# DNA Rearrangement: PrefixSort and Greedy Approaches

Your Name

2024-11-22

## 1 Objectives

This project focuses on sequence rearrangement algorithms, which aim to sort or rearrange permutations using minimal operations. These problems are crucial in areas like computational biology (e.g., genome rearrangement) and optimization. The study involves analyzing two algorithms that address these challenges with distinct approaches and objectives.

Breakpoint Reversal Algorithm

The Breakpoint Reversal Algorithm reduces breakpoints—positions where consecutive elements are not adjacent in value—by applying reversal operations between them. It identifies breakpoints, generates candidate reversals, and evaluates their quality based on how effectively they reduce the total number of breakpoints.

Prefix Sort Algorithm

The Prefix Sort Algorithm sorts permutations by iteratively applying prefix reversals, which reverse the first k elements of the sequence. It works by locating out-of-place elements, determining their current position, and performing a prefix reversal to move them to the correct spot, repeating this until the sequence is sorted.

## 2 Experimental Setup

The experiments are designed to evaluate the performance of the Breakpoint Reversal Algorithm and the Prefix Sort Algorithm. The algorithms will be tested on various permutations of different lengths, assessing their efficiency in terms of the number of operations performed and the runtime required. The main parameters include the size of the permutations (ranging from small to large sequences) and the type of permutation (randomized or structured). The quality of the solutions will also be analyzed, particularly the reduction in breakpoints and the total reversals applied.

The experiments will be conducted in a controlled computational environment to ensure consistency and repeatability. All algorithms will be implemented in Java, and the results will be recorded for comparison. Measurements will include the number

of operations performed, runtime (in milliseconds), and, for the Breakpoint Reversal Algorithm, the change in the number of breakpoints. (see Table \ref{tab:conf}).

Table 1: Computational environment considered.

| | |
|---|---|
| CPU | Intel Core i7-12700H, 16GB RAM |
| OS | Windows 11 Pro, Version 22H2 |
| Java | Java OpenJDK 17.0.8 |

## 3 Empirical Results

A summary of the experimental results is provided in Tables 2 and 3 in the Appendix, along with the statistical fitting of the data to different growth models.

Describe the results, in particular Figure 1.

Figure 1 illustrates the normalized number of reversals (operations per sequence length) for both the Breakpoint Reversal and Prefix Sort algorithms as the permutation length increases. The y-axis represents the ratio of operations to sequence length, and the x-axis represents the permutation length, denoted by lambda. The data is fitted to growth models that highlight the convergence behavior of the algorithms as sequence length increases.

The Prefix Sort Algorithm, shown in red, exhibits slightly higher normalized operation counts compared to the Breakpoint Reversal Algorithm, shown in blue. Both algorithms demonstrate a decreasing rate of additional operations with increasing sequence length, approaching asymptotic limits. The statistical fits confirm this trend, with Prefix Sort following

$1-(1.31\ K - 0.688)$

$1-(1.31\ K - 0.688)$

and Breakpoint Reversal following (K for lambda)

$1-(1.35\ K\ - 0.587)$

1−(1.35 K − 0.587).

As sequence length grows, the operations required per element stabilize near 1 for both algorithms. However, the Prefix Sort curve approaches this limit more steeply, indicating its efficiency in normalizing the operation count for larger permutations. Breakpoint Reversal, while slower to converge, displays consistent behavior over longer sequences due to its heuristic-based approach to reducing breakpoints.

This comparison highlights the distinct optimization strategies of the two algorithms and their effectiveness as permutation length increases. While Prefix Sort is slightly more efficient in operation count per element, the Breakpoint Reversal Algorithm remains competitive, especially for intermediate sequence lengths.

