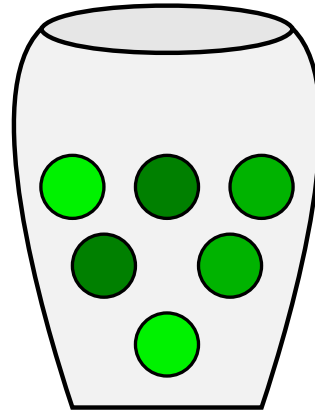


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

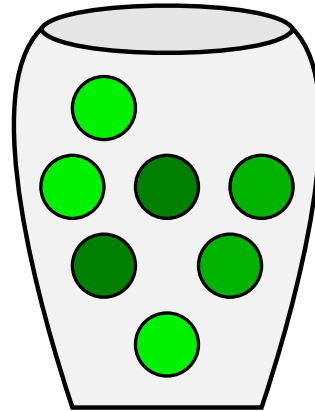


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.

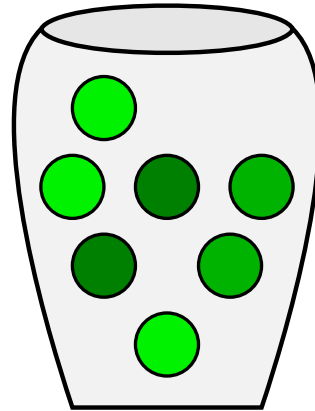


- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ σ

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.



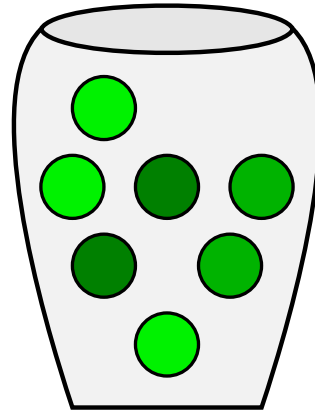
- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

λ σ μ σ

Why same distribution as “random permutation \rightarrow pivots \rightarrow relabel”?

Proof idea for: “Count” is optimal (among all algorithms)

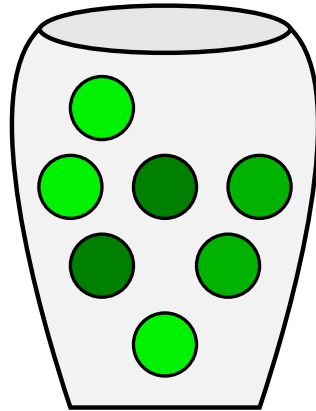
Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.



- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

Proof idea for: “Count” is optimal (among all algorithms)

Obtain same distribution on $\{\sigma, \mu, \lambda\}$ -sequences by **Pólya urn** with three colors.



- 1 Put one light green ball, one green ball, one dark green ball in urn.
- 2 Round $i = 1, \dots, n - 2$:
Choose ball from urn at random. Take down its color c , put it back, and put another ball of the same color in the urn.

In Round i :

$$\Pr(\text{new element is } \sigma \text{ (small)}) = \frac{s_{i-1} + 1}{i + 2}.$$

(analogous formulas for medium/large elements.) Then: induction!

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$.

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$. “Count” compares with small pivot first.

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$. “Count” compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{“Count” gets extra comparison in step } i) = \frac{\ell_{i-1} + 1}{i + 2}.$$

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$. “Count” compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{“Count” gets extra comparison in step } i) = \frac{\ell_{i-1} + 1}{i + 2}.$$

If \mathcal{S} (based on full history up to $i - 1$, even using randomness) uses small pivot first, no difference.

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$. “Count” compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{“Count” gets extra comparison in step } i) = \frac{\ell_{i-1} + 1}{i + 2}.$$

If \mathcal{S} (based on full history up to $i - 1$, even using randomness) uses small pivot first, no difference.

If \mathcal{S} takes large pivot first:

$$\Pr(\mathcal{S} \text{ gets extra comparison in step } i) = \frac{s_{i-1} + 1}{i + 2},$$

“Count” is optimal (among all algorithms)

Claim: Every partitioning strategy \mathcal{S} (decision in round i is based on the full history up to round $i - 1$) makes at least as many comparisons in round i as “Count” (on average).

Proof: Assume e.g. $s_{i-1} \geq \ell_{i-1}$. “Count” compares with small pivot first. Probability to generate additional cost 1 in this step:

$$\Pr(\text{“Count” gets extra comparison in step } i) = \frac{\ell_{i-1} + 1}{i + 2}.$$

If \mathcal{S} (based on full history up to $i - 1$, even using randomness) uses small pivot first, no difference.

If \mathcal{S} takes large pivot first:

$$\Pr(\mathcal{S} \text{ gets extra comparison in step } i) = \frac{s_{i-1} + 1}{i + 2},$$

at least as big as the probability for “Count”.

□

Exact analysis of “Count”

Goal

Analyze “Count” **exactly**.

Exact analysis of “Count”

Goal

Analyze “Count” **exactly**.

Exact DP recurrence:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k) \mathbb{E}(C_k)$$

Exact analysis of “Count”

Goal

Analyze “Count” **exactly**.

Exact DP recurrence:

$$\mathbb{E}(C_n) = \mathbb{E}(P_n) + \frac{3}{\binom{n}{2}} \sum_{k=1}^{n-2} (n-1-k) \mathbb{E}(C_k)$$

(Wild 2013) showed how to solve this exactly, using **generating functions**, if $\mathbb{E}(P_n)$ is given.

Reminder: Generating functions.

Analytic function $f: \mathbb{C} \rightarrow \mathbb{C}$ (with a certain convergence radius) can be written $f(z) = \sum_{n \geq 0} a_n z^n$, represents sequence (a_0, a_1, a_2, \dots) .

Reminder: Generating functions.

Analytic function $f: \mathbb{C} \rightarrow \mathbb{C}$ (with a certain convergence radius) can be written $f(z) = \sum_{n \geq 0} a_n z^n$, represents sequence (a_0, a_1, a_2, \dots) .

Example:

$$\begin{aligned} \operatorname{arctanh}(z) &= \frac{1}{2}(\ln(1+z) - \ln(1-z)) \\ &= \frac{1}{2} \sum_{n \geq 1} \frac{(-1)^n + 1}{n} z^n \\ &= \sum_{n \geq 1} \frac{[n \text{ odd}]}{n} z^n. \end{aligned}$$

Reminder: Generating functions.

Analytic function $f: \mathbb{C} \rightarrow \mathbb{C}$ (with a certain convergence radius) can be written $f(z) = \sum_{n \geq 0} a_n z^n$, represents sequence (a_0, a_1, a_2, \dots) .

Example:

$$\begin{aligned} \operatorname{arctanh}(z) &= \frac{1}{2}(\ln(1+z) - \ln(1-z)) \\ &= \frac{1}{2} \sum_{n \geq 1} \frac{(-1)^n + 1}{n} z^n \\ &= \sum_{n \geq 1} \frac{[n \text{ odd}]}{n} z^n. \end{aligned}$$

Represents $(0, 1, 0, \frac{1}{3}, 0, \frac{1}{5}, 0, \frac{1}{7}, \dots)$

Reminder: Generating functions.

Analytic function $f: \mathbb{C} \rightarrow \mathbb{C}$ (with a certain convergence radius) can be written $f(z) = \sum_{n \geq 0} a_n z^n$, represents sequence (a_0, a_1, a_2, \dots) .

Example:

$$\begin{aligned} \operatorname{arctanh}(z) &= \frac{1}{2}(\ln(1+z) - \ln(1-z)) \\ &= \frac{1}{2} \sum_{n \geq 1} \frac{(-1)^n + 1}{n} z^n \\ &= \sum_{n \geq 1} \frac{[n \text{ odd}]}{n} z^n. \end{aligned}$$

Represents $(0, 1, 0, \frac{1}{3}, 0, \frac{1}{5}, 0, \frac{1}{7}, \dots)$

Exact analysis of “Count”

For comparison count of sorting: $C(z) = \sum_{n \geq 0} \mathbb{E}(C_n) z^n,$

for comparison count of partitioning: $P(z) = \sum_{n \geq 0} \mathbb{E}(P_n) z^n.$

(Wild 2013) showed: If $\mathbb{E}(P_n)$ and $\mathbb{E}(C_n)$ have generating functions $P(z)$ and $C(z)$, then

$$C(z) = (1 - z)^3 \int_0^z (1 - t)^{-6} \int_0^t (1 - s)^3 P''(s) ds dt.$$

Exact analysis of “Count”

For comparison count of sorting: $C(z) = \sum_{n \geq 0} \mathbb{E}(C_n)z^n,$

for comparison count of partitioning: $P(z) = \sum_{n \geq 0} \mathbb{E}(P_n)z^n.$

(Wild 2013) showed: If $\mathbb{E}(P_n)$ and $\mathbb{E}(C_n)$ have generating functions $P(z)$ and $C(z)$, then

$$C(z) = (1 - z)^3 \int_0^z (1 - t)^{-6} \int_0^t (1 - s)^3 P''(s) ds dt.$$

Task: Find exact formula for $\mathbb{E}(P_n)$ (and its generating function) for strategy “Count”.

Exact analysis of “Count”

For comparison count of sorting: $C(z) = \sum_{n \geq 0} \mathbb{E}(C_n)z^n,$

for comparison count of partitioning: $P(z) = \sum_{n \geq 0} \mathbb{E}(P_n)z^n.$

(Wild 2013) showed: If $\mathbb{E}(P_n)$ and $\mathbb{E}(C_n)$ have generating functions $P(z)$ and $C(z)$, then

$$C(z) = (1 - z)^3 \int_0^z (1 - t)^{-6} \int_0^t (1 - s)^3 P''(s) ds dt.$$

Task: Find exact formula for $\mathbb{E}(P_n)$ (and its generating function) for strategy “Count”.

Exact analysis of “Count”

Define “random walk”:

$$X_i = s_i - \ell_i, \quad \text{for } 0 \leq i \leq n.$$

Exact analysis of “Count”

Define “random walk”:

$$X_i = s_i - \ell_i, \quad \text{for } 0 \leq i \leq n.$$

Classification Strategy, round i :

- $X_{i-1} \geq 0$: compare with smaller pivot first.
- $X_{i-1} < 0$: compare with larger pivot first.

Exact analysis of “Count”

Define “random walk”:

$$X_i = s_i - \ell_i, \quad \text{for } 0 \leq i \leq n.$$

Classification Strategy, round i :

- $X_{i-1} \geq 0$: compare with smaller pivot first.
- $X_{i-1} < 0$: compare with larger pivot first.

First study simplified situation, ignore medium elements.

Exact analysis of “Count”

Define “random walk”:

$$X_i = s_i - \ell_i, \quad \text{for } 0 \leq i \leq n.$$

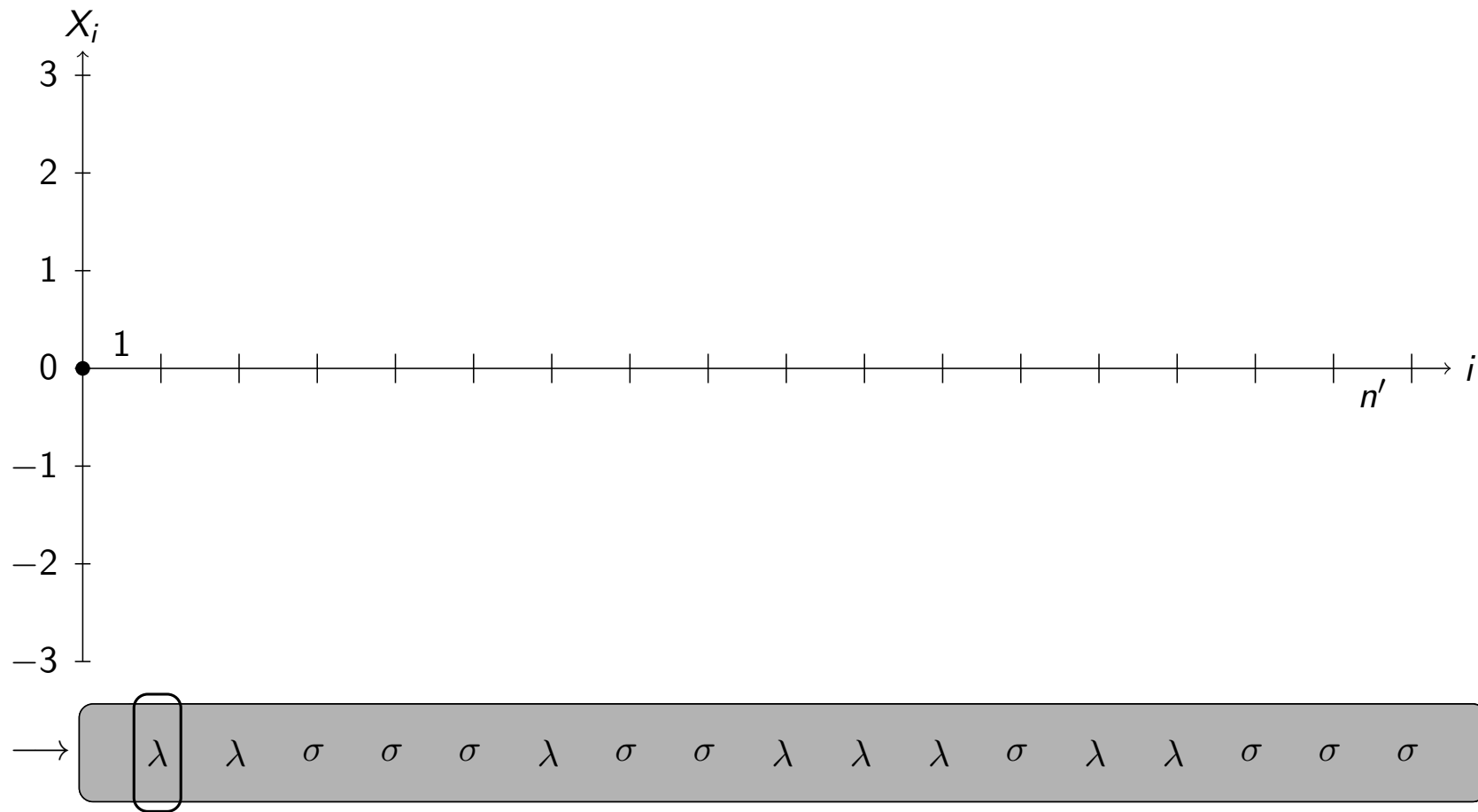
Classification Strategy, round i :

- $X_{i-1} \geq 0$: compare with smaller pivot first.
- $X_{i-1} < 0$: compare with larger pivot first.

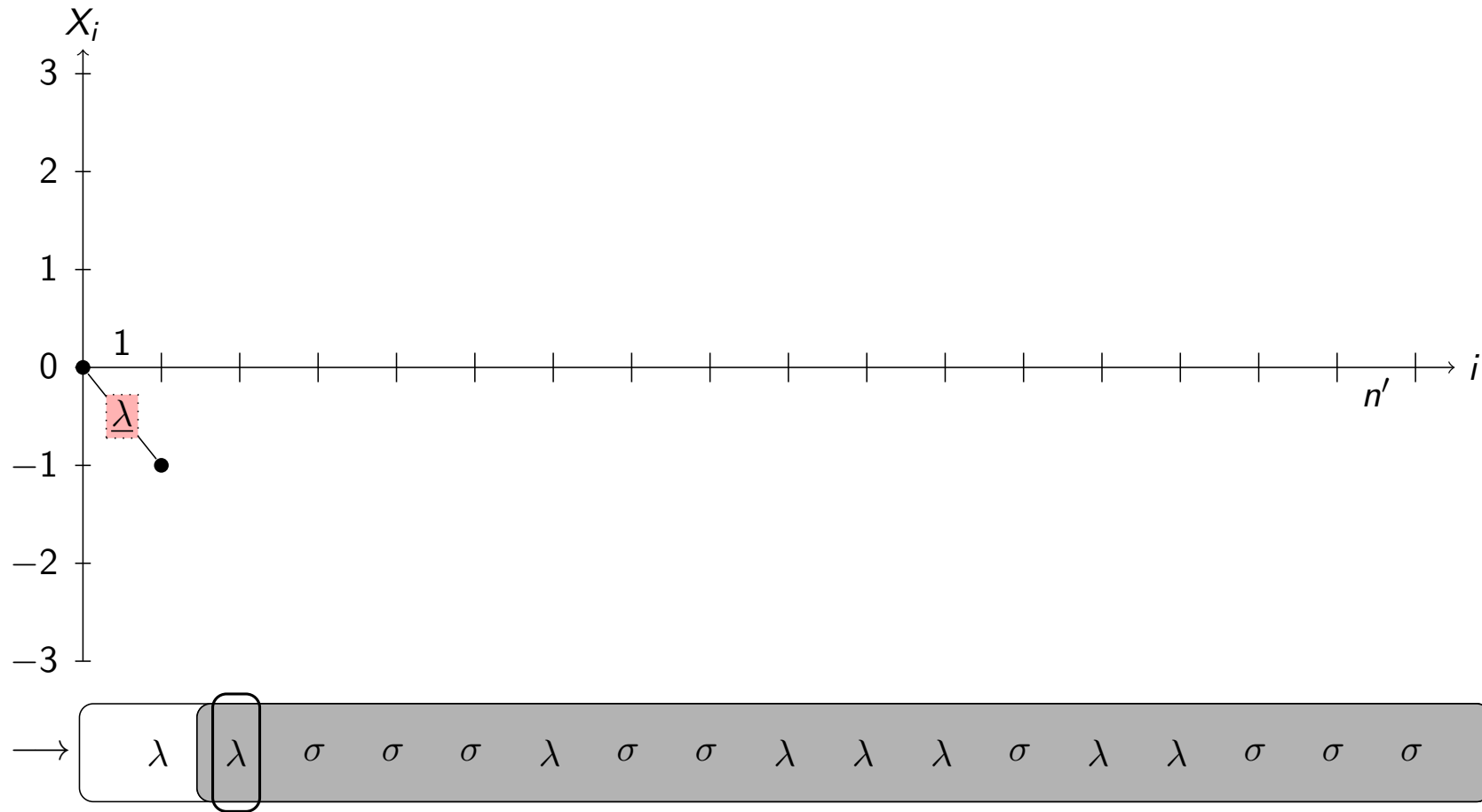
First study simplified situation, ignore medium elements.

We classify $n' = s + \ell$ elements.

