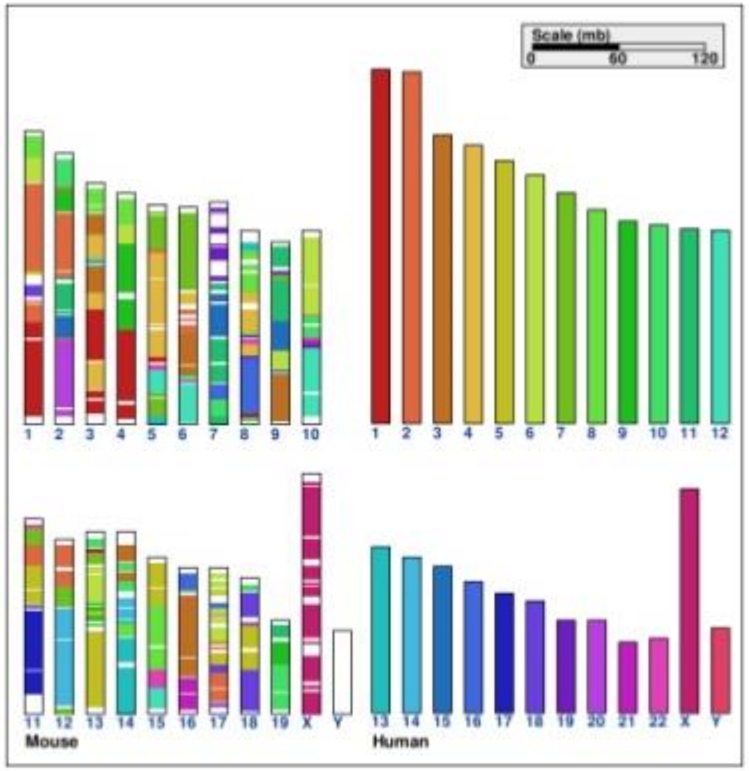
Synteny Blocks

MPA-PRG: Programming in Bioinformatics

Exercise 8

Synteny Blocks

- a group of consecutive genes that do not have any structural aberrations (insertions, deletions, translocations of other genes)



[5]Sinha and Meller, BMC Bioinformatics 2007

Comparison of Synteny Blocks

- compare the chromosome set of two organisms
- find the synteny blocks
- compare the positions of synteny blocks
- infer the possible steps of chromosomal aberrations leading from an unknown ancestor to the current descendants
- comparison of synteny blocks within several organisms can give a rough picture of the genome of their mutual ancestor

Sorting by reversals

- let $\pi = \pi_1\ \pi_2 \dots \pi_n$ be a permutation of n distinct numbers and $1 \leq \pi_i \leq n$

- the reversal $\rho = \rho(i, j)$ for $1 \leq i < j \leq n$ applied to π reverses the values of $\pi_i\ \pi_{i+1} \dots \pi_{j-1}\ \pi_j$ and thus transforms π into a permutation $\pi \cdot \rho(i, j) = \pi_1 \dots \pi_j\ \pi_{j-1} \dots \pi_{i+1}\ \pi_i \dots \pi_n$
- for example: π = 1 2 4 3 7 5 6, then $\pi \cdot \rho(3, 6)$ = 1 2 5 7 3 4 6

- the identity permutation I is the permutation where each $\pi_i = i$ for $1 \leq i \leq n$
- the distance $d(\pi, I)$ between π and I is the minimum number of reversals ρ that transform π to I

Breakpoint Sort

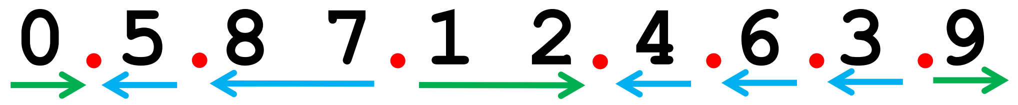
- greedy algorithm
- minimize breakpoints between synteny blocks
- adjacency = a pair of adjacent elements that are consecutive
- breakpoint = a pair of adjacent elements that are not consecutive

```
1  2.4  3
```

- block reversals are performed in such a way, that the number of breakpoints is reduced (or remains the same)
- each reversal removes at most 2 breakpoints
- number of reversals = hypothetical number of chromosomal changes