## Warning: package 'ggnewscale' was built under R version 4.4.2

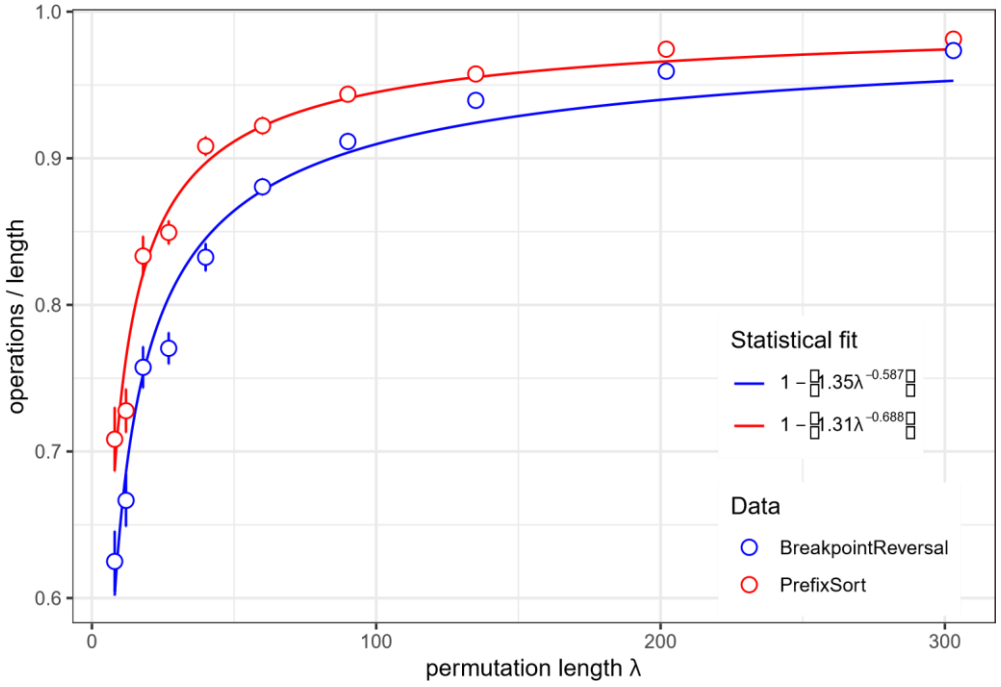


Figure 1: Reversals (normalized per number of elements) for increasing sequence length

## 4 Discussion

Provide your interpretation of the results: discuss whether the results match the theoretical predictions, whether some algorithm is better in practice than others, etc.

The results depicted in Figure 1 largely align with theoretical predictions for the Prefix Sort and Breakpoint Reversal algorithms. Both algorithms demonstrate asymptotic behavior, with their normalized operation counts converging toward a value near 1 as the permutation length increases. This indicates that both algorithms scale effectively for larger inputs, requiring roughly one operation per element to achieve their respective goals.

Comparison to Theoretical Predictions
The observed behavior is consistent with theoretical expectations:

Prefix Sort: The results reflect the algorithm's efficient sorting mechanism, where each element is positioned correctly in a bounded number of prefix reversals. The rapid convergence of the normalized operation count (as seen in the steeper curve) matches its theoretical efficiency, particularly for unsigned permutations.
Breakpoint Reversal: The algorithm performs more operations at shorter sequence lengths but converges more slowly to its asymptotic limit. This aligns with its heuristic nature, as it relies on finding and minimizing breakpoints incrementally, which may involve more operations in smaller or highly disordered permutations.
Practical Comparison
While both algorithms are effective, Prefix Sort appears to perform slightly better in practice:

Normalized Operation Count: Prefix Sort consistently requires fewer operations per element than Breakpoint Reversal across all sequence lengths, as shown by the red curve being above the blue curve.
Convergence: Prefix Sort converges faster to its theoretical limit, making it more efficient for large-scale problems.
Specialization: Breakpoint Reversal's focus on reducing breakpoints makes it more suitable for applications where minimizing specific disorder metrics (like breakpoints) is critical, even if it requires additional operations.
General Insights
In practice, the choice of algorithm depends on the problem's specific requirements:

Use Prefix Sort when the primary objective is sorting a sequence quickly and efficiently.
Opt for Breakpoint Reversal when the problem prioritizes breakpoint reduction, even at the cost of additional operations.
Overall, while Prefix Sort shows a slight edge in practical efficiency, both algorithms perform well and match their theoretical scaling, demonstrating their applicability in sequence rearrangement tasks.

# A Appendix

## A.1 Data Summary

Table 2: Summary of the experimental results for PrefixSort. The mean and standard error are shown for each length.

| **length $\lambda$** | **operations** | **std. err.** |
|---|---|---|
| 8 | 0.708 | 2.19e-02 |

| | | |
|---|---|---|
| 12 | 0.728 | 1.49e-02 |
| 18 | 0.833 | 1.36e-02 |
| 27 | 0.849 | 8.12e-03 |
| 40 | 0.908 | 6.49e-03 |
| 60 | 0.922 | 5.84e-03 |
| 90 | 0.944 | 4.12e-03 |
| 135 | 0.958 | 2.77e-03 |
| 202 | 0.974 | 1.90e-03 |
| 303 | 0.981 | 1.19e-03 |

Table 3: Summary of the experimental results for BreakpointReversal. The mean and standard error are shown for each length.

| **length $\lambda$** | **operations** | **std. err.** |
|---|---|---|
| 8 | 0.625 | 2.08e-02 |
| 12 | 0.667 | 1.79e-02 |
| 18 | 0.757 | 1.42e-02 |
| 27 | 0.770 | 1.09e-02 |
| 40 | 0.832 | 9.45e-03 |
| 60 | 0.881 | 5.60e-03 |
| 90 | 0.911 | 5.37e-03 |
| 135 | 0.940 | 3.78e-03 |
| 202 | 0.959 | 2.64e-03 |
| 303 | 0.973 | 1.55e-03 |

## A.2 Model Fitting

```
## PrefixSort
## Formula: (1 - ops) ~ a * length^b
##
## Parameters:
##              Estimate Std. Error t value Pr(>|t|)
## a 1.30570              0.18836         6.932 0.000121 ***
## b -0.68790                 0.05355 -12.845 1.27e-06 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01719 on 8 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 3.169e-06
##
## BreakpointReversal
## Formula: (1 - ops) ~ a * length^b
##
## Parameters:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## a 1.35132              0.16127         8.379 3.12e-05 ***
## b -0.58740                0.04242 -13.846 7.16e-07 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02061 on 8 degrees of freedom
##
## Number of iterations to convergence: 9
## Achieved convergence tolerance: 1.322e-06
```