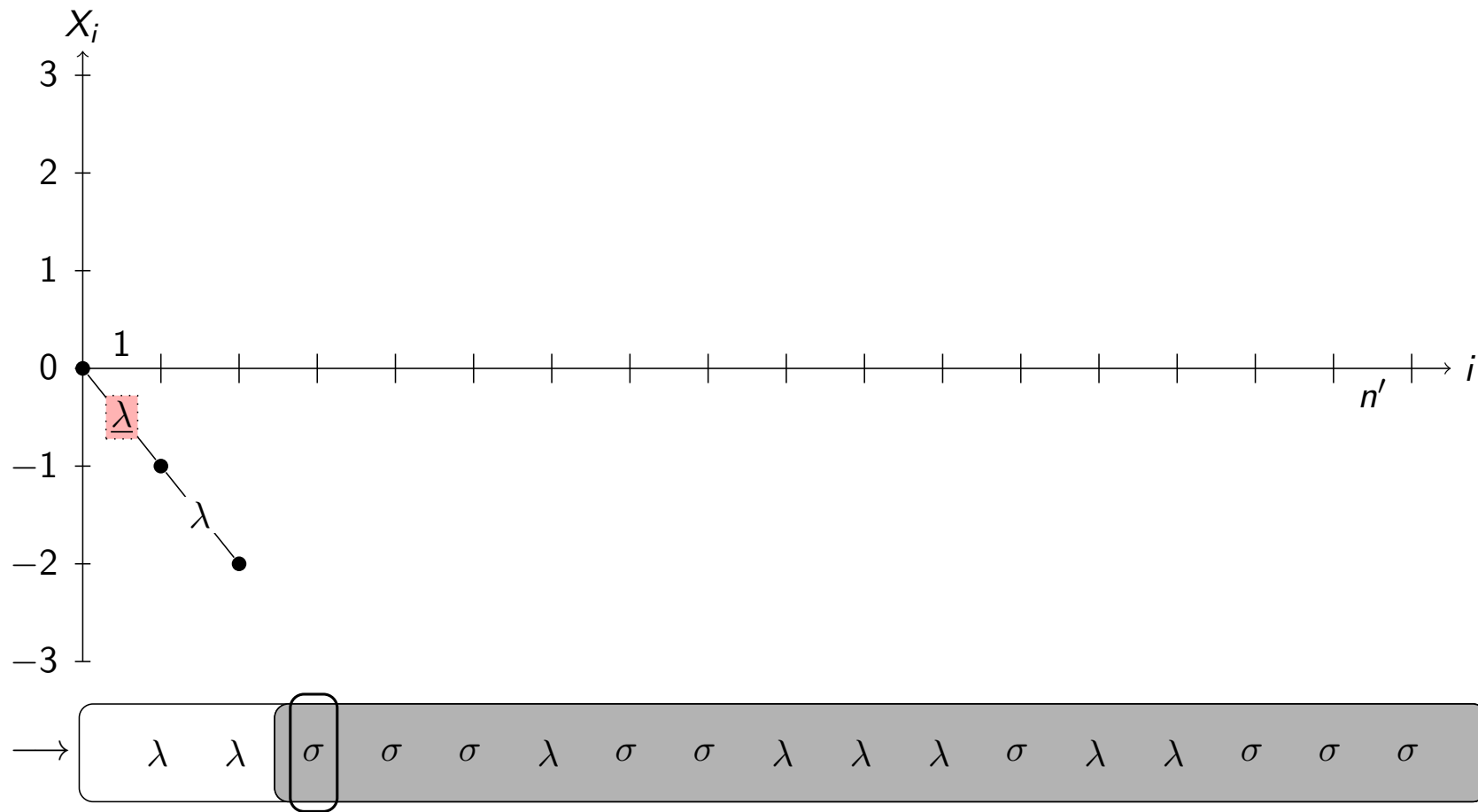
Random Walks & Analysis of Count



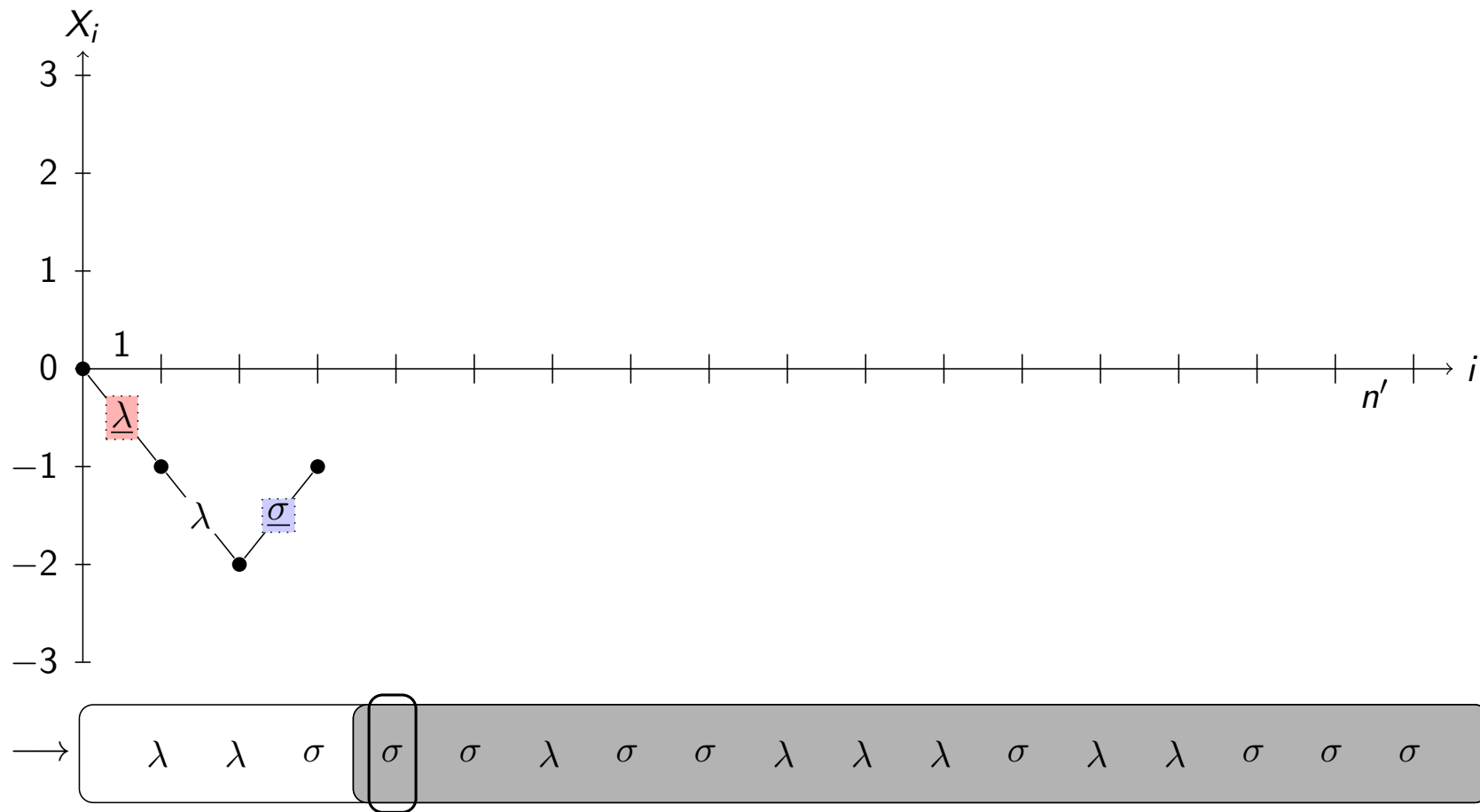
Random Walks & Analysis of Count



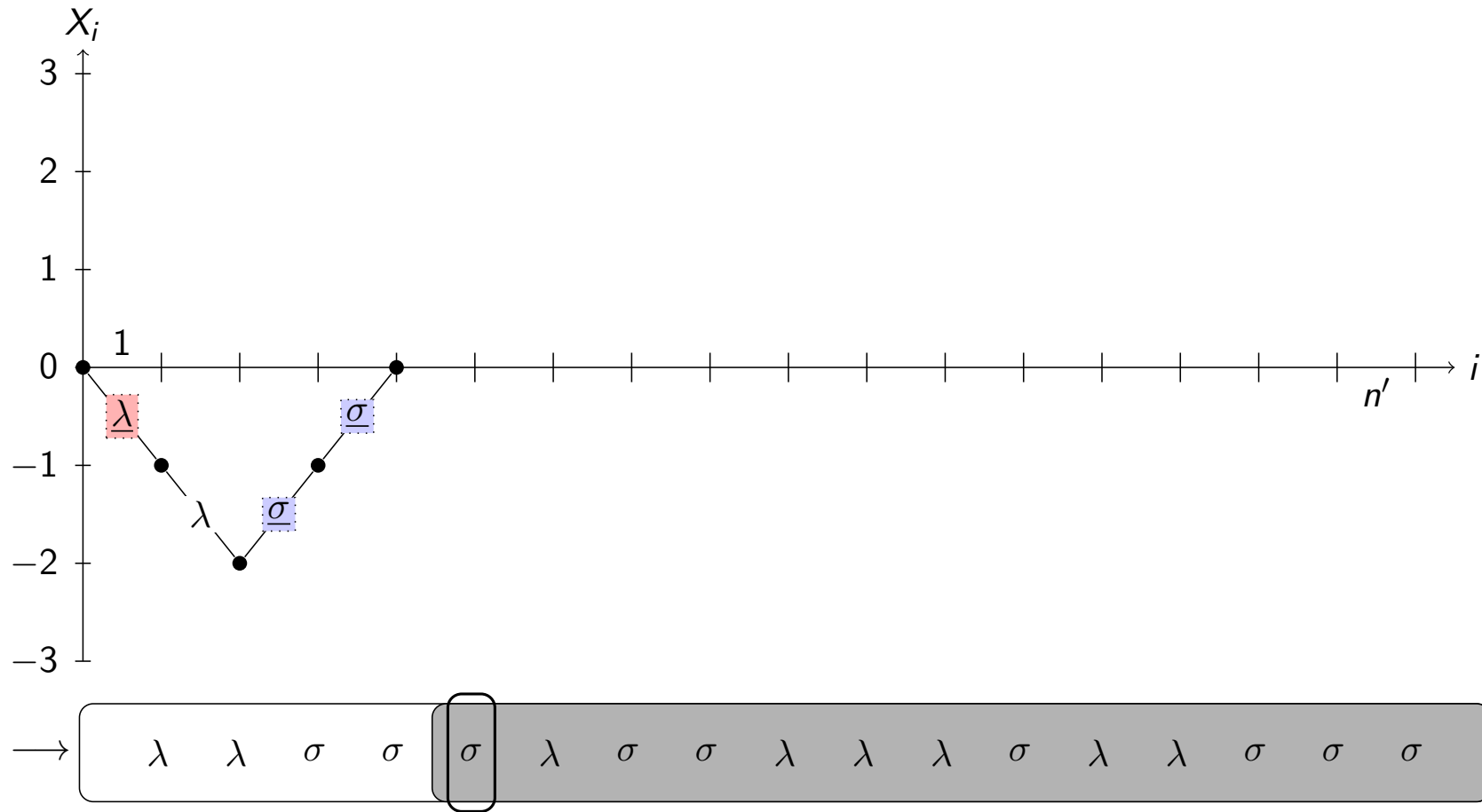
Random Walks & Analysis of Count



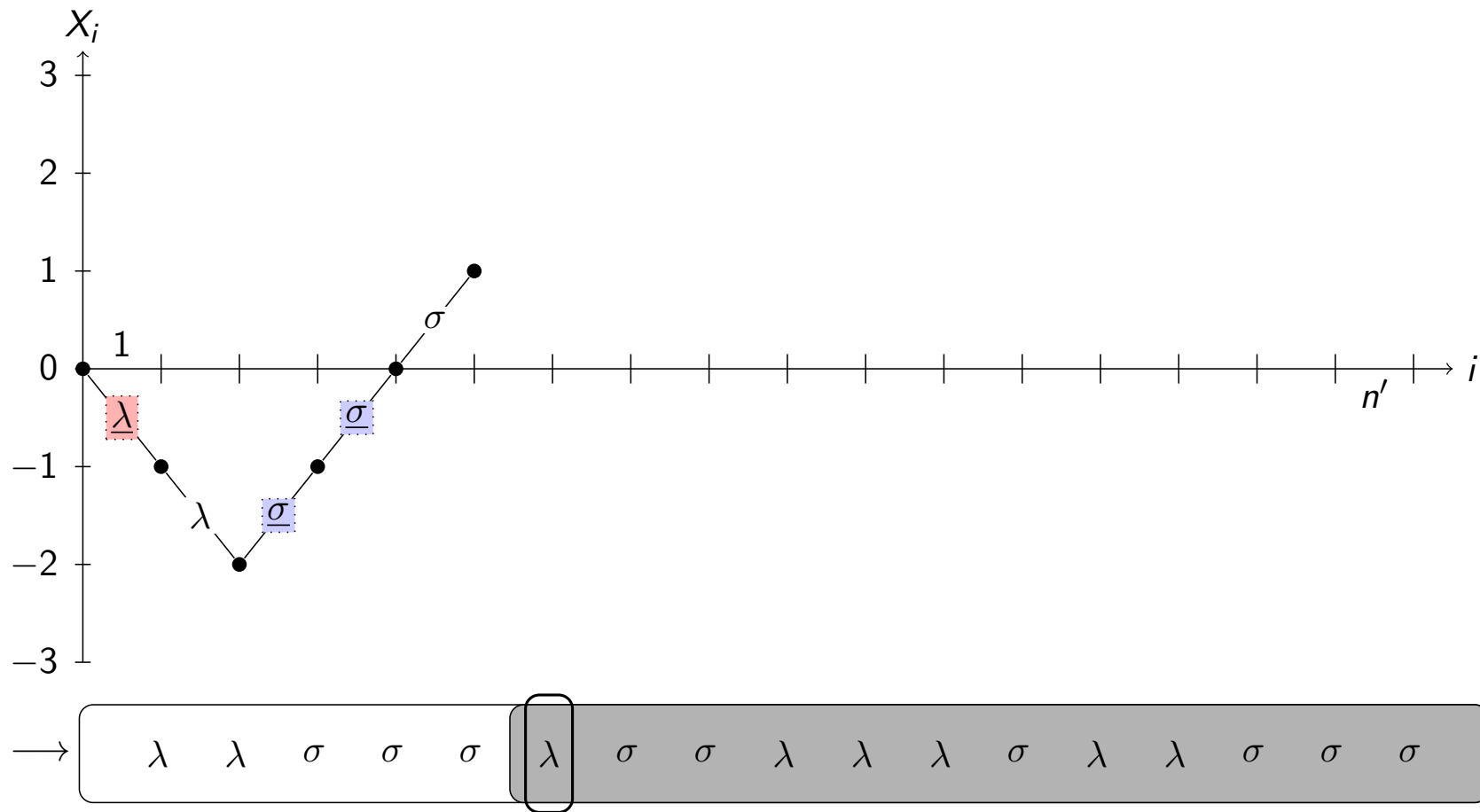
Random Walks & Analysis of Count



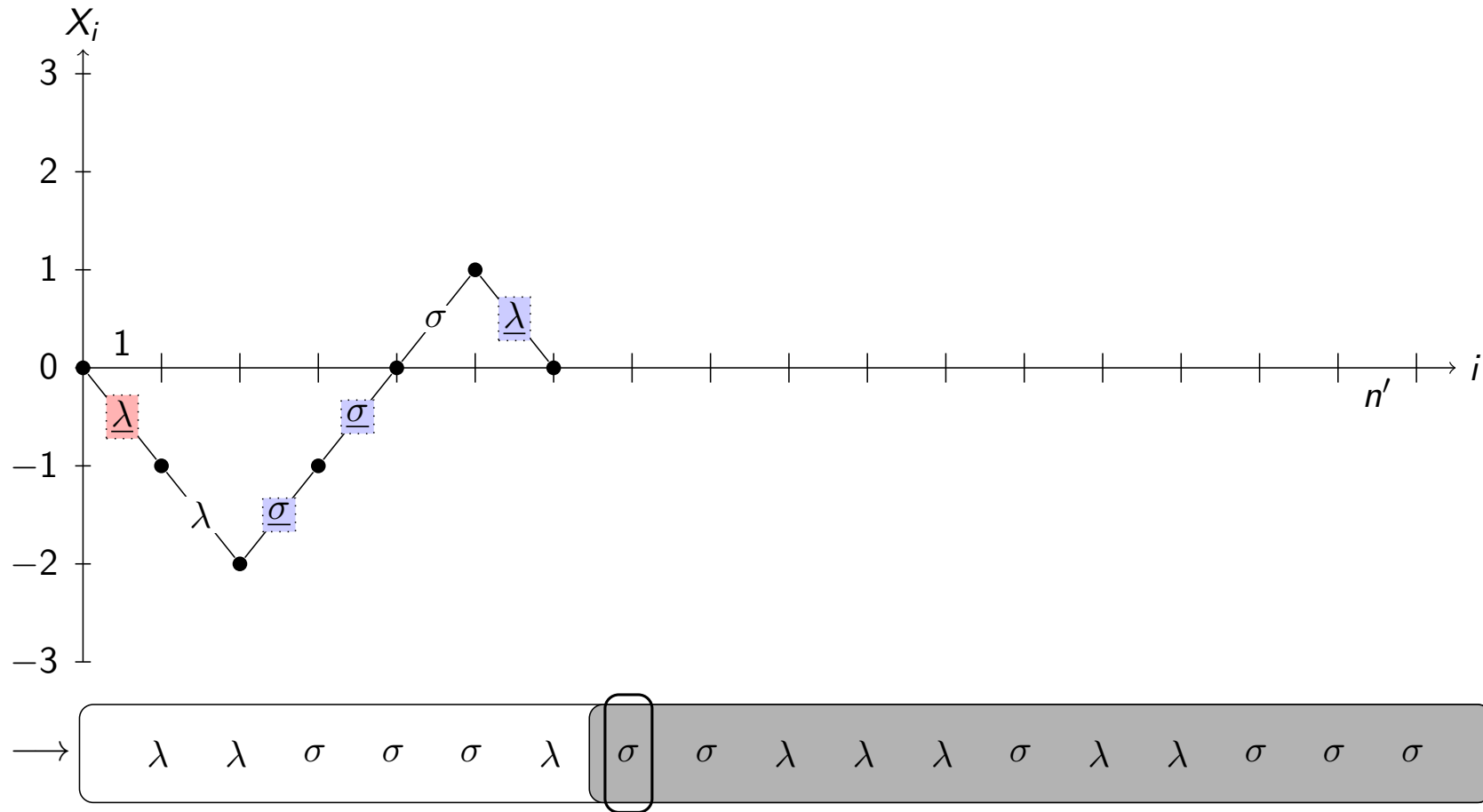
Random Walks & Analysis of Count



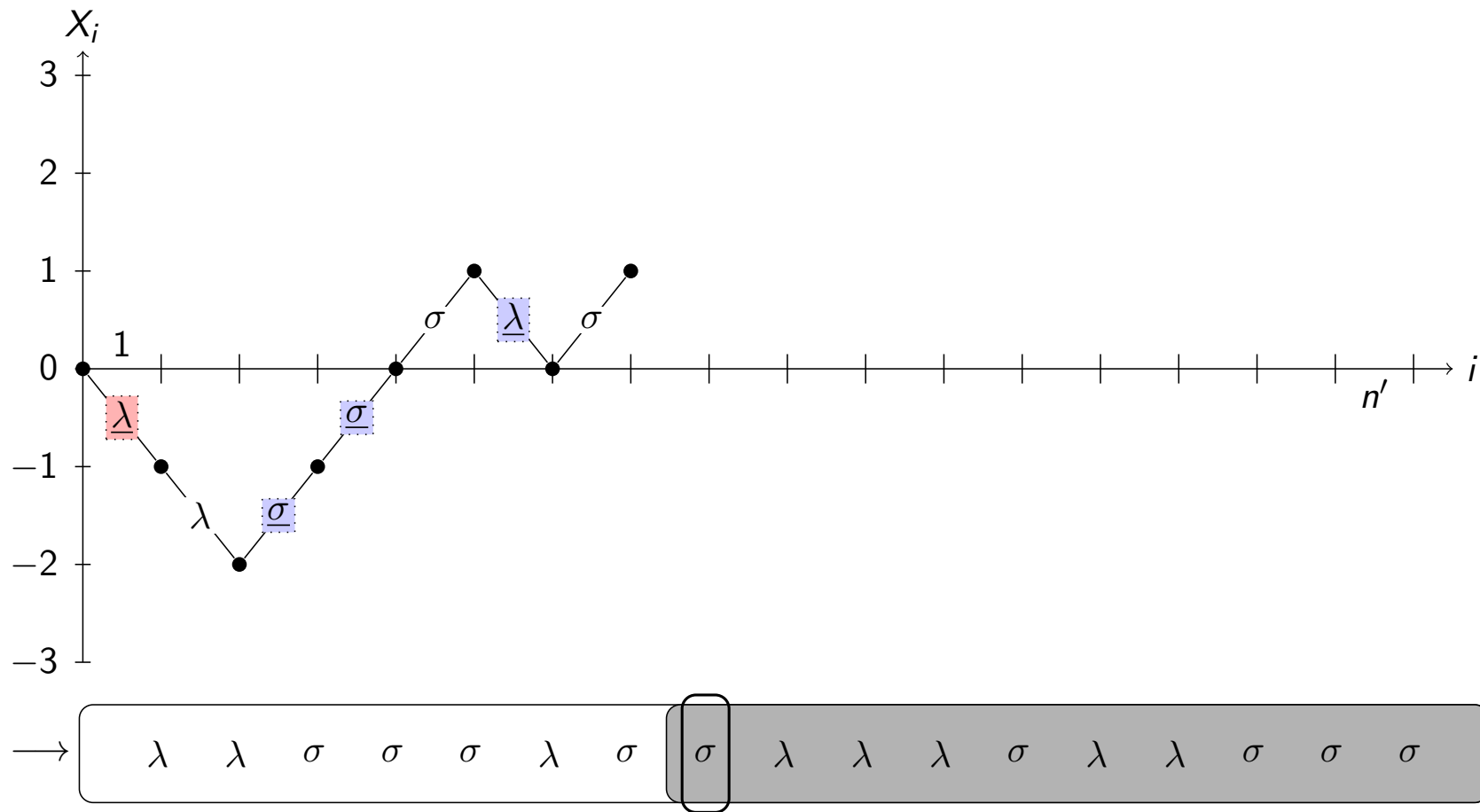
Random Walks & Analysis of Count



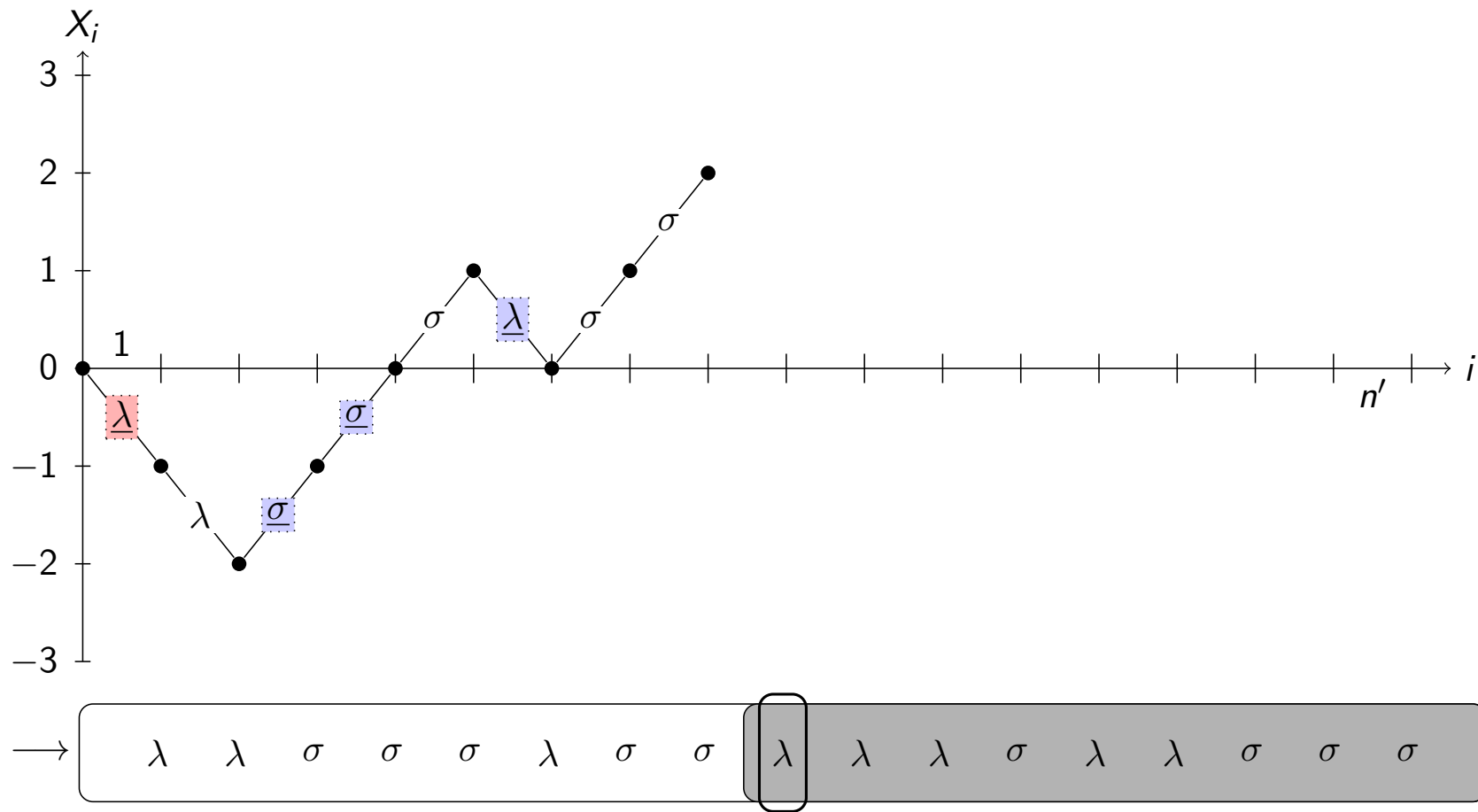
Random Walks & Analysis of Count



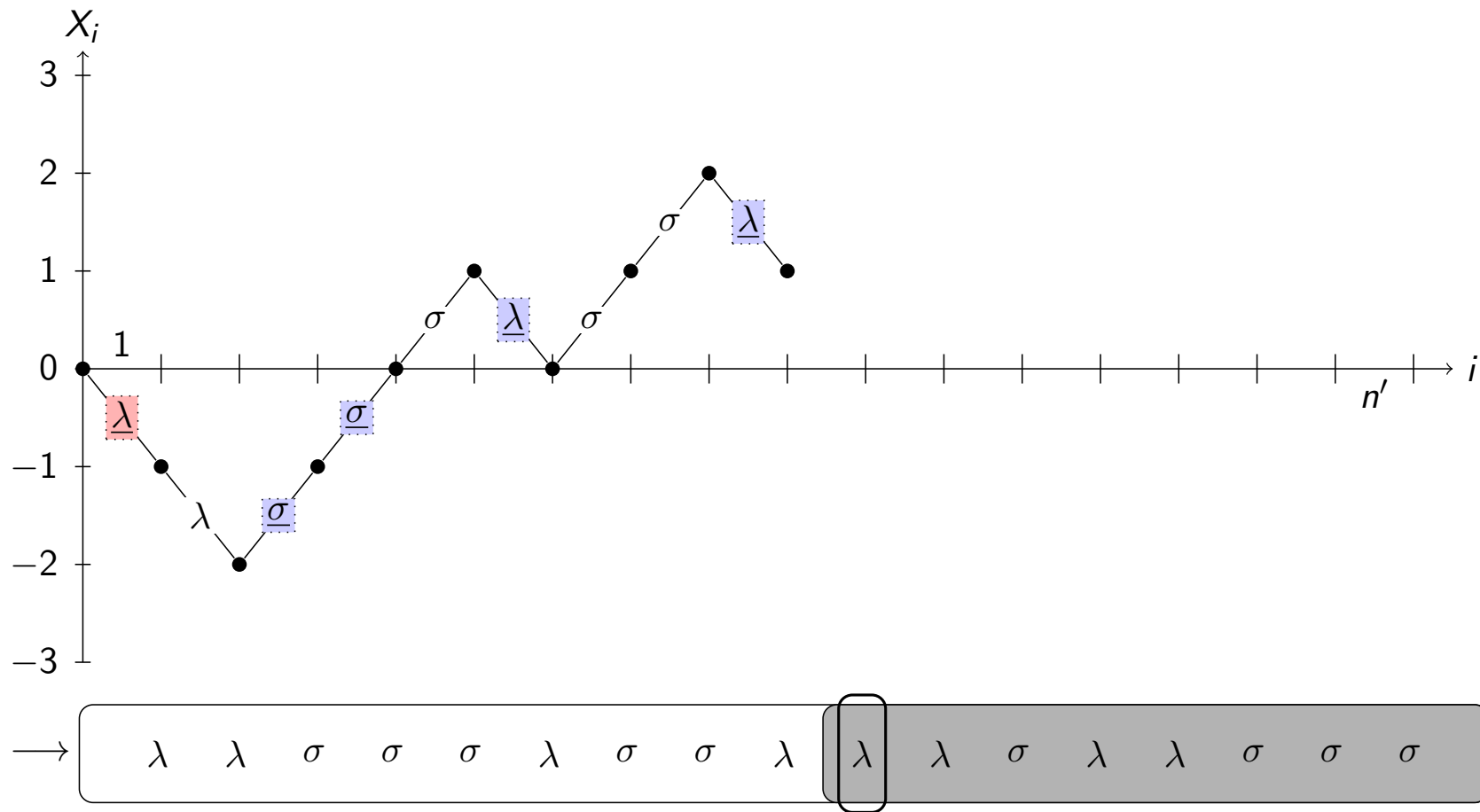
Random Walks & Analysis of Count



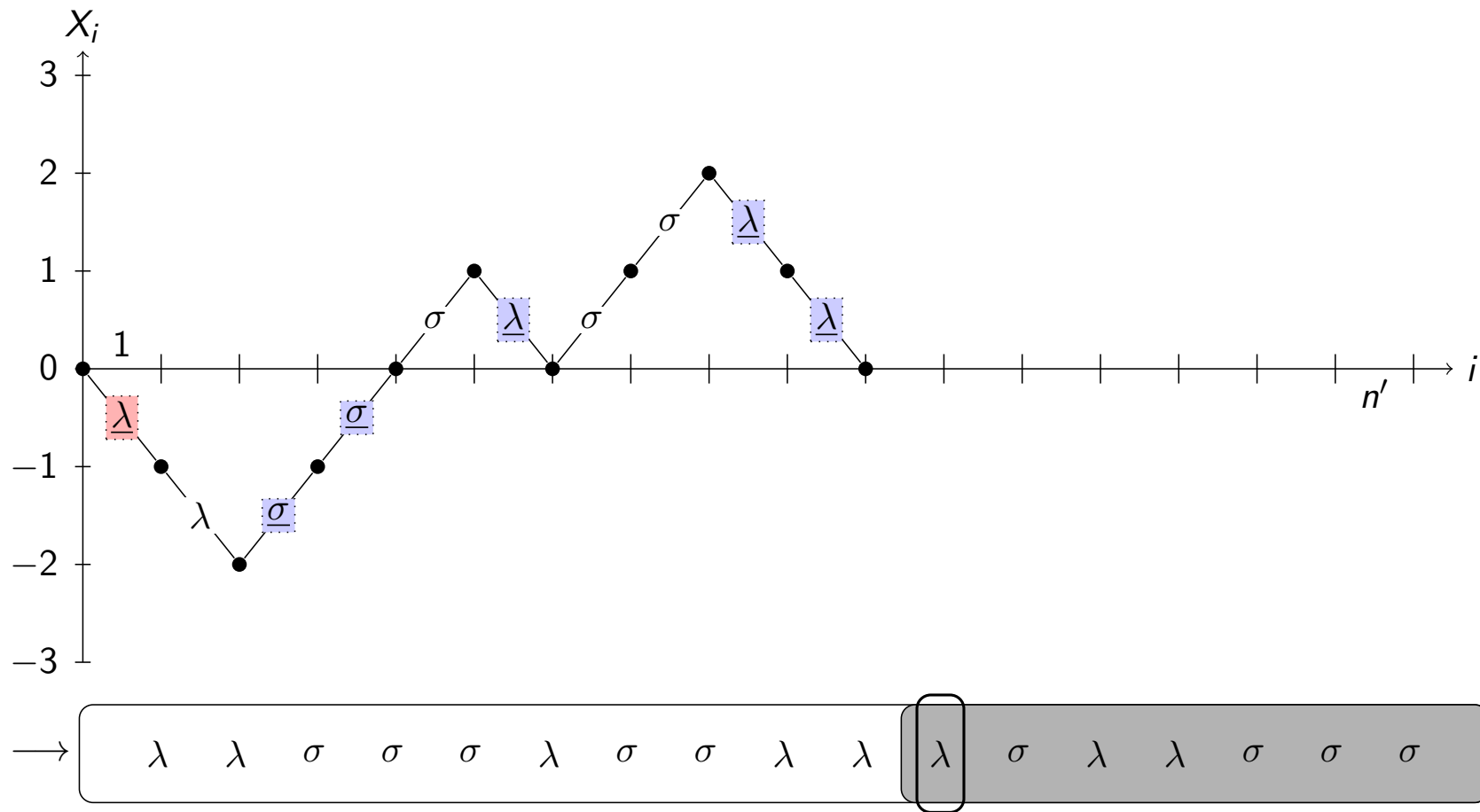
Random Walks & Analysis of Count



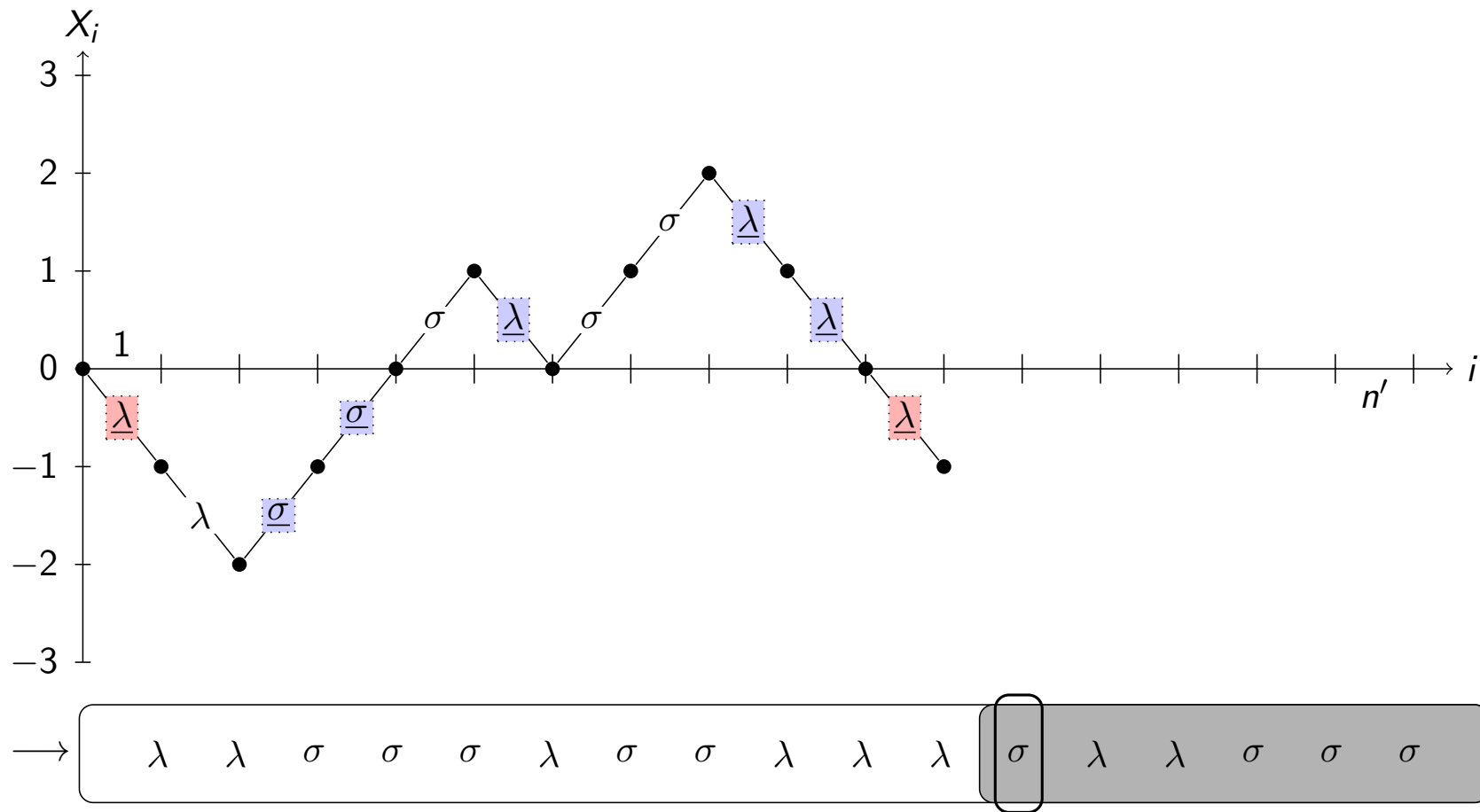
Random Walks & Analysis of Count



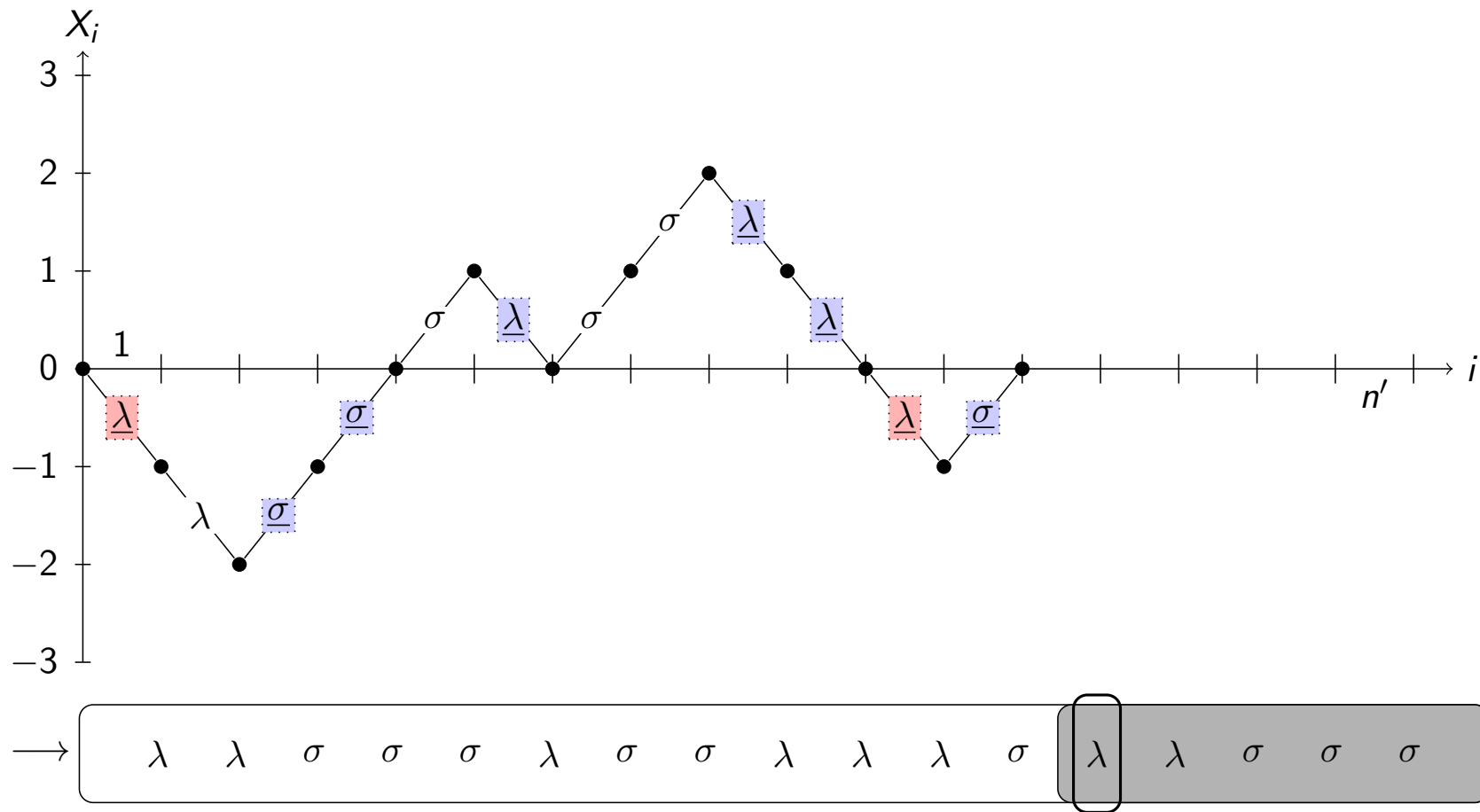
Random Walks & Analysis of Count



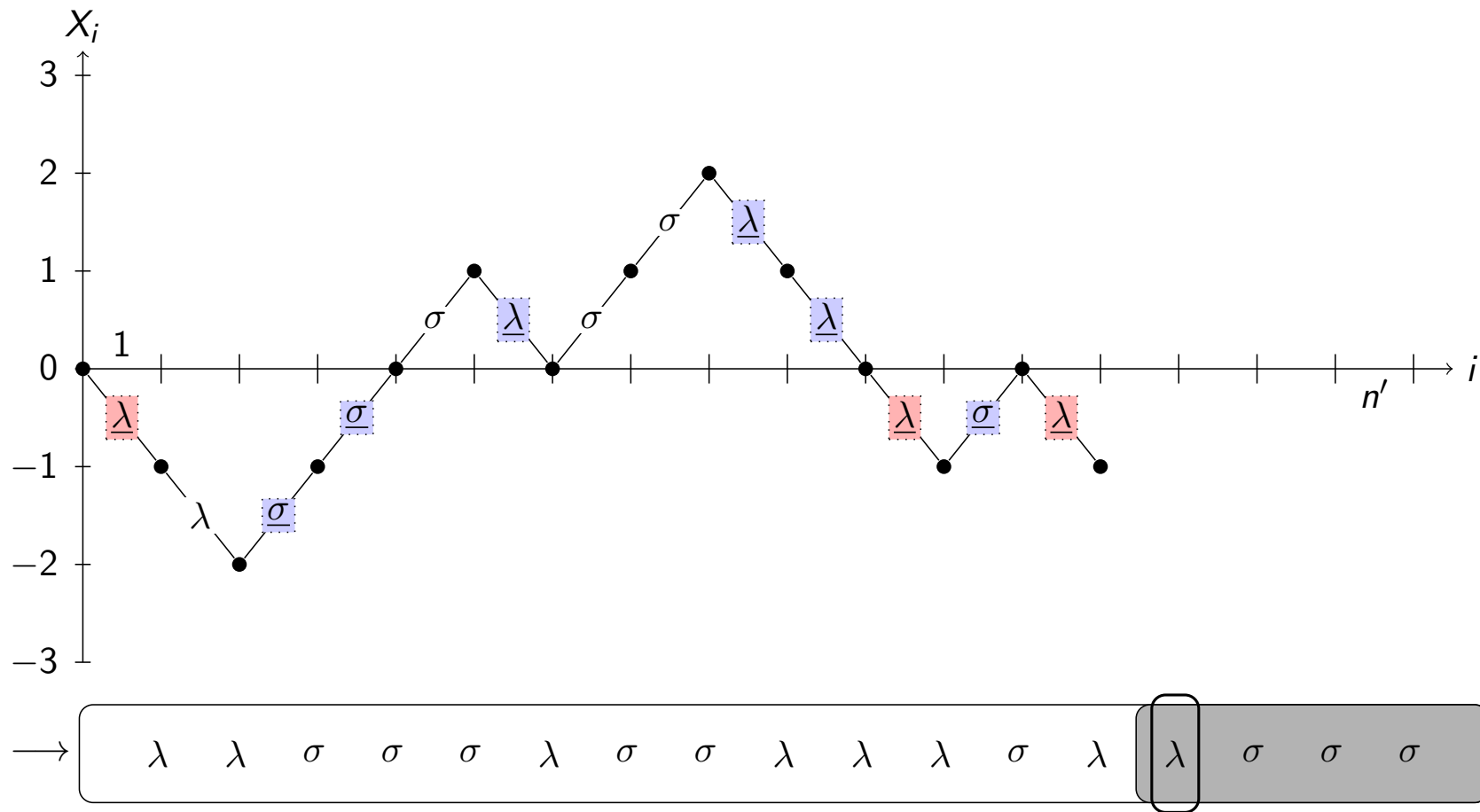
Random Walks & Analysis of Count



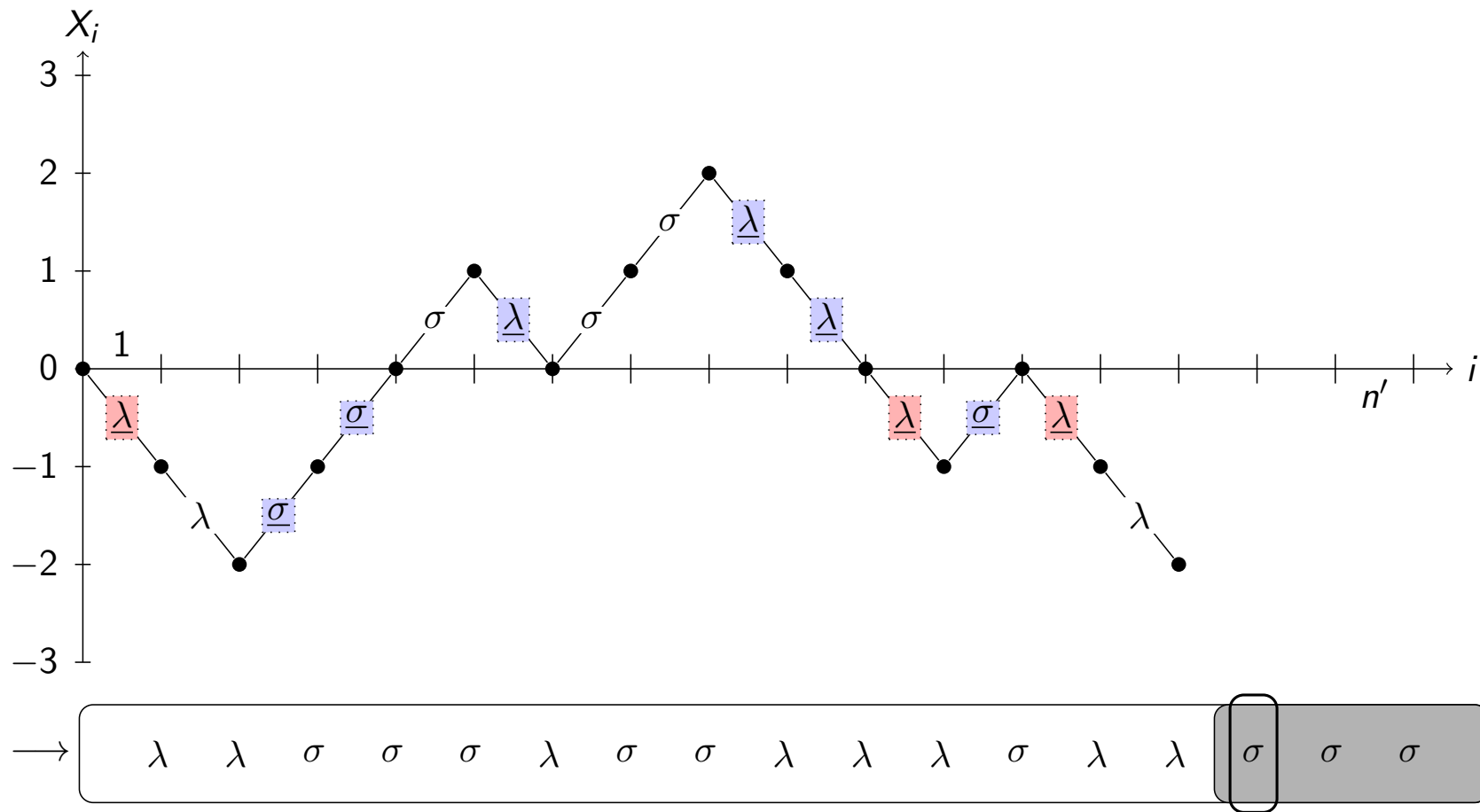
Random Walks & Analysis of Count



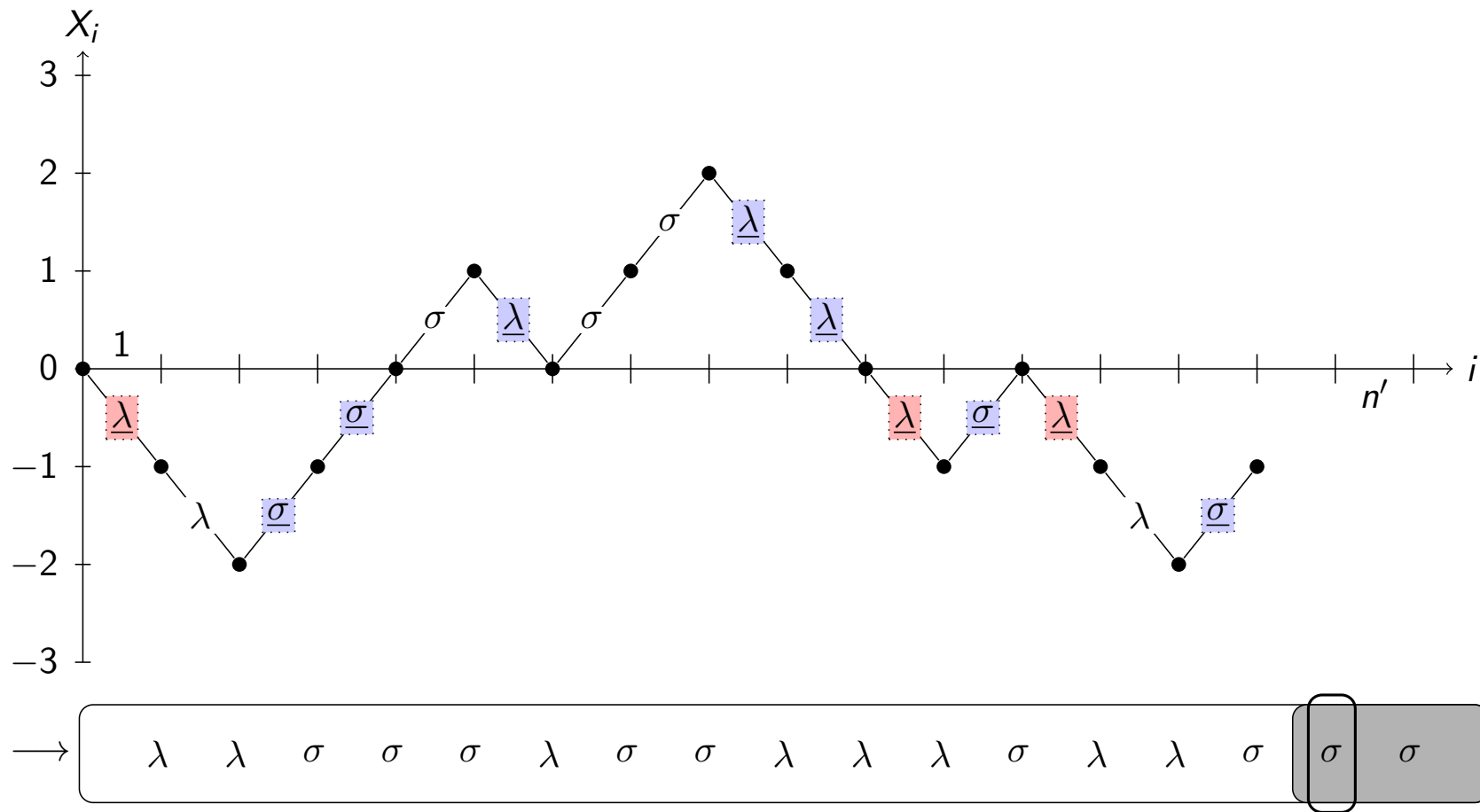
Random Walks & Analysis of Count



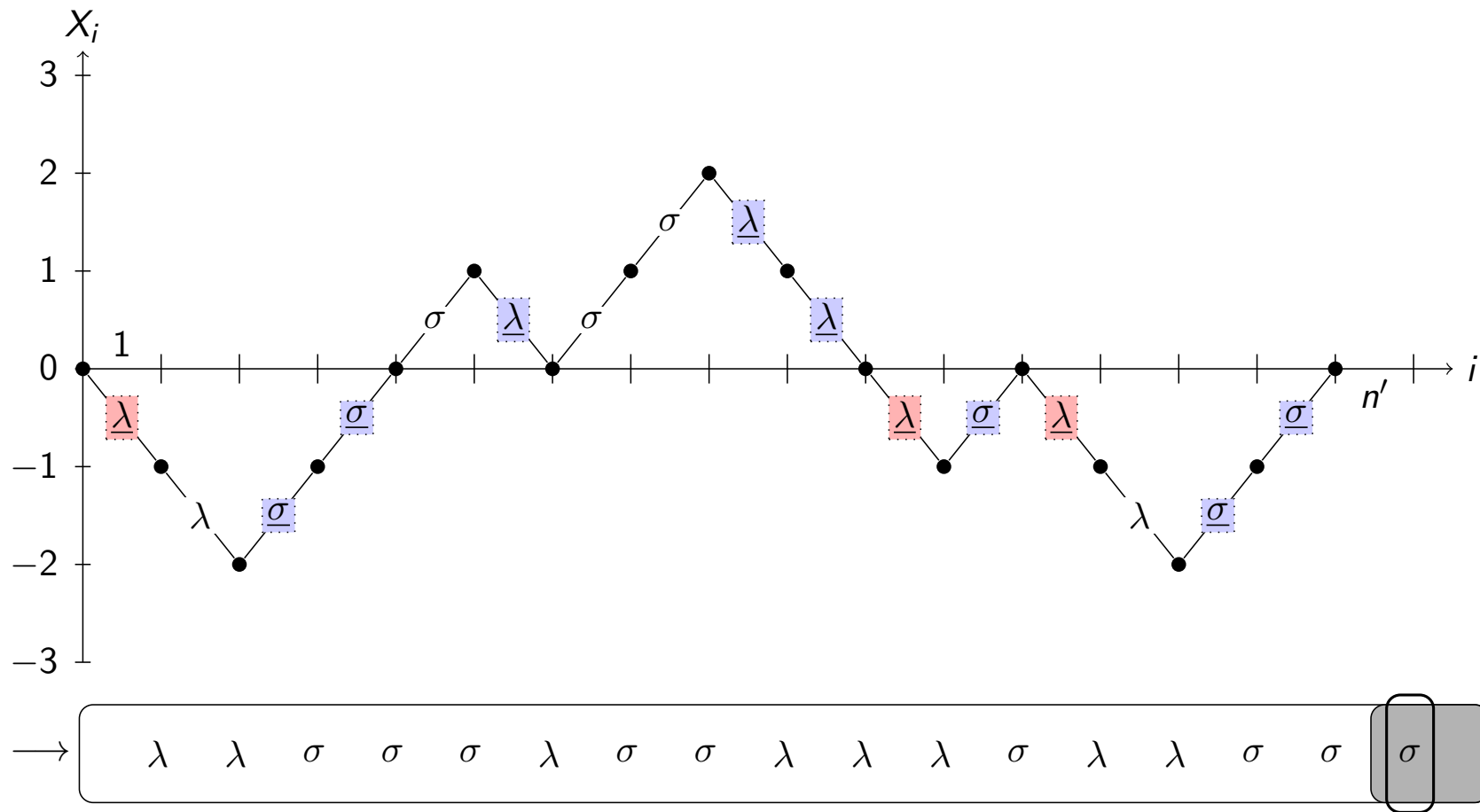
Random Walks & Analysis of Count



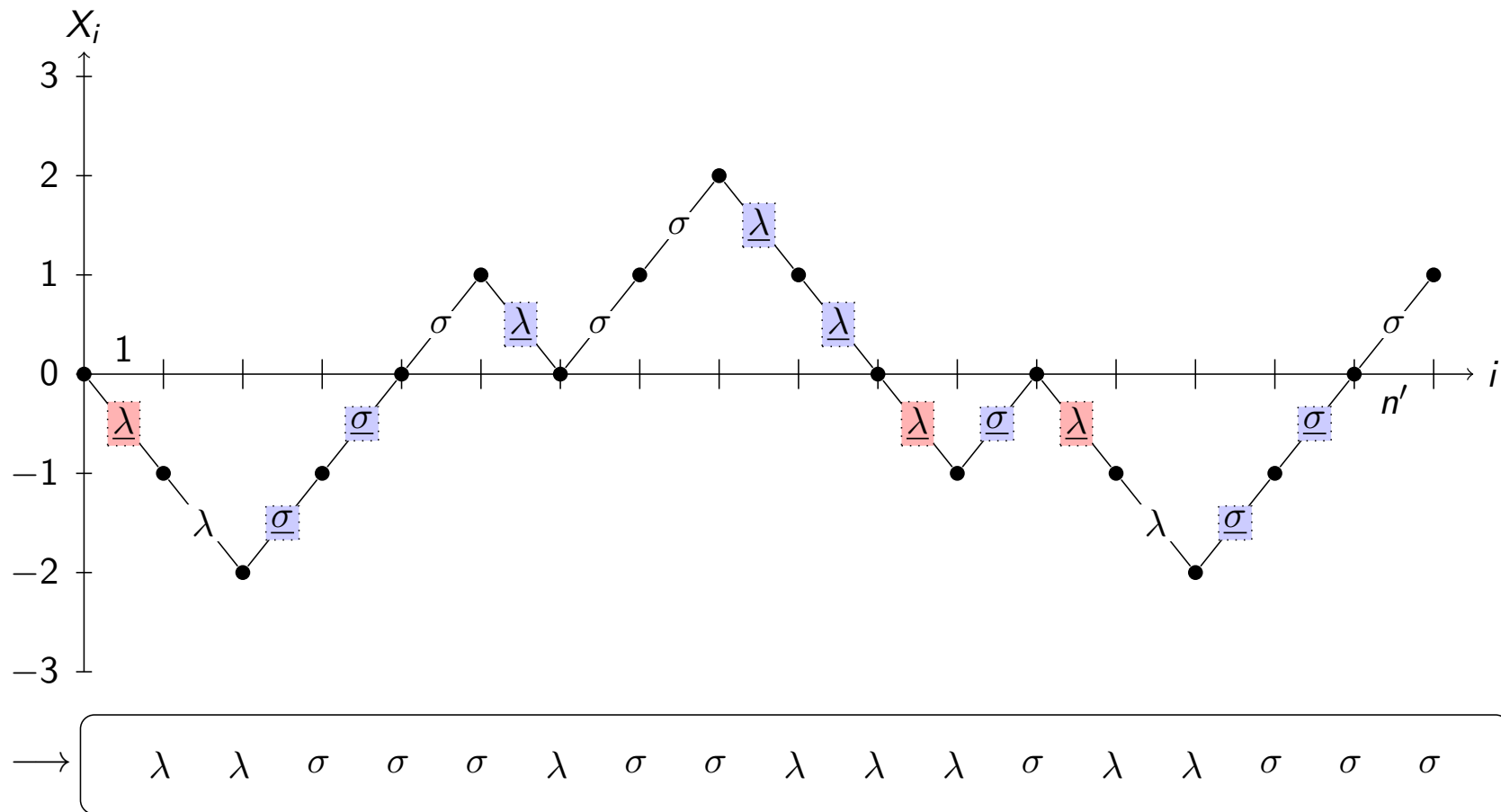
Random Walks & Analysis of Count



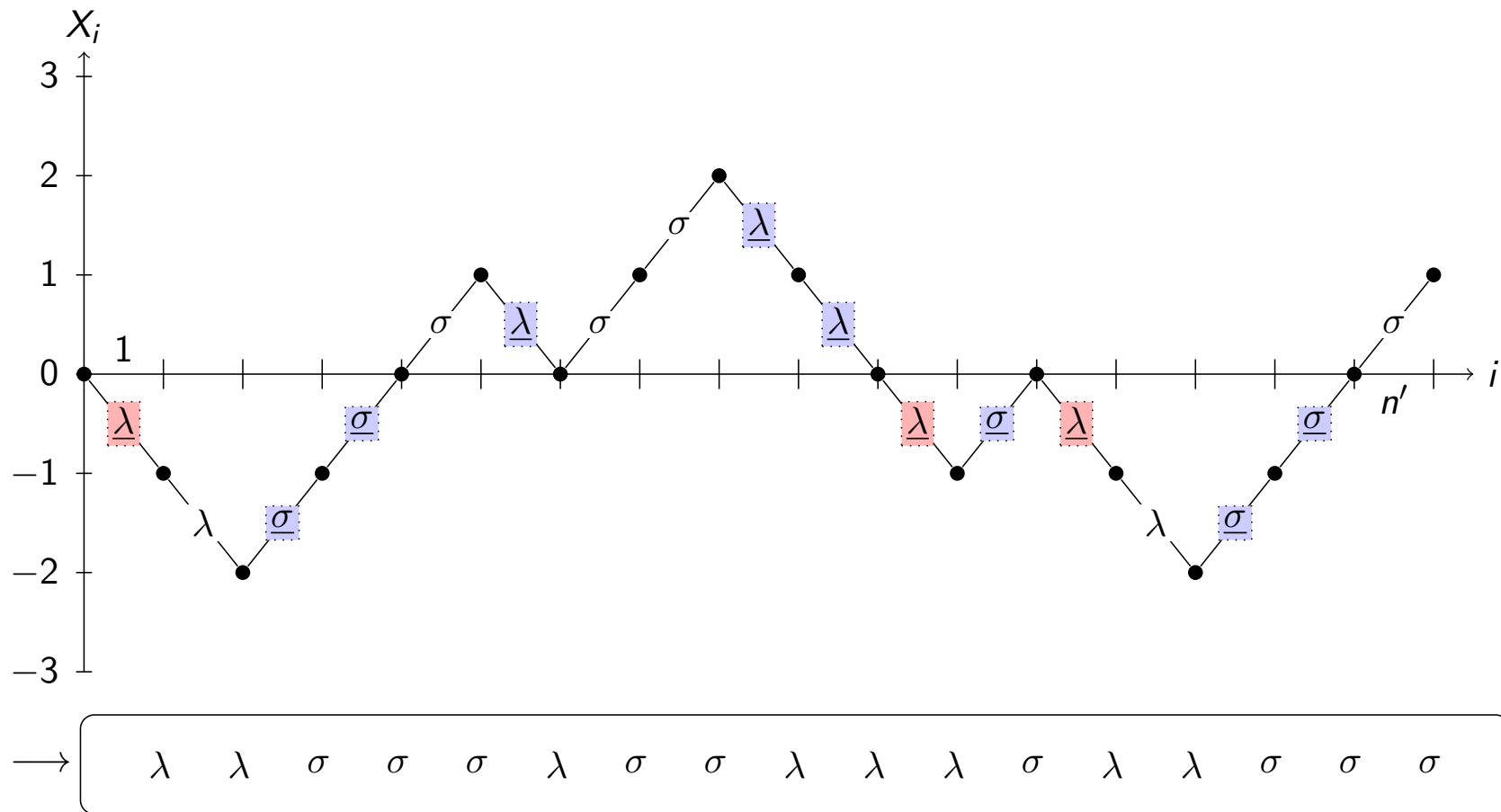
Random Walks & Analysis of Count



Random Walks & Analysis of Count

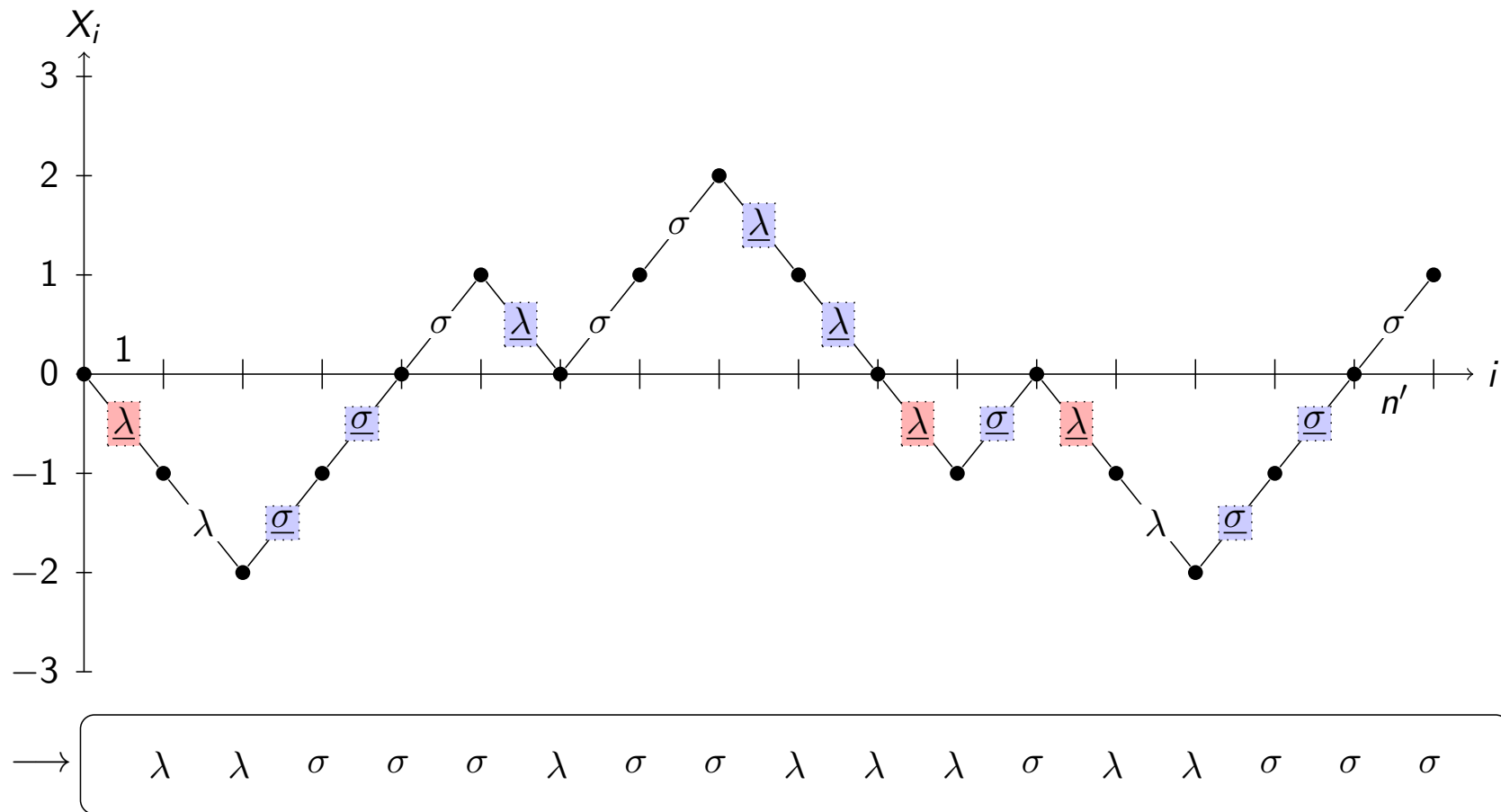


Random Walks & Analysis of Count



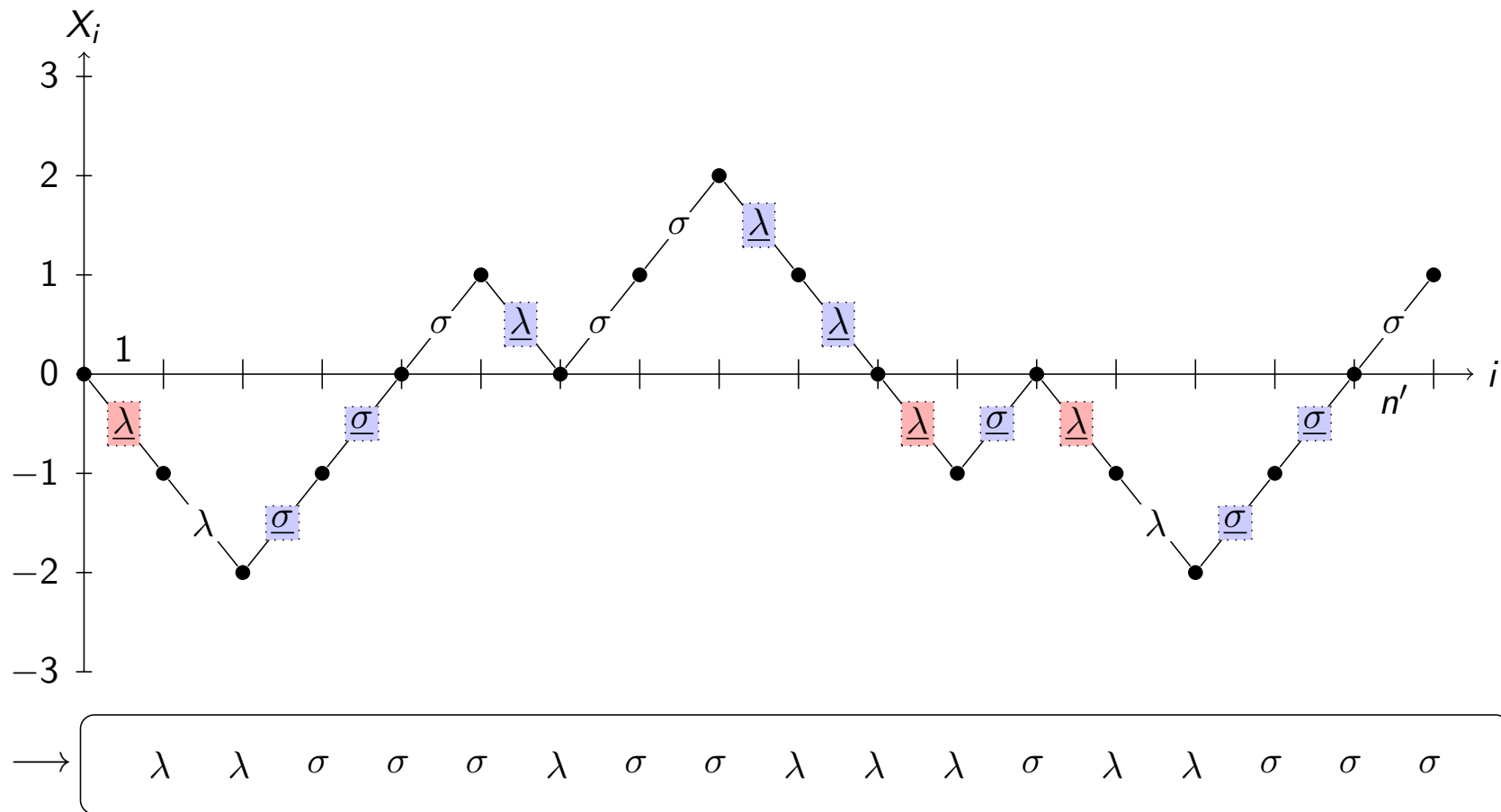
Observation: Extra comparison in round $i \Leftrightarrow$

Random Walks & Analysis of Count



Observation: Extra comparison in round $i \Leftrightarrow$
 move towards zero in $X_{i-1} \rightarrow X_i$

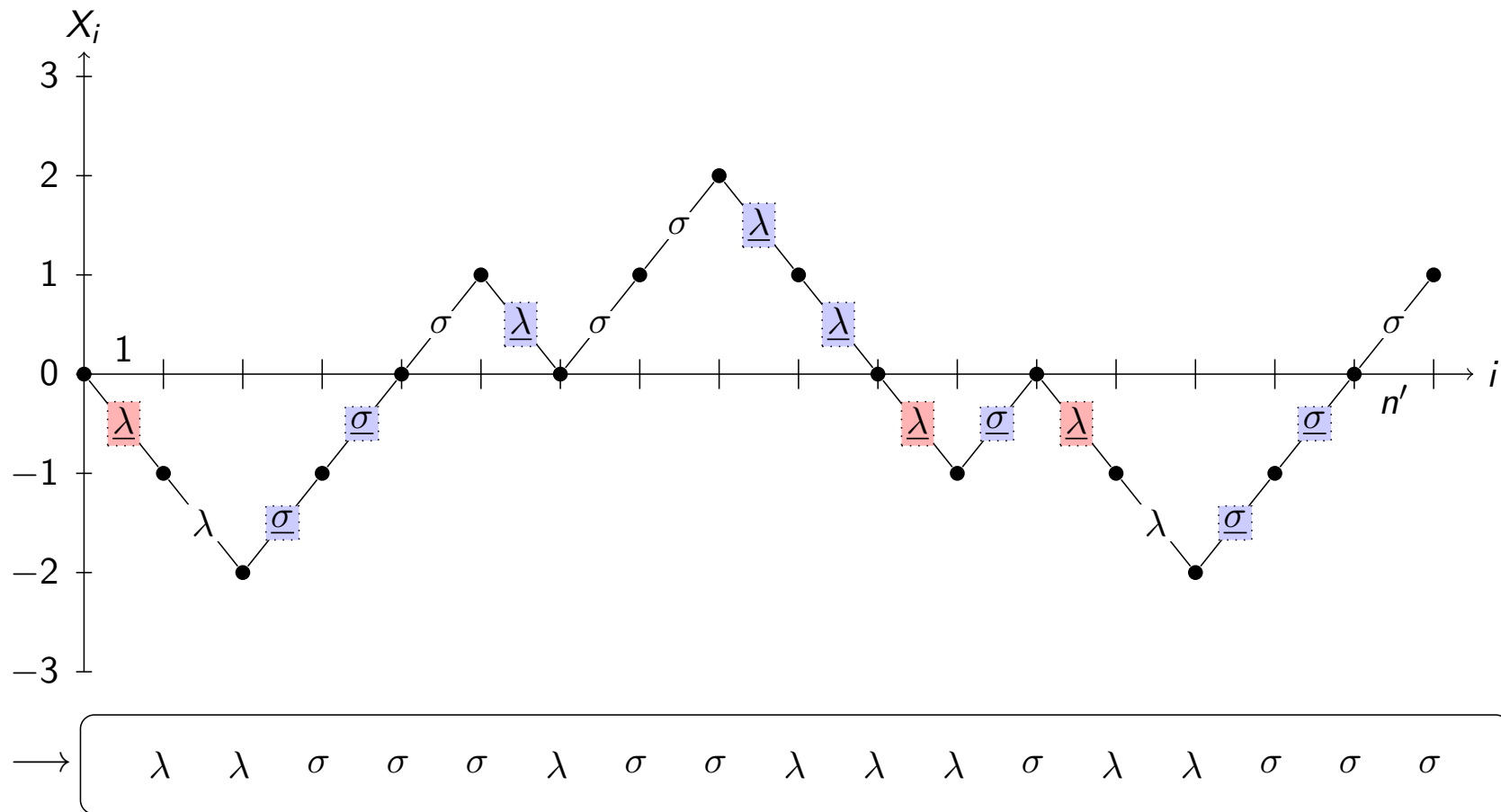
Random Walks & Analysis of Count



Observation: Extra comparison in round $i \Leftrightarrow$

move towards zero in $X_{i-1} \rightarrow X_i$ or move down from a zero.

Random Walks & Analysis of Count



Observation: Extra comparison in round $i \Leftrightarrow$

move towards zero in $X_{i-1} \rightarrow X_i$ or move down from a zero.

Easy: Exactly $\min\{s, \ell\}$ many “move towards zero” situations.

Counting Zeros in Random Walks

Number of zeros (without time n'): $Z_{n'} := \#\{i \mid 0 \leq i \leq n', X_i = 0\}$.

Counting Zeros in Random Walks