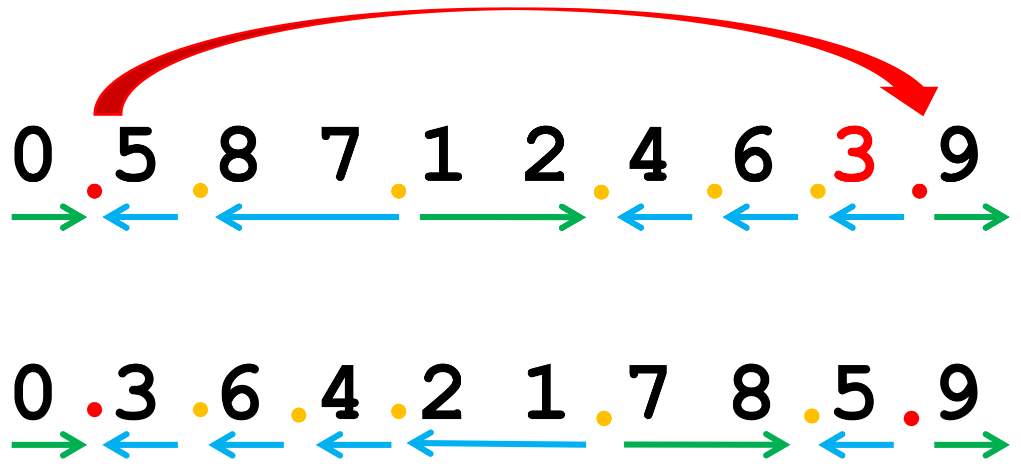
Breakpoint Sort – Steps

- extend the permutation $\pi_1 \ldots \pi_n$ by $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ on the ends, π_0 and π_{n+1} never change their positions
- mark the ascending and descending parts of the permutation
- reverse the descending part that will lead to the largest reduction in the number of breakpoints

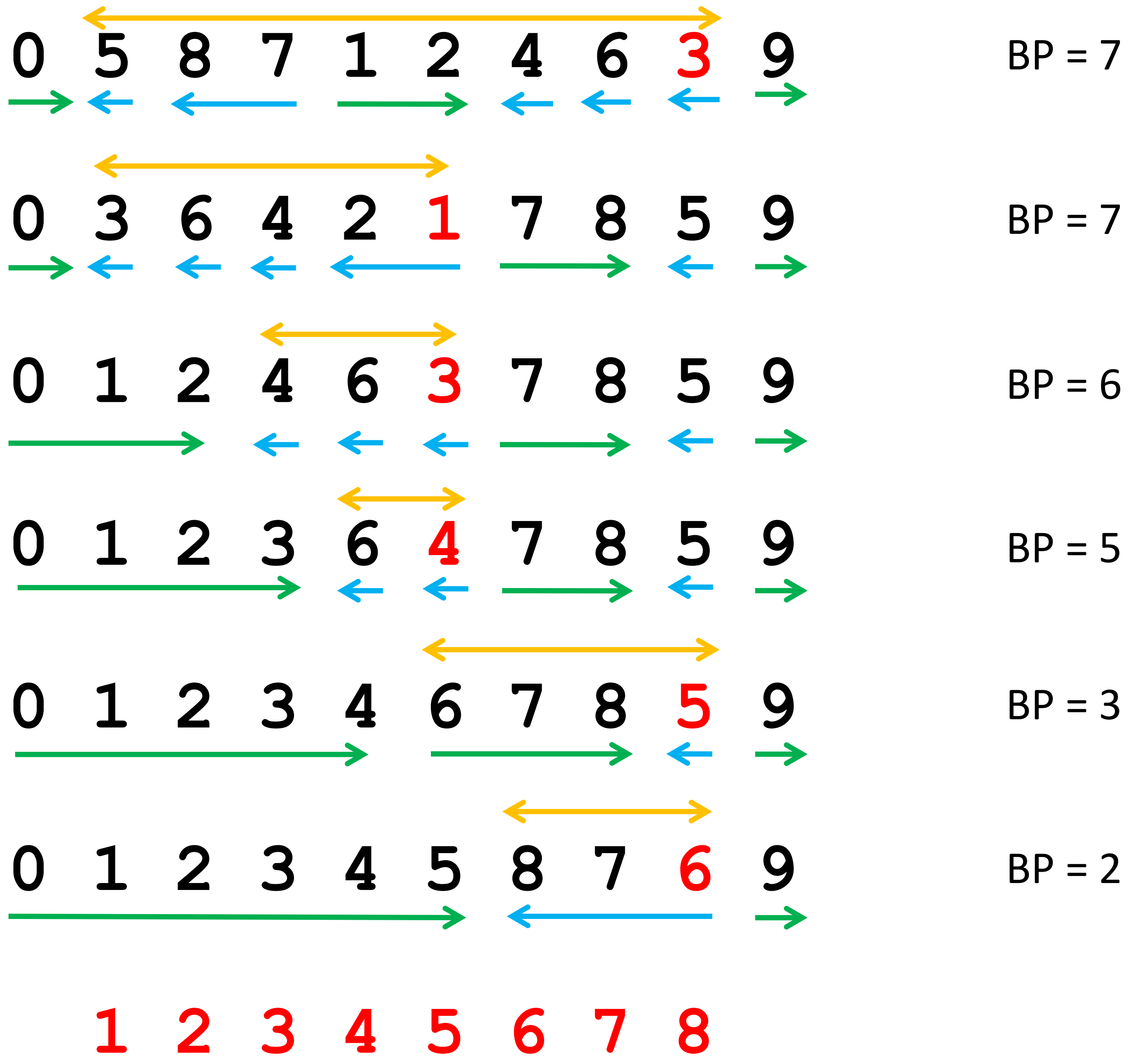


Reversal

- find the descending part with the smallest value at the end
- reverse the region between the first breakpoint and the breakpoint following the selected descending part



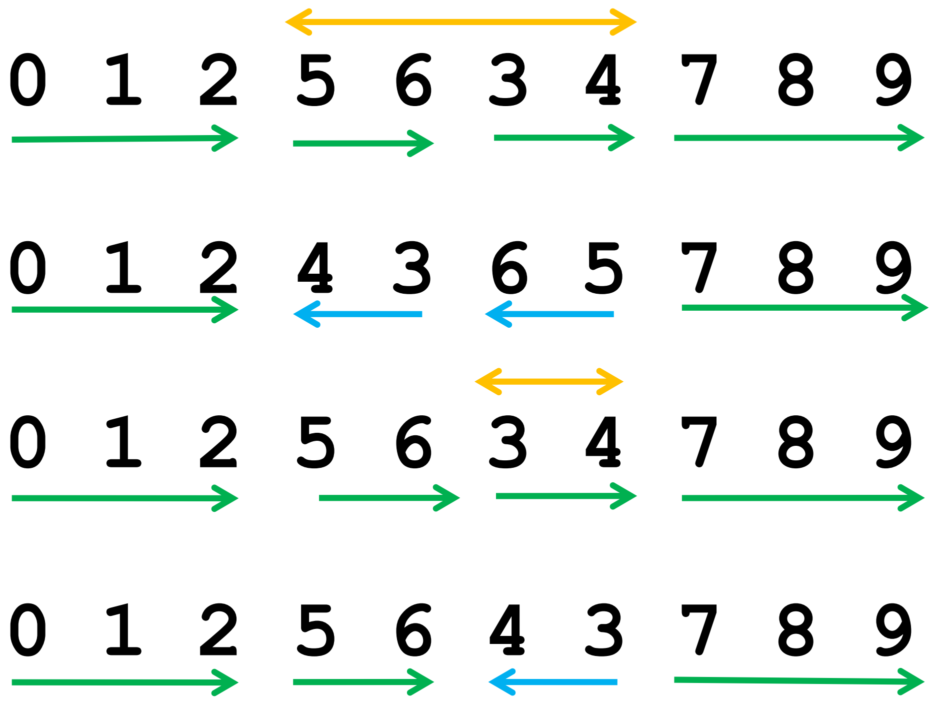
Example



0 5 8 7 1 2 4 6 3 9
BP = 7
0 3 6 4 2 1 7 8 5 9
BP = 7
0 1 2 4 6 3 7 8 5 9
BP = 6
0 1 2 3 6 4 7 8 5 9
BP = 5
0 1 2 3 4 6 7 8 5 9
BP = 3
0 1 2 3 4 5 8 7 6 9
BP = 2
1 2 3 4 5 6 7 8