Number of zeros (without time n'): $Z_{n'} := \#\{i \mid 0 \leq i \leq n', X_i = 0\}$.

Can only have zeros at **even** positions i .

What is $\mathbb{E}(Z_{n'})$?

Counting Zeros in Random Walks

Number of zeros (without time n'): $Z_{n'} := \#\{i \mid 0 \leq i \leq n', X_i = 0\}$.

Can only have zeros at **even** positions i .

What is $\mathbb{E}(Z_{n'})$?

$$\begin{aligned}\mathbb{E}(Z_{n'}) &= \frac{1}{n' + 1} \sum_{m=0}^{\lfloor n'/2 \rfloor} \sum_{\ell=m}^{n'-m} \frac{\binom{2m}{m} \binom{n'-2m}{\ell-m}}{\binom{n'}{\ell}} \\ &= \frac{4}{n' + 1} \sum_{0 \leq k < \ell < \lceil n'/2 \rceil} \frac{\binom{n'}{k}}{\binom{n'}{\ell}} + [n' \text{ even}] \frac{1}{n' + 1} \left(\frac{2^{n'}}{\binom{n'}{n'/2}} - 1 \right) + 1.\end{aligned}$$

Counting Zeros in Random Walks

Number of zeros (without time n'): $Z_{n'} := \#\{i \mid 0 \leq i \leq n', X_i = 0\}$.

Can only have zeros at **even** positions i .

What is $\mathbb{E}(Z_{n'})$?

$$\begin{aligned}\mathbb{E}(Z_{n'}) &= \frac{1}{n' + 1} \sum_{m=0}^{\lfloor n'/2 \rfloor} \sum_{\ell=m}^{n'-m} \frac{\binom{2m}{m} \binom{n'-2m}{\ell-m}}{\binom{n'}{\ell}} \\ &= \frac{4}{n' + 1} \sum_{0 \leq k < \ell < \lceil n'/2 \rceil} \frac{\binom{n'}{k}}{\binom{n'}{\ell}} + [n' \text{ even}] \frac{1}{n' + 1} \left(\frac{2^{n'}}{\binom{n'}{n'/2}} - 1 \right) + 1.\end{aligned}$$

(By generating function manipulations.)

Counting Zeros in Random Walks

Number of zeros (without time n'): $Z_{n'} := \#\{i \mid 0 \leq i \leq n', X_i = 0\}$.

Can only have zeros at **even** positions i .

What is $\mathbb{E}(Z_{n'})$?

$$\begin{aligned}\mathbb{E}(Z_{n'}) &= \frac{1}{n' + 1} \sum_{m=0}^{\lfloor n'/2 \rfloor} \sum_{\ell=m}^{n'-m} \frac{\binom{2m}{m} \binom{n'-2m}{\ell-m}}{\binom{n'}{\ell}} \\ &= \frac{4}{n' + 1} \sum_{0 \leq k < \ell < \lceil n'/2 \rceil} \frac{\binom{n'}{k}}{\binom{n'}{\ell}} + [n' \text{ even}] \frac{1}{n' + 1} \left(\frac{2^{n'}}{\binom{n'}{n'/2}} - 1 \right) + 1.\end{aligned}$$

(By generating function manipulations.)

Counting Zeros in Random Walks

Central observation

For each even i , $0 \leq i \leq n'$:

$$\mathbb{P}(X_i = 0) = \frac{1}{i+1}.$$

Counting Zeros in Random Walks

Central observation

For each even i , $0 \leq i \leq n'$:

$$\mathbb{P}(X_i = 0) = \frac{1}{i+1}.$$

Hence, with linearity of expectation:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} =: \mathcal{H}_{n'+1}^{\text{odd}}.$$

Counting Zeros in Random Walks

Central observation

For each even i , $0 \leq i \leq n'$:

$$\mathbb{P}(X_i = 0) = \frac{1}{i+1}.$$

Hence, with linearity of expectation:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} =: \mathcal{H}_{n'+1}^{\text{odd}}.$$

(Hence $\mathcal{H}_{n'+1}^{\text{odd}}$ is equal to the two complicated sums!)

Counting Zeros in Random Walks

Central observation

For each even i , $0 \leq i \leq n'$:

$$\mathbb{P}(X_i = 0) = \frac{1}{i+1}.$$

Hence, with linearity of expectation:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} =: \mathcal{H}_{n'+1}^{\text{odd}}.$$

(Hence $\mathcal{H}_{n'+1}^{\text{odd}}$ is equal to the two complicated sums!)

Background: Distribution of s_i is given by another Pólya urn experiment (two colors, initially one ball of each color; it is known that such s_i is uniform in $\{0, 1, \dots, i\}$).

Counting Zeros in Random Walks

Central observation

For each even i , $0 \leq i \leq n'$:

$$\mathbb{P}(X_i = 0) = \frac{1}{i+1}.$$

Hence, with linearity of expectation:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} =: \mathcal{H}_{n'+1}^{\text{odd}}.$$

(Hence $\mathcal{H}_{n'+1}^{\text{odd}}$ is equal to the two complicated sums!)

Background: Distribution of s_i is given by another Pólya urn experiment (two colors, initially one ball of each color; it is known that such s_i is uniform in $\{0, 1, \dots, i\}$).

Exact Average Partitioning Cost

Have seen:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} = \mathcal{H}_{n'+1}^{\text{odd}}.$$

Exact Average Partitioning Cost

Have seen:

$$\mathbb{E}(Z_{n'}) = \sum_{\substack{0 \leq i \leq n' \\ i \text{ even}}} \frac{1}{i+1} = \mathcal{H}_{n'+1}^{\text{odd}}.$$

For “moves down from zero” use symmetry: “up from zero” has same probability as “down from zero”. No “down from zero” at position n' .

Thus:

$$\mathbb{E}(\#(\text{extra comp's}) \mid s + \ell = n') = \min(s, \ell) + \frac{1}{2} \left(\mathbb{E}(Z_{n'}) - \frac{[n' \text{ even}]}{n' + 1} \right).$$

Exact Average Partitioning Cost

Back to general situation including medium elements.