No Descending Part

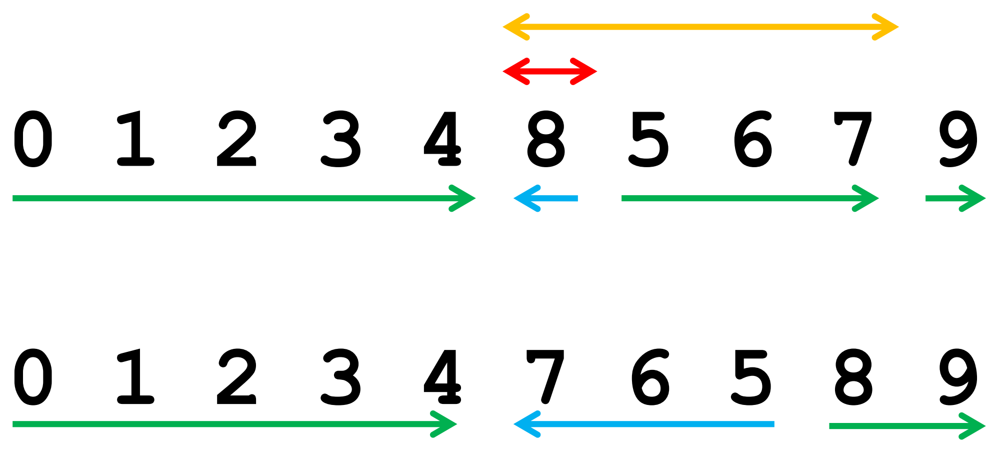
- reverse the entire block between the first and last breakpoint
- or the ascending part that starts with the value "end of sorted part" + 1



or

One Descending Value in front of Sorted Part

- if the descending vector consists of one element and only the sorted part of the vector is in front of it, its reversal does not solve anything
- reverse the entire region between the first and last breakpoint



Example

- sort permutation π = 5 3 2 1 8 7 4 6 using the Breakpoint Sort

Tasks – The Breakpoint Sort

- implement function `FindSorted()`
- implement function `IndicateAscending()`
- implement function `BreakpointSort()`

Task 1

- in R, create a function `FindSorted()` to find an index, at which the unsorted part starts
- input:
  - a vector (permutation) of integers e.g. `0 1 2 3 6 7 4 5 8`
- output:
  - an index, at which the unsorted part starts e.g. `5`

Hint: Compare successively values of the permutation with an increasing number starting at zero (`0`, `1`, `2`, ...) and ending at length of the permutation - 1. The comparison ends when the value in permutation is not the same as the tested value or when the tested value is equal to the length of the permutation - 1.

Task 2

- in R, create a function `IndicateAscending()` to mark ascending and descending parts of a permutation
- input:
  - a vector (permutation) of integers e.g. `0 4 5 3 2 1 6 7 8`
- output:
  - a vector of zeros and ones, where ascending parts are marked by `1` and descending by `0` e.g. `1 1 1 0 0 0 1 1 1`

Hint: Create an indication vector of the same length as the permutation containing only `0` values, and then set the first and last values to `1`. The ascending parts of the permutation vector will be marked with `1` values in the indication vector. Create a loop that iterates through the permutation and if two values next to each other are ascending, i.e. the second is the first + 1, then the indication vector is set to `1` at the given indexes.

Task 3

- in R, create a function `BreakpointSort()` to sort a permutation using Breakpoint Sort
- input:
  - a vector (permutation) of integers e.g. `5 1 4 3 7 8 9 2 6`
- output:
  - sorted permutation of integers e.g. `1 2 3 4 5 6 7 8 9`

Hint: Add marginal values to the permutation and the following steps are repeated in the loop:

- find the start of the unsorted region,
- mark ascending/descending parts,
- find the smallest value that is marked as descending part,
- reversal between the start of unsorted region and the smallest value marked as descending.

The loop ends when the permutation vector is sorted. Watch out for collision situations.