Averaging over all $\binom{n}{2}$ pivot choices and adding $\frac{4}{3}(n-2) + 1$ “forced” comparisons yields:

$$\mathbb{E}(P_n^{\text{ct}}) = \frac{3n}{2} + \frac{1}{2} \mathcal{H}_n^{\text{odd}} - \frac{19}{8} - \frac{3[n \text{ odd}]}{8n} - \frac{[n \text{ even}]}{8(n-1)}.$$

Exact Average Partitioning Cost

Back to general situation including medium elements.

Averaging over all $\binom{n}{2}$ pivot choices and adding $\frac{4}{3}(n-2) + 1$ “forced” comparisons yields:

$$\mathbb{E}(P_n^{\text{ct}}) = \frac{3n}{2} + \frac{1}{2} \mathcal{H}_n^{\text{odd}} - \frac{19}{8} - \frac{3[n \text{ odd}]}{8n} - \frac{[n \text{ even}]}{8(n-1)}.$$

We can “easily” write down the **generating function**, term by term:

$$P^{\text{ct}}(z) = \frac{3}{2(1-z)^2} + \frac{\operatorname{arctanh}(z)}{2(1-z)} - \frac{31z^2}{8(1-z)} - \frac{3+z}{8} \operatorname{arctanh}(z) - \frac{3}{2} - \frac{25z}{8}.$$

Exact Average Partitioning Cost

Back to general situation including medium elements.

Averaging over all $\binom{n}{2}$ pivot choices and adding $\frac{4}{3}(n-2) + 1$ “forced” comparisons yields:

$$\mathbb{E}(P_n^{\text{ct}}) = \frac{3n}{2} + \frac{1}{2} \mathcal{H}_n^{\text{odd}} - \frac{19}{8} - \frac{3[n \text{ odd}]}{8n} - \frac{[n \text{ even}]}{8(n-1)}.$$

We can “easily” write down the **generating function**, term by term:

$$P^{\text{ct}}(z) = \frac{3}{2(1-z)^2} + \frac{\operatorname{arctanh}(z)}{2(1-z)} - \frac{31z^2}{8(1-z)} - \frac{3+z}{8} \operatorname{arctanh}(z) - \frac{3}{2} - \frac{25z}{8}.$$

Now solving

$$C^{\text{ct}}(z) = (1-z)^3 \int_0^z (1-t)^{-6} \int_0^t (1-s)^3 (P^{\text{ct}})''(s) ds dt$$

by integration (for each term separately) gives $C^{\text{ct}}(z)$ and then a fully explicit formula for $\mathbb{E}(C_n^{\text{ct}})$.

Expected number of comparisons:

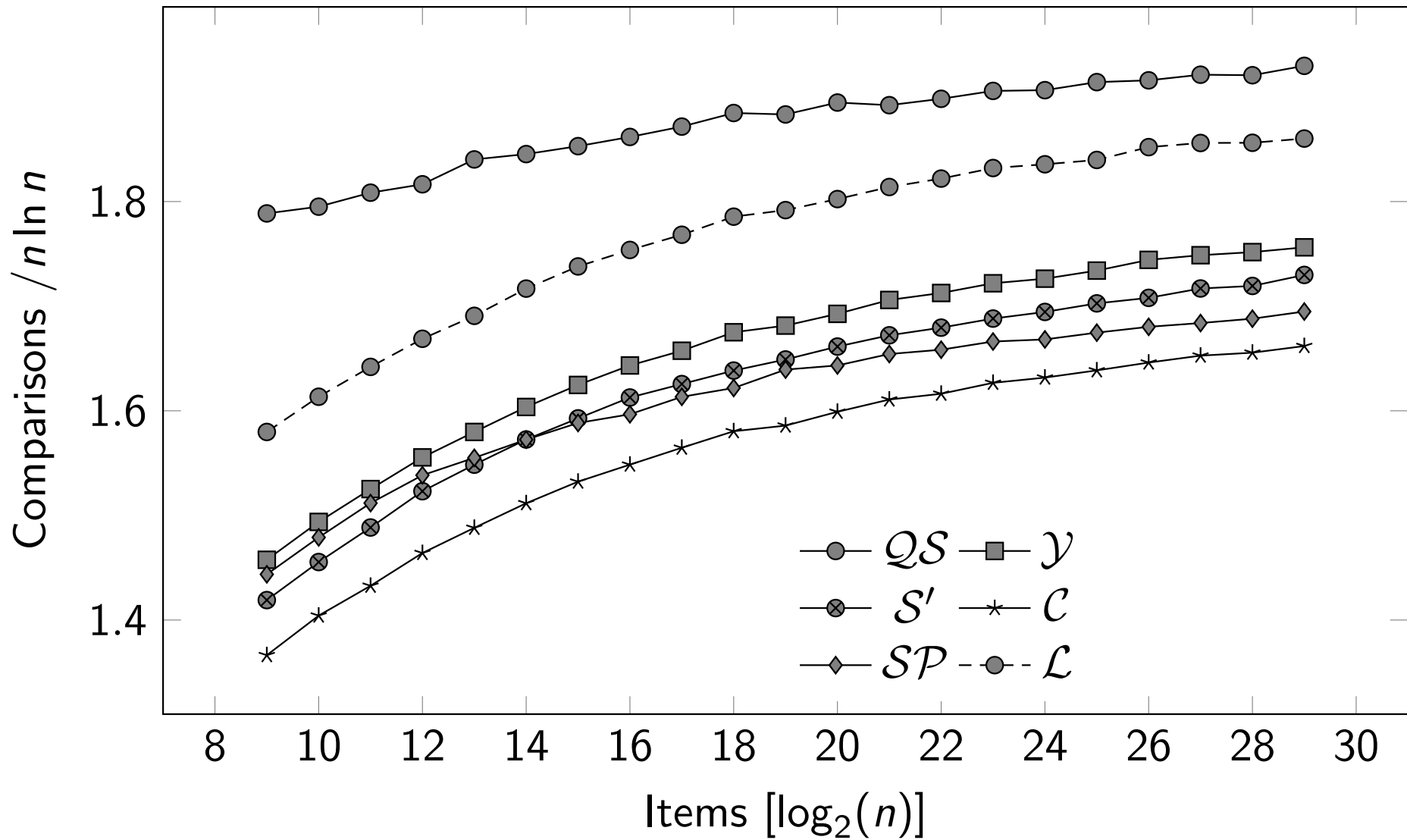
$$\mathbb{E}(C_n) = \frac{9}{5}nH_n - \frac{1}{5}nH_n^{\text{alt}} - \frac{89}{25}n + \frac{67}{40}H_n - \frac{3}{40}nH_n^{\text{alt}} - \frac{83}{800} + \frac{(-1)^n}{10} \\ - \frac{[n \text{ even}]}{320} \left(\frac{1}{n-3} + \frac{3}{n-1} \right) + \frac{[n \text{ odd}]}{320} \left(\frac{3}{n-2} + \frac{1}{n} \right),$$

with $H_n^{\text{alt}} = \sum_{1 \leq i \leq n} \frac{(-1)^i}{i}$ ($\rightarrow \ln 2$).

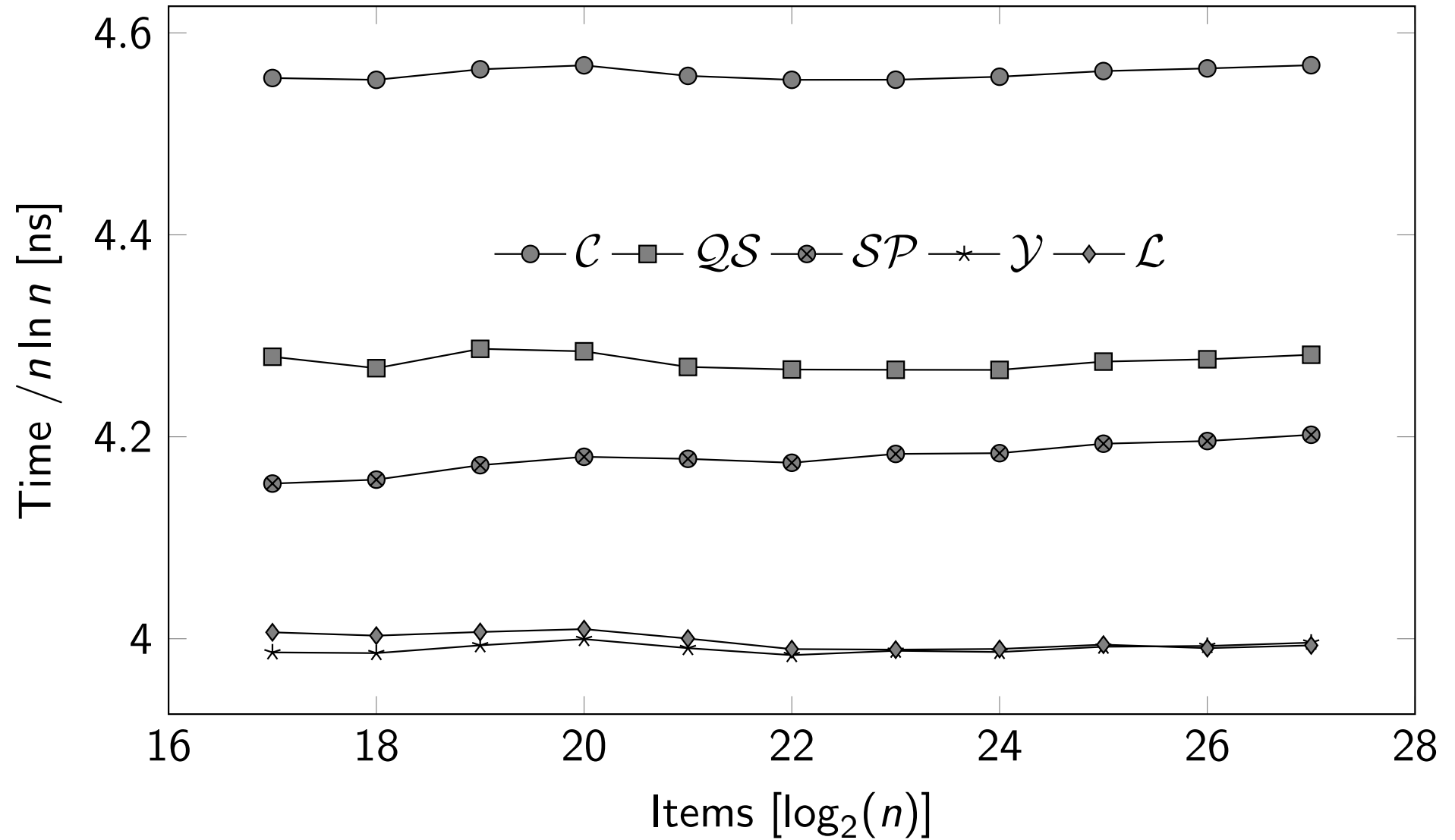
$$\mathbb{E}(C_n) = 1.8n \ln n + An + B \ln n + C + \frac{D}{n} + O\left(\frac{1}{n^2}\right),$$

where $A \approx -2.38$, $B = 1.675$, $C \approx 1.82$, $D = 0.6875$.

Comparisons in Experiments



Running Time Experiments



“Pivot Sampling”

Enhanced Classical Quicksort: Take Median-of-Three as Pivot.

“Pivot Sampling”

Enhanced Classical Quicksort: Take Median-of-Three as Pivot.

YBB: Take second and fourth smallest of five entries as pivots.

“Pivot Sampling”

Enhanced Classical Quicksort: Take Median-of-Three as Pivot.

YBB: Take second and fourth smallest of five entries as pivots.

Can show with urn model: “Count” is optimal also in this case.

“Pivot Sampling”

Enhanced Classical Quicksort: Take Median-of-Three as Pivot.

YBB: Take second and fourth smallest of five entries as pivots.

Can show with urn model: “Count” is optimal also in this case.

No exact analysis available.

“Pivot Sampling”

Enhanced Classical Quicksort: Take Median-of-Three as Pivot.

YBB: Take second and fourth smallest of five entries as pivots.

Can show with urn model: “Count” is optimal also in this case.

No exact analysis available.

Remarks on Quicksort with $k > 2$ Pivots

Input: Random permutation of $\{1, \dots, n\}$.

Remarks on Quicksort with $k > 2$ Pivots

Input: Random permutation of $\{1, \dots, n\}$.

First k entries are pivots p_1, \dots, p_k .

Can ignore sorting of the pivots.

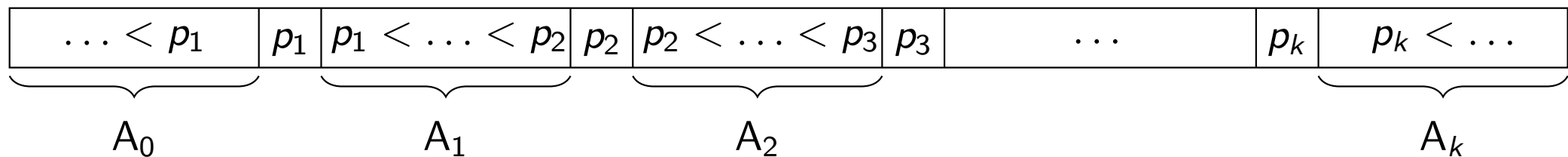
Remarks on Quicksort with $k > 2$ Pivots

Input: Random permutation of $\{1, \dots, n\}$.

First k entries are pivots p_1, \dots, p_k .

Can ignore sorting of the pivots.

Classification: Split $n - k$ remaining entries into $k + 1$ classes



Sizes of A_0, \dots, A_k : a_0, \dots, a_k .

Then sort classes recursively.

Quicksort with $k > 2$ Pivots

(Hennequin, 1991) or Roura's C.M.T.

If partitioning cost is $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then

Quicksort with $k > 2$ Pivots

(Hennequin, 1991) or Roura's C.M.T.

If partitioning cost is $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then

$$\mathbb{E}(C_n) = \frac{1}{H_{k+1} - 1} \cdot a \cdot n \ln n + O(n),$$

for $H_{k+1} = \sum_{i=1}^{k+1} (1/i)$.

Quicksort with $k > 2$ Pivots

(Hennequin, 1991) or Roura's C.M.T.

If partitioning cost is $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then

$$\mathbb{E}(C_n) = \frac{1}{H_{k+1} - 1} \cdot a \cdot n \ln n + O(n),$$

for $H_{k+1} = \sum_{i=1}^{k+1} (1/i)$.

So again: Only design and analyze classification strategy.

Quicksort with $k > 2$ Pivots

(Hennequin, 1991) or Roura's C.M.T.

If partitioning cost is $\mathbb{E}(P_n) = a \cdot n + O(n^{1-\varepsilon})$, then

$$\mathbb{E}(C_n) = \frac{1}{H_{k+1} - 1} \cdot a \cdot n \ln n + O(n),$$

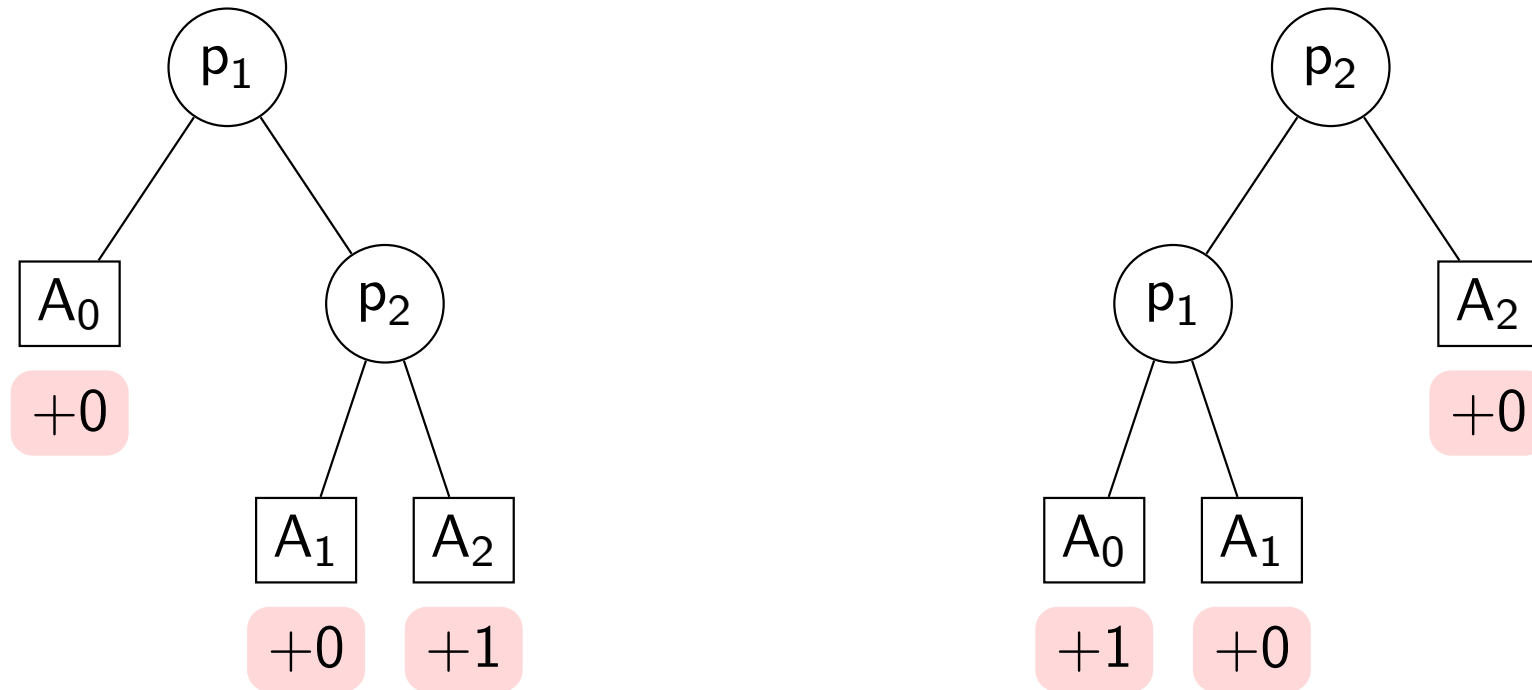
for $H_{k+1} = \sum_{i=1}^{k+1} (1/i)$.

So again: Only design and analyze classification strategy.

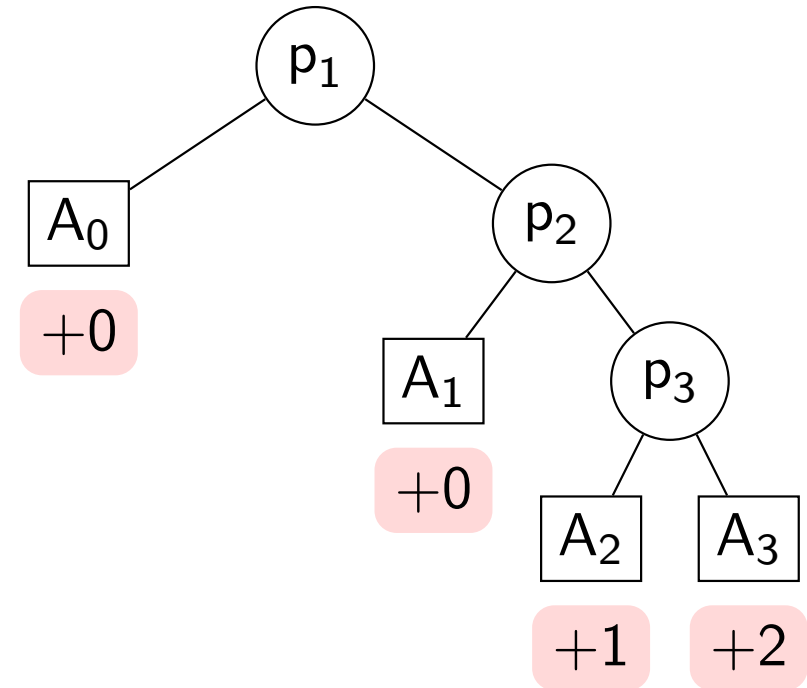
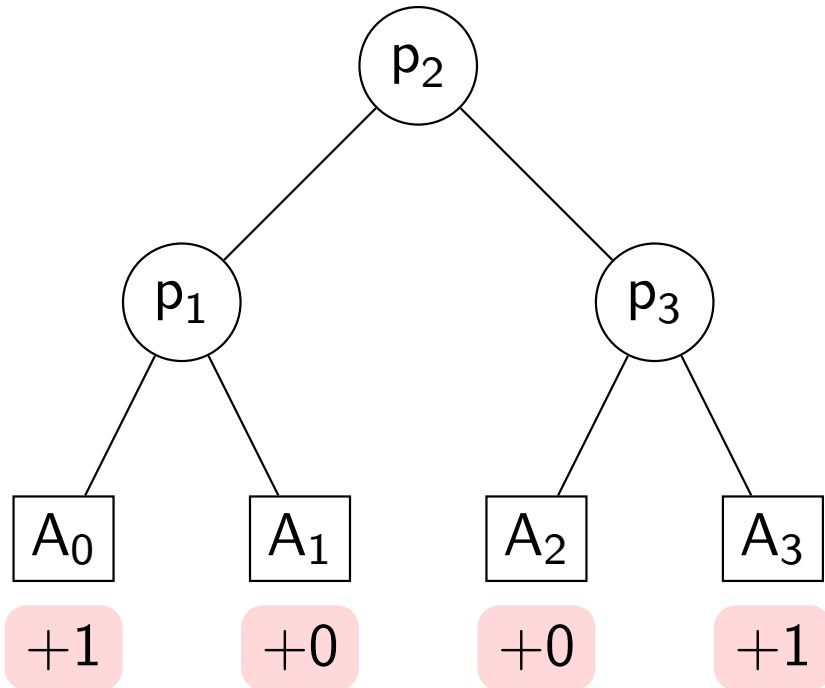
Necessary comparisons: 1 for A_0 , A_k and 2 for A_1, \dots, A_{k-1} .

Count “extra” comparisons!

Quicksort with k Pivots



Classification Using Comparison Trees



$$\text{cost}_T(a_0, a_1, a_2, a_3) = a_0 + a_3.$$

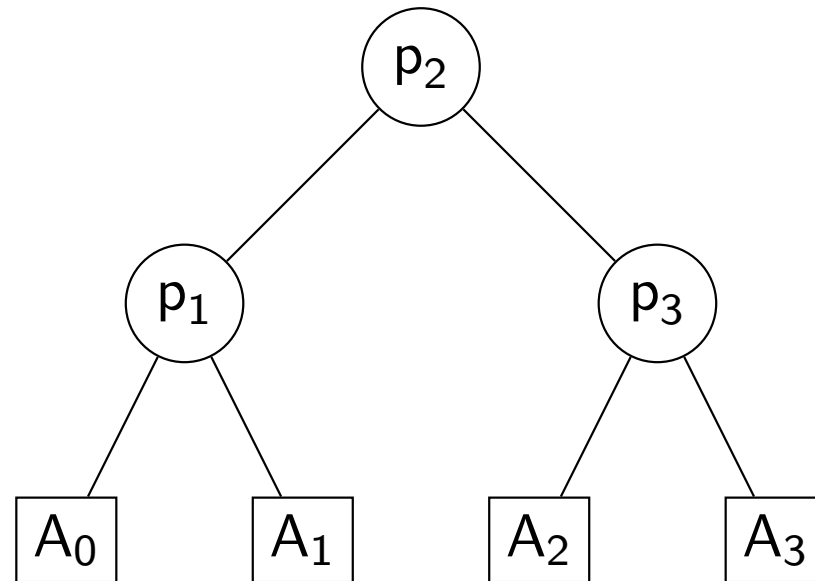
$$\text{cost}_T(a_0, a_1, a_2, a_3) = a_2 + 2a_3.$$

Which tree is best depends on class sizes $|A_0|, \dots, |A_k|$.

Good 3-Pivot Algorithm

Balanced Tree (Kushagra, López-Ortiz, Qiao, Munro 2014)

Always compare with p_2 first.



$1.846n \ln n$ comparisons.

With good implementation: Quite good practical performance (even slightly better than YBB algorithm).

Optimal: Count

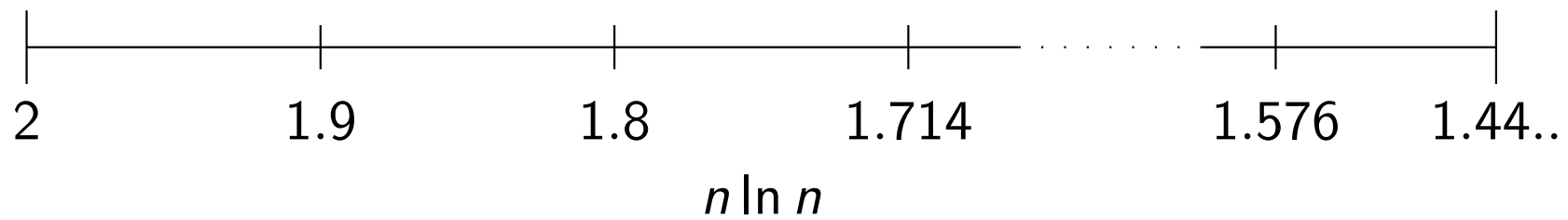
- Keep track of sizes $|A_0^i|, |A_1^i|, \dots, |A_k^i|$ of elements seen by round i .
- Use comparison tree optimal for estimated class sizes.

Optimal: Count

- Keep track of sizes $|A_0^i|, |A_1^i|, \dots, |A_k^i|$ of elements seen by round i .
- Use comparison tree optimal for estimated class sizes.

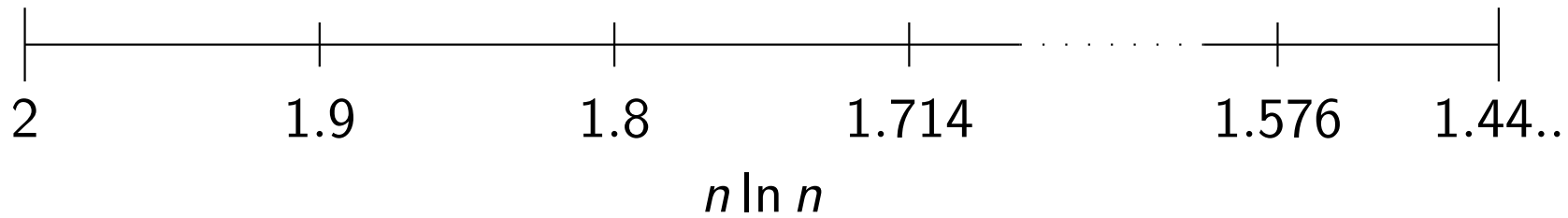
Determining optimal comparison tree is very expensive – usually impractical.

Comparison Counts: Summary

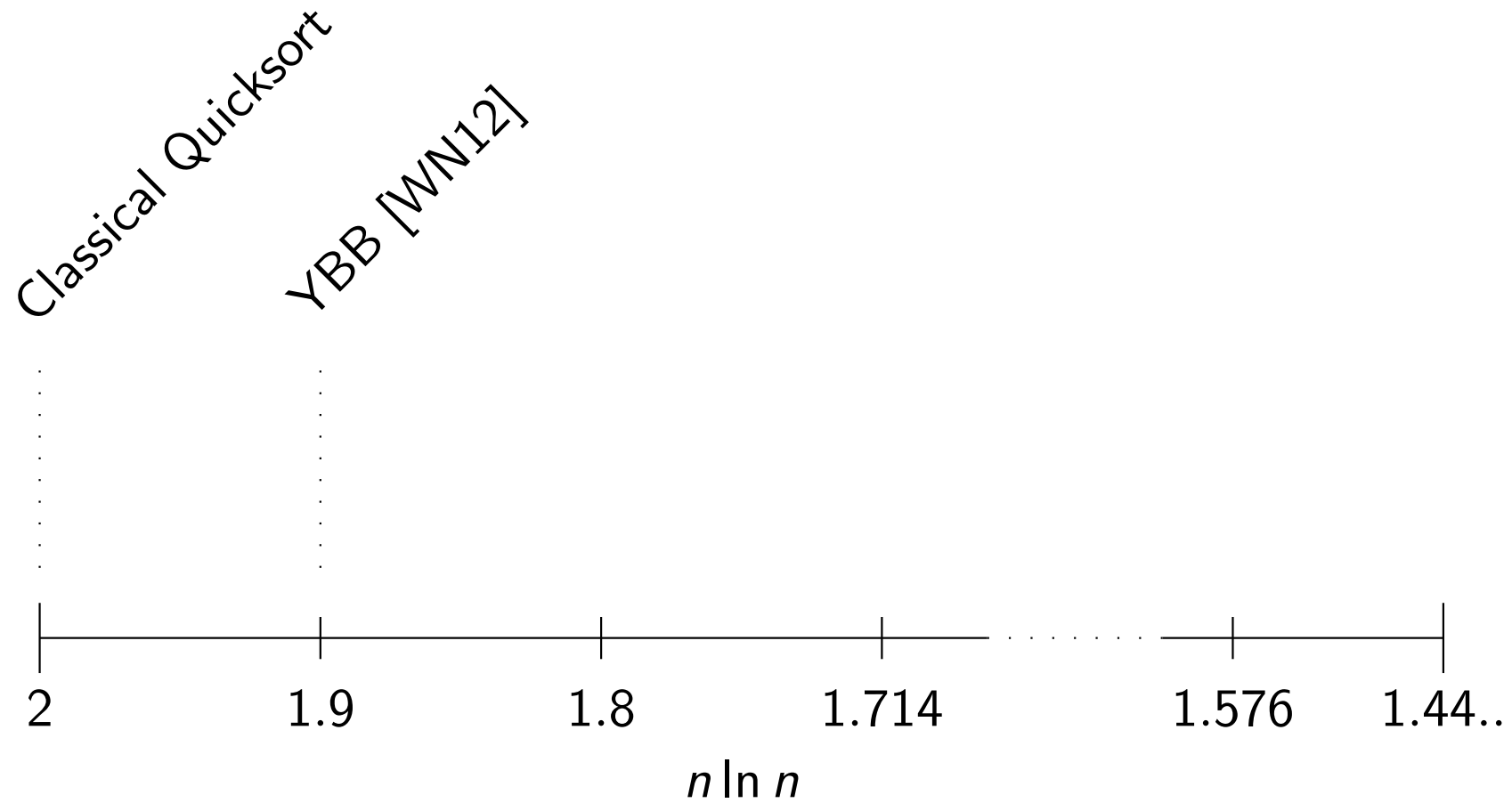


Comparison Counts: Summary

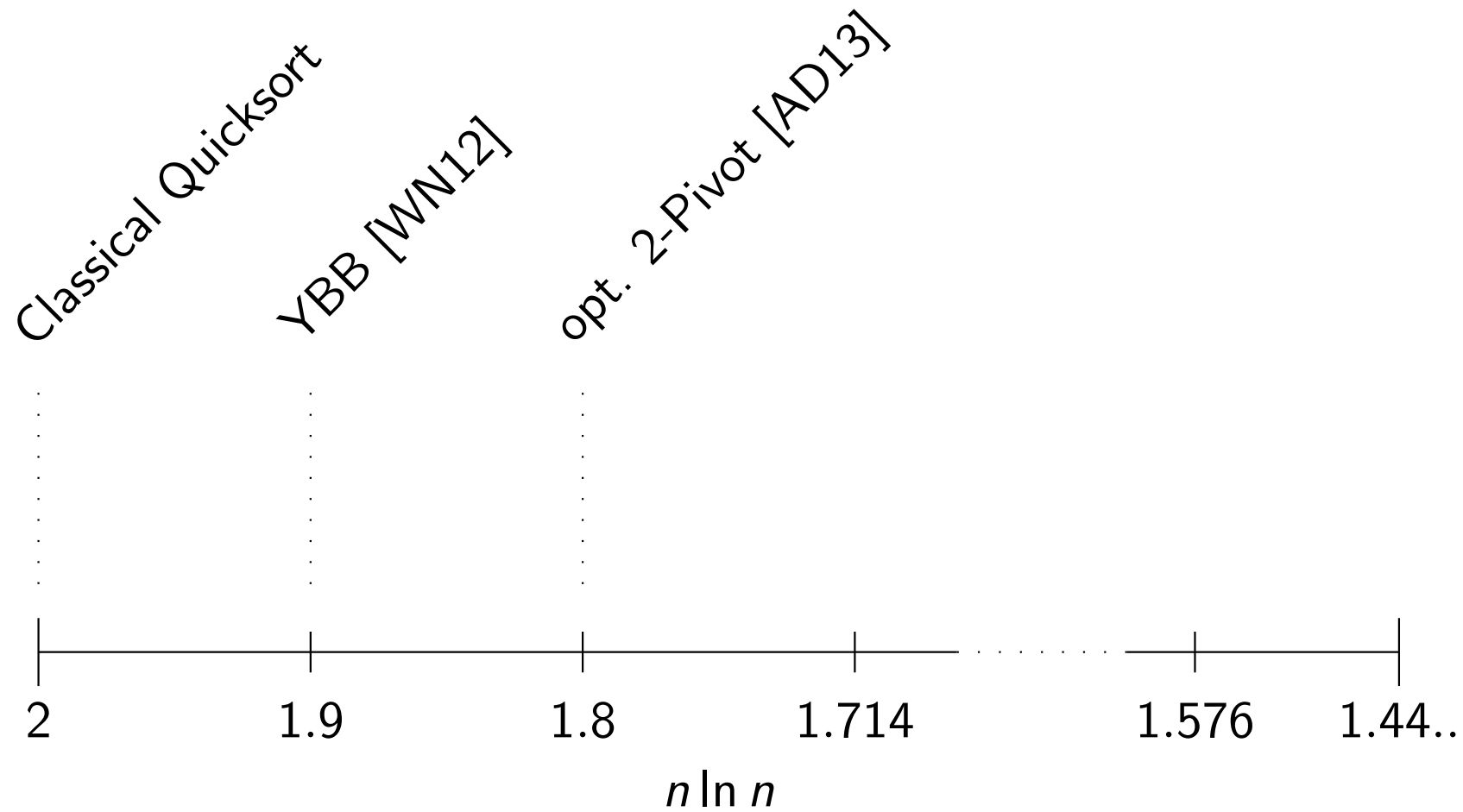
Classical Quicksort



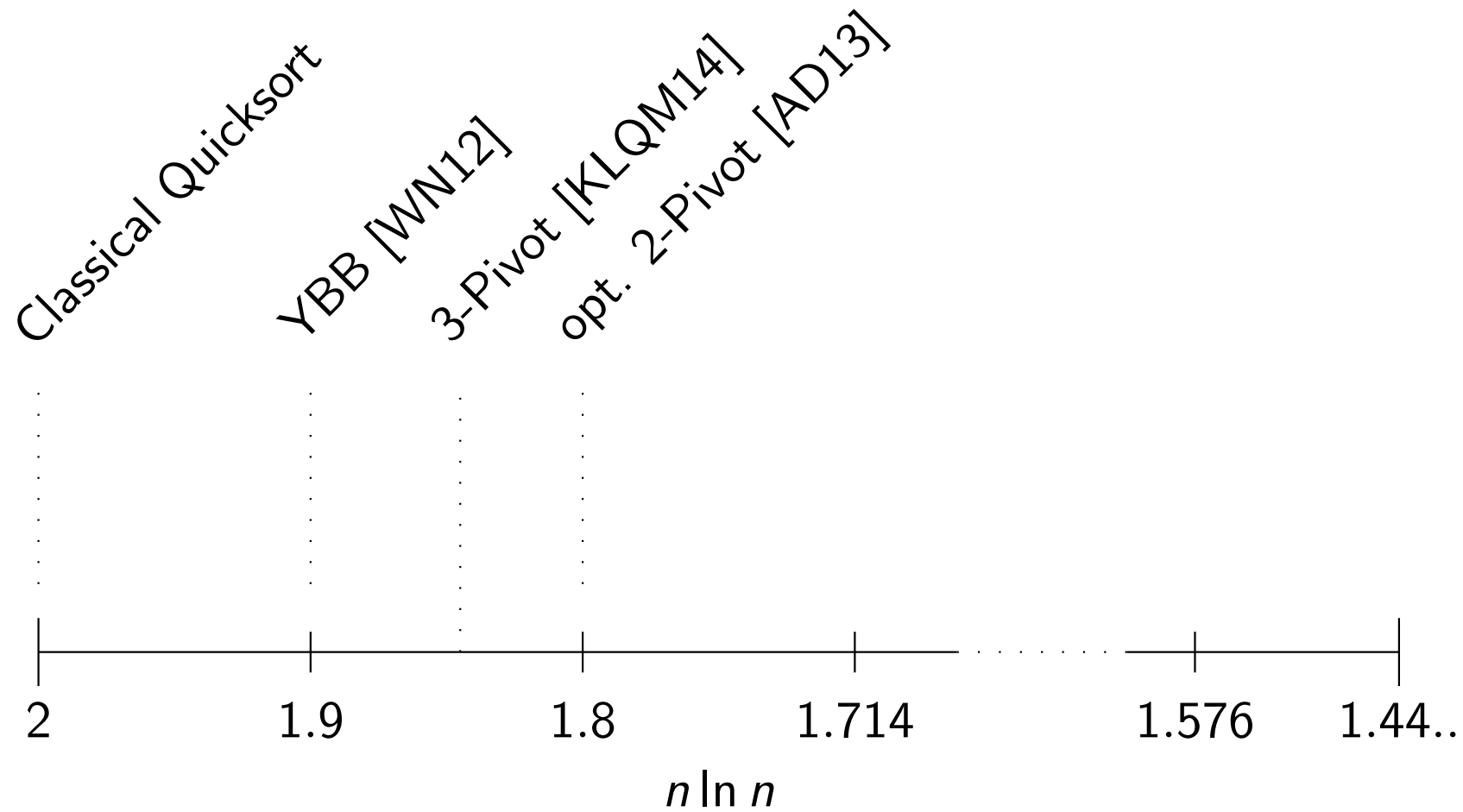
Comparison Counts: Summary



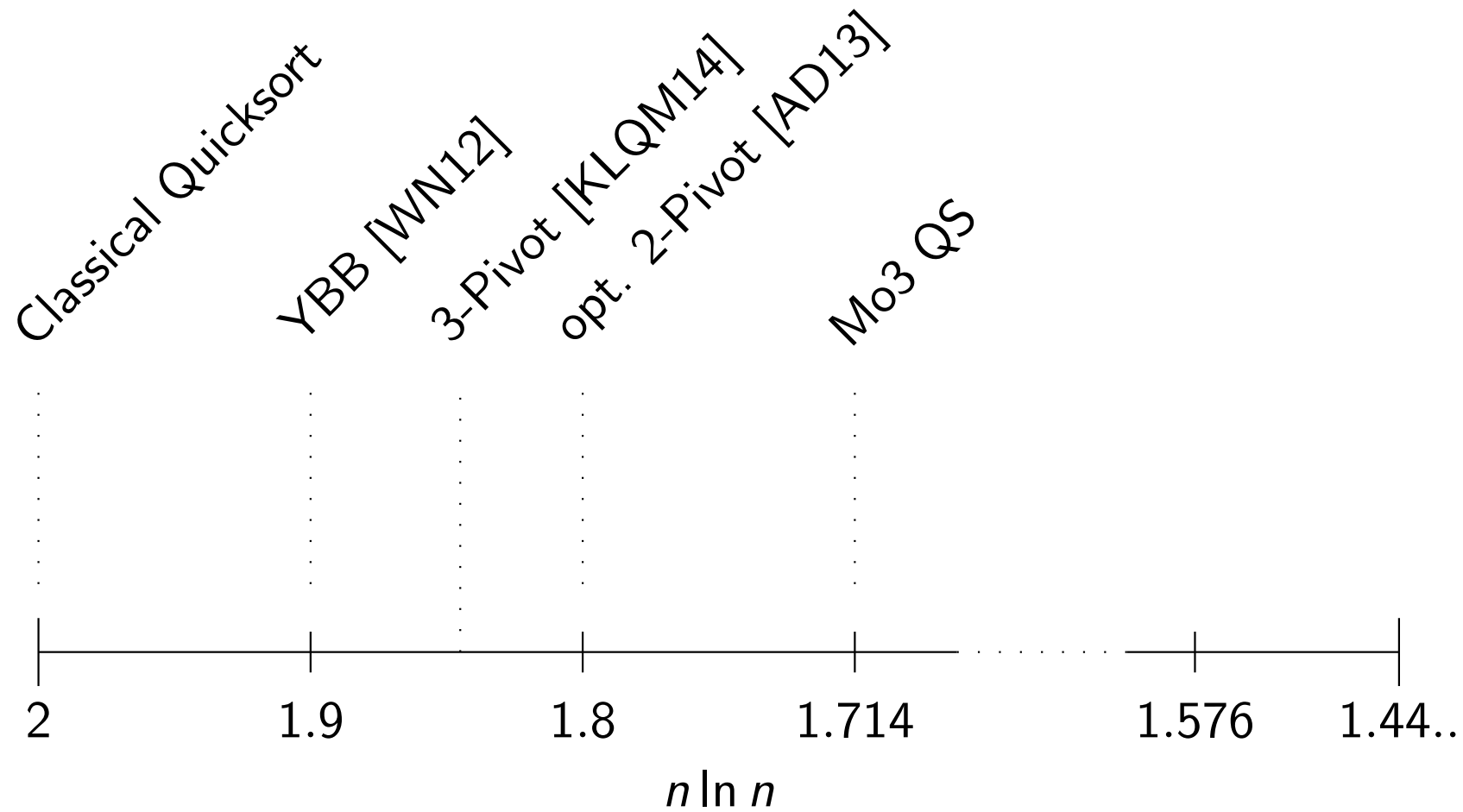
Comparison Counts: Summary



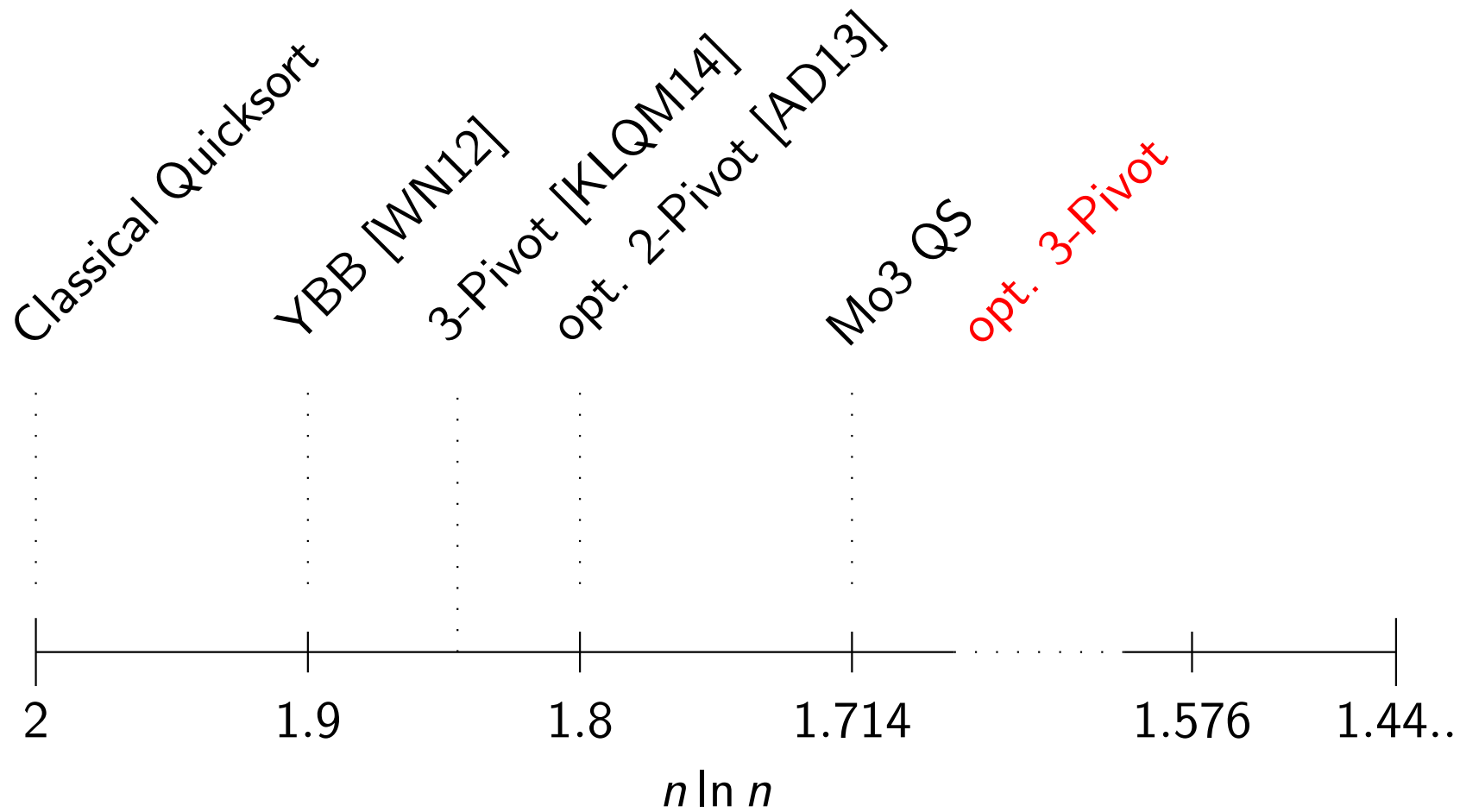
Comparison Counts: Summary



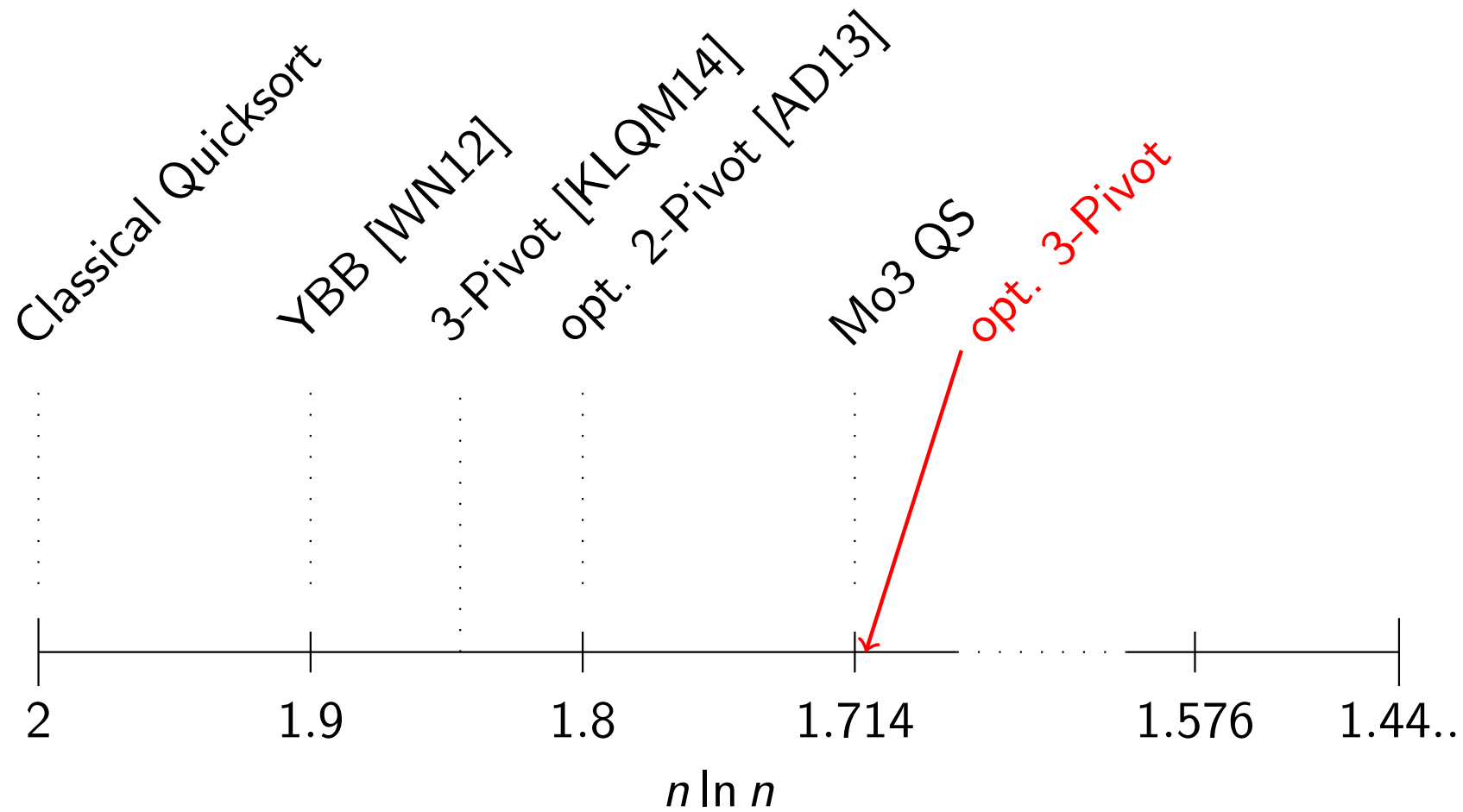
Comparison Counts: Summary



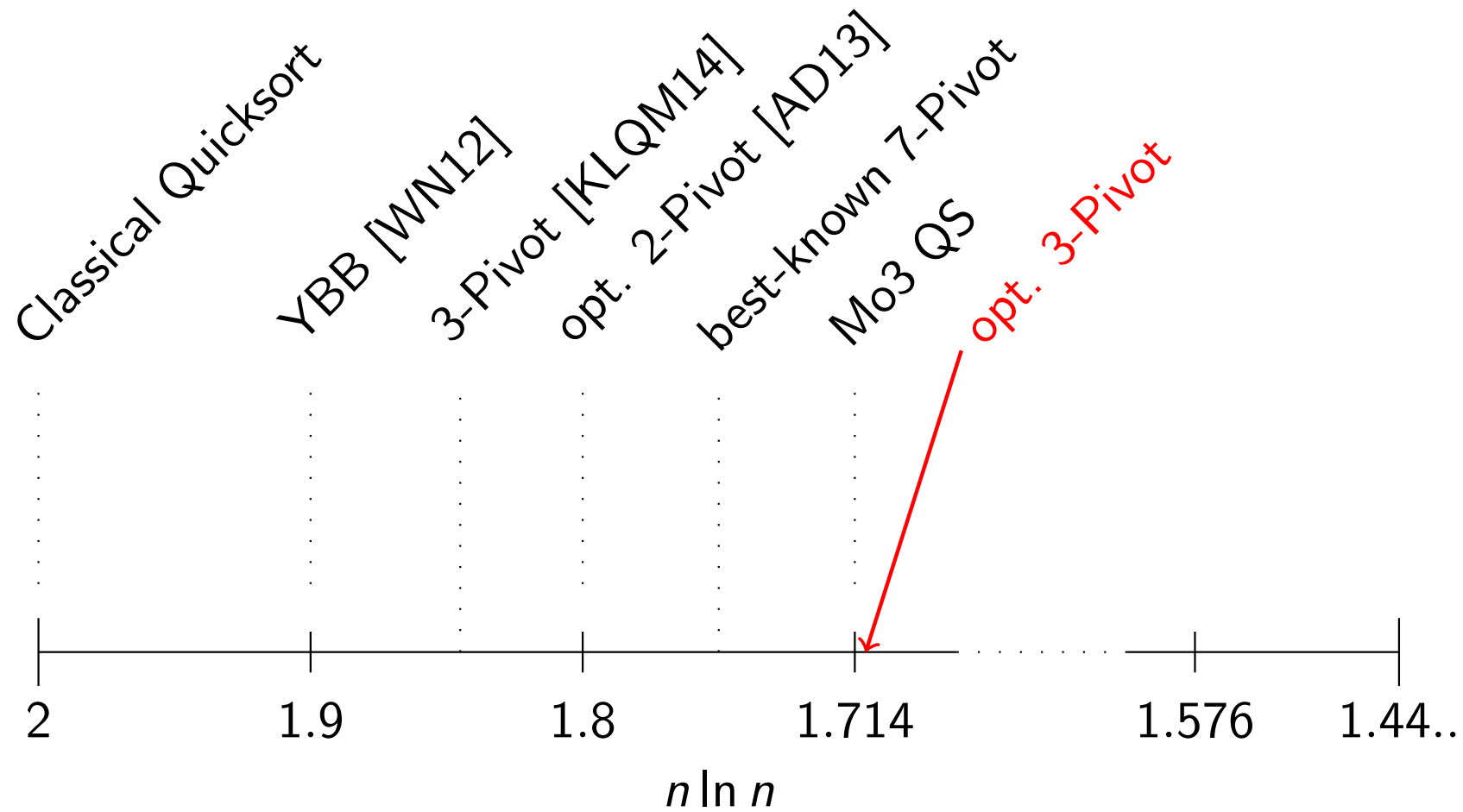
Comparison Counts: Summary



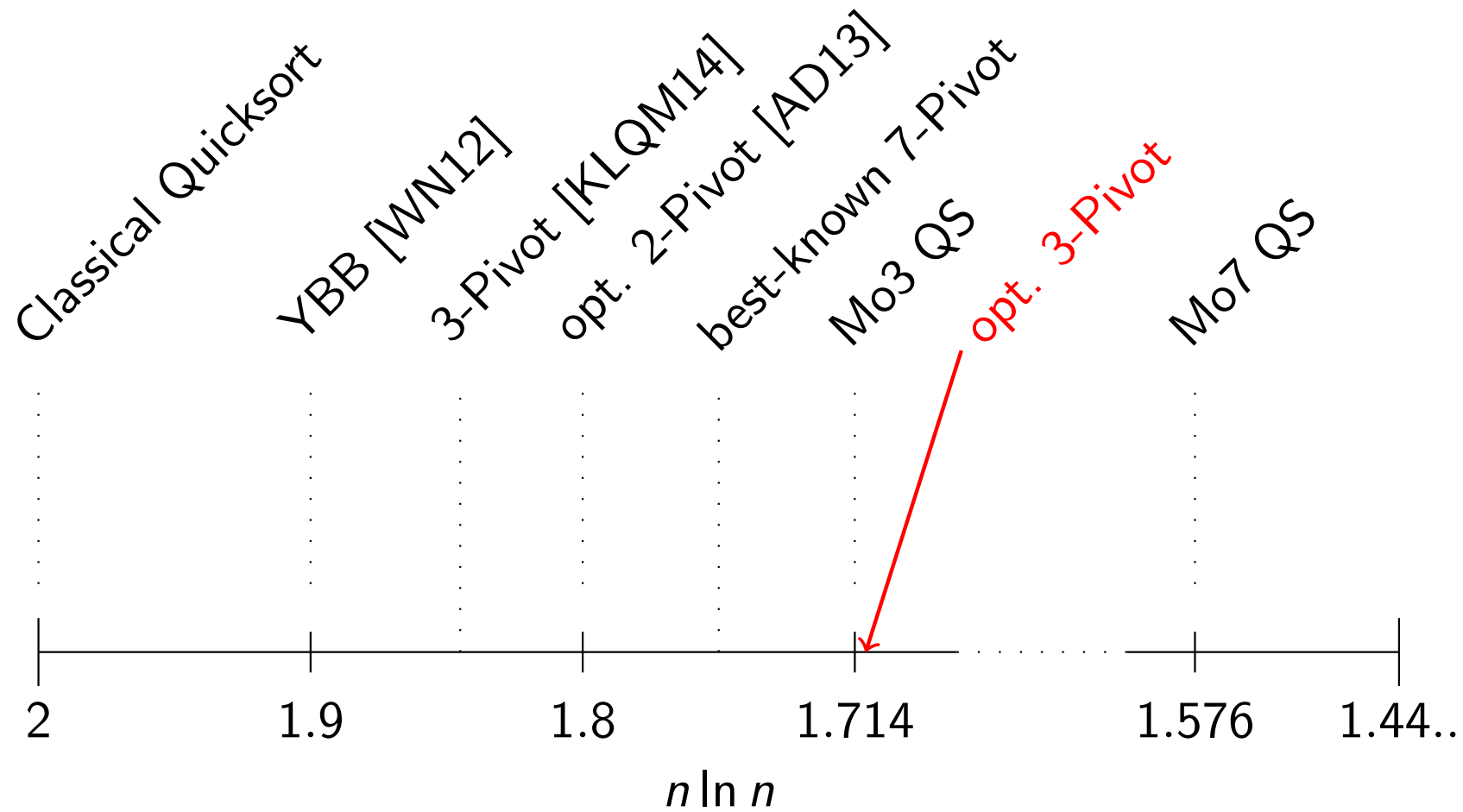
Comparison Counts: Summary



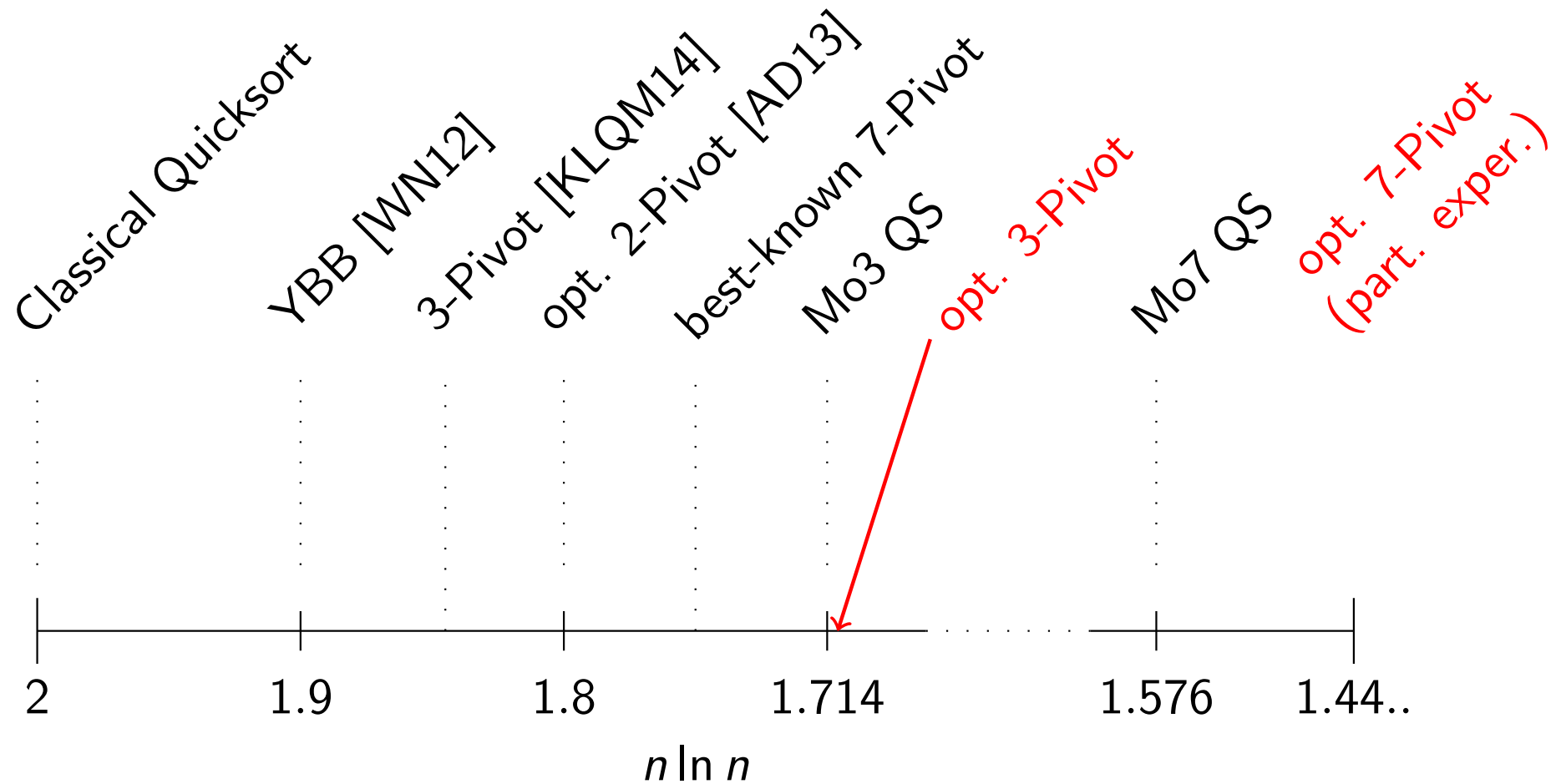
Comparison Counts: Summary



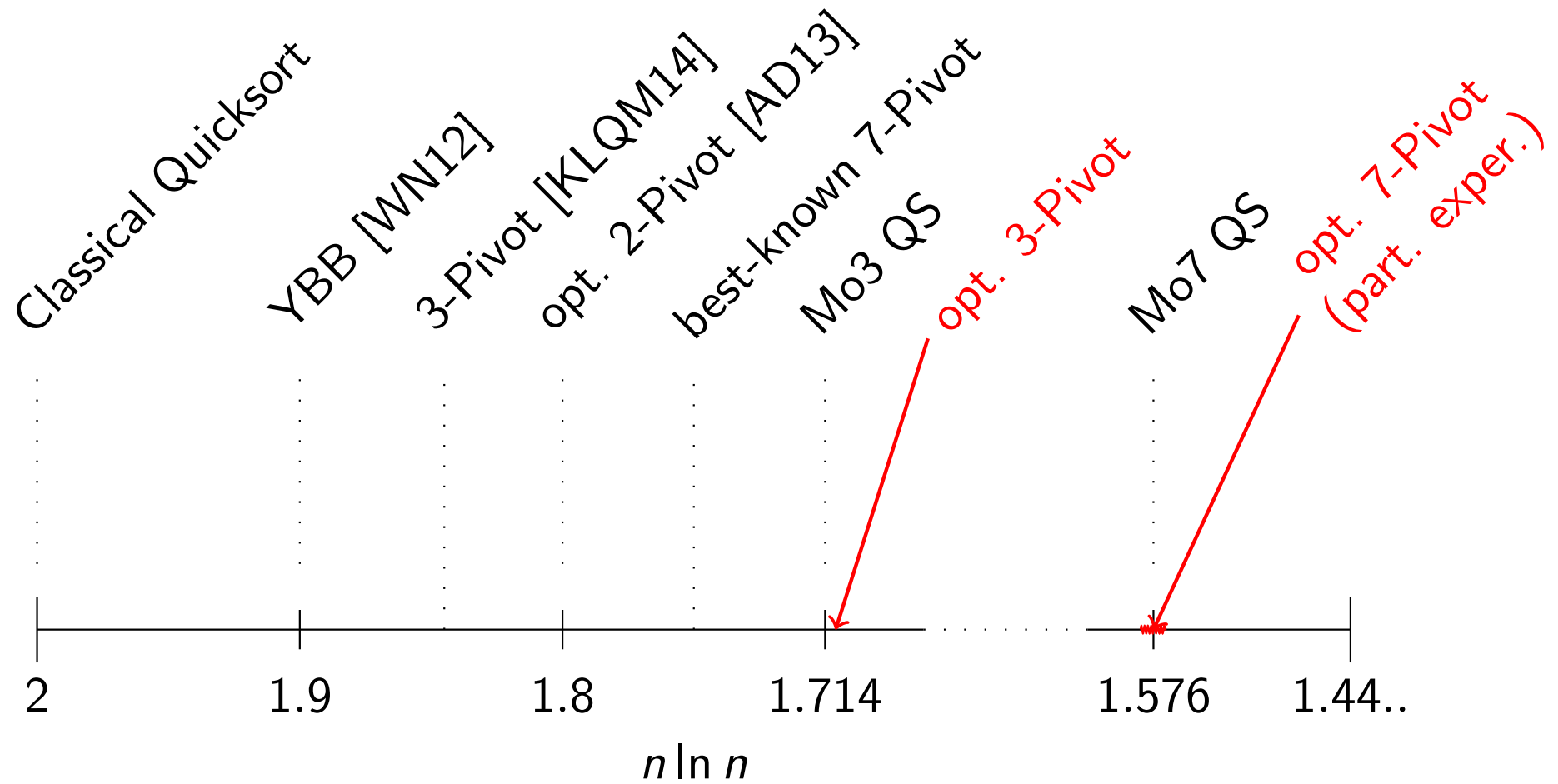
Comparison Counts: Summary



Comparison Counts: Summary



Comparison Counts: Summary



Summary and Research Directions

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy
- Optimal strategy for $k > 2$, even with pivot sampling: Count.

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy
- Optimal strategy for $k > 2$, even with pivot sampling: Count.
- Ongoing/Future work: Analyze Count with $k > 4$ pivots exactly. (Beware: Not necessarily practically useful.)

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy
- Optimal strategy for $k > 2$, even with pivot sampling: Count.
- Ongoing/Future work: Analyze Count with $k > 4$ pivots exactly. (Beware: Not necessarily practically useful.)
- Understand behavior of Count if repeated elements are present.

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy
- Optimal strategy for $k > 2$, even with pivot sampling: Count.
- Ongoing/Future work: Analyze Count with $k > 4$ pivots exactly. (Beware: Not necessarily practically useful.)
- Understand behavior of Count if repeated elements are present.
- Optimal algorithms with regard to other complexity measures, more relevant for practical performance.

Summary and Research Directions

- Easy-to-use formula for partitioning cost for dual-pivot quicksort
- $1.8n \ln n + O(n)$ comparisons is optimal for two pivots
- Exact formulas for the optimal dual-pivot strategy
- Optimal strategy for $k > 2$, even with pivot sampling: Count.
- Ongoing/Future work: Analyze Count with $k > 4$ pivots exactly. (Beware: Not necessarily practically useful.)
- Understand behavior of Count if repeated elements are present.
- Optimal algorithms with regard to other complexity measures, more relevant for practical performance.

Many thanks for your attention.

Literature

- Martin Aumüller, Martin Dietzfelbinger: Optimal Partitioning for Dual-Pivot Quicksort. *ACM Trans. Algorithms* 12(2): 18:1-18:36 (2016)
- Martin Aumüller, Martin Dietzfelbinger, Pascal Klaue: How Good Is Multi-Pivot Quicksort? *ACM Trans. Algorithms* 13(1): 8:1-8:47 (2016)
- Martin Aumüller, Martin Dietzfelbinger, Clemens Heuberger, Daniel Krenn, Helmut Prodinger: Counting Zeros in Random Walks on the Integers and Analysis of Optimal Dual-Pivot Quicksort. *CoRR* abs/1602.04031 (2016) (Proceedings of AofA'16)
- Martin Aumüller, Martin Dietzfelbinger, Clemens Heuberger, Daniel Krenn, Helmut Prodinger: Counting Zeros in Random Walks on the Integers and Analysis of Optimal Dual-Pivot Quicksort. *CoRR* abs/1602.04031 (2016)

Literature (cont.)

- Sebastian Wild, Markus E. Nebel, Ralph Neininger: Average Case and Distributional Analysis of Dual-Pivot Quicksort. ACM Trans. Algorithms 11(3): 22:1-22:42 (2015) (Preliminary version in ESA 2012.)
- Sebastian Wild, Markus E. Nebel, Raphael Reitzig, Ulrich Laube: Engineering Java 7's Dual Pivot Quicksort Using MaLiJan. ALENEX 2013: 55-69
- Markus E. Nebel, Sebastian Wild, Conrado Martínez: Analysis of Pivot Sampling in Dual-Pivot Quicksort: A Holistic Analysis of Yaroslavskiy's Partitioning Scheme. Algorithmica 75(4): 632-683 (2016)
- Conrado Martínez, Markus E. Nebel, Sebastian Wild: Analysis of Branch Misses in Quicksort. ANALCO 2015: 114-128
- Sebastian Wild: Why Is Dual-Pivot Quicksort Fast? CoRR abs/1511.01138 (2015)

Literature (cont.)

- Sebastian Wild: Dual-Pivot Quicksort and Beyond: Analysis of Multiway Partitioning and Its Practical Potential. PhD Thesis, Universität Kaiserslautern, 2016
- Shrinu Kushagra, Alejandro López-Ortiz, Aurick Qiao, J. Ian Munro: Multi-Pivot Quicksort: Theory and Experiments. ALENEX 2014: 47-60
- Charles A. R. Hoare, Quicksort, Comput. J. 5 (1962), no. 1, 10–15.
- Pascal Hennequin: Analyse en moyenne d'algorithmes, tri rapide et arbres de recherche. PhD Thesis, Palaiseau, Ecole polytechnique, 1991
- Robert Sedgwick. Quicksort. PhD thesis, Stanford University, Stanford, CA, May 1975. Stanford Computer Science Report STAN-CS-75-492.

Literature (cont.)

- Vladimir Yaroslavskiy, Replacement of quicksort in `java.util.array`s with new dual-pivot quicksort, <http://mail.openjdk.java.net/pipermail/core-libs-dev/2009-September/002630.html>, 2009, Archived version of the discussion in the OpenJDK mailing list.
- Philippe Flajolet and Robert Sedgewick, *Analytic combinatorics*, Cambridge University Press, Cambridge, 2009.