

Greedy Algorithms And Genome Rearrangements

US Change Problem

United States Change Problem:

Convert some amount of money into the fewest number of coins.

Input: An amount of money, M , in cents.

Output: The smallest number of quarters q , dimes d , nickels n , and pennies p whose values add to M (i.e., $25q + 10d + 5n + p = M$ and $q + d + n + p$ is as small as possible).

US Change Problem

```
1  
2  
3  
4  
5  
BETTERCHANGE( $M, \mathbf{c}, d$ )  
1   $r \leftarrow M$   
2  for  $k \leftarrow 1$  to  $d$   
3       $i_k \leftarrow r / c_k$   
4       $r \leftarrow r - c_k \cdot i_k$   
5  return  $(i_1, i_2, \dots, i_d)$ 
```

It is a **greedy algorithm**:

At every step of iteration, a greedy algorithm tries to find the best optimal solution (e.g., used the most the coin with the biggest value)

US Change Problem

```
1  
2 BETTERCHANGE( $M, \mathbf{c}, d$ )  
3   1  $r \leftarrow M$   
4   2 for  $k \leftarrow 1$  to  $d$   
5     3  $i_k \leftarrow r/c_k$   
6     4  $r \leftarrow r - c_k \cdot i_k$   
7   5 return  $(i_1, i_2, \dots, i_d)$ 
```

Does it always find a correct solution?

When $c_1 = 25$, $c_2 = 20$, $c_3 = 10$, $c_4 = 5$, $c_5 = 1$,

if $M = 40$, BetterChange returns $i_1 = 1$, $i_3 = 1$, $i_4 = 1$

We would solve the problem with $i_2 = 2$...

US Change Problem

```
1  BETTERCHANGE( $M, \mathbf{c}, d$ )  
2   $r \leftarrow M$   
3  for  $k \leftarrow 1$  to  $d$   
4       $i_k \leftarrow r / c_k$   
5       $r \leftarrow r - c_k \cdot i_k$   
6  return  $(i_1, i_2, \dots, i_d)$ 
```

But.. how close are we from the optimal solution?

Maybe this algorithm works almost always correctly

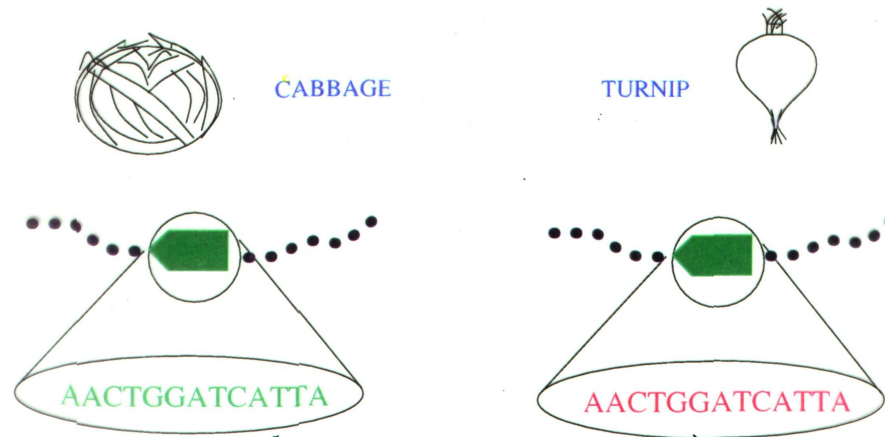
Turnip vs Cabbage: Look and Taste Different

- Although cabbages and turnips share a recent common ancestor, they look and taste different



Turnip vs Cabbage: Comparing Gene Sequences Yields No Evolutionary Information

GENE SEQUENCE COMPARISON



AACTGGATCATT A
AACTGGATCATT A

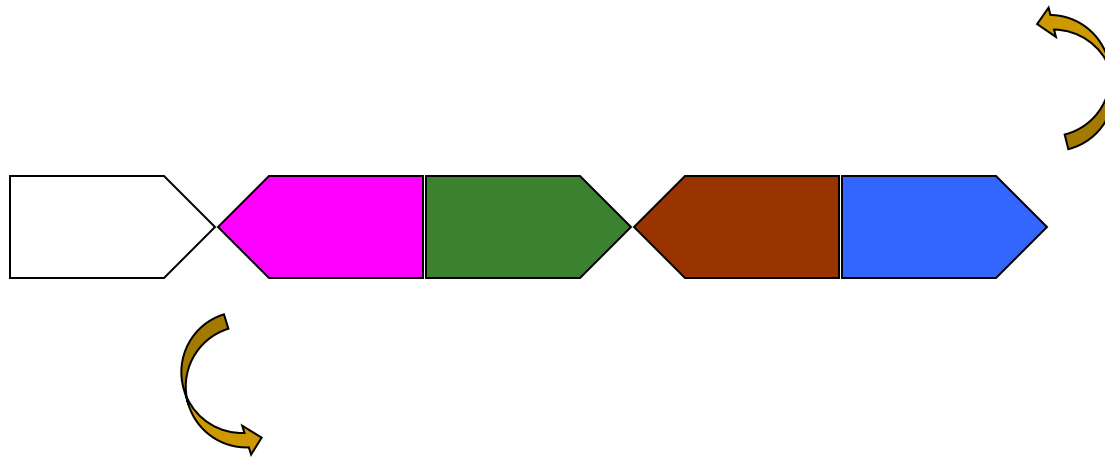
Comparing gene sequences yields
no evolutionary information

Turnip vs Cabbage: Almost Identical mtDNA gene sequences

- In 1980s Jeffrey Palmer studied evolution of plant by comparing genomes of the cabbage and turnip
- 99% similarity between genes
- These surprisingly identical gene sequences differed in gene order
- This study helped pave the way to analyzing genome rearrangements in molecular evolution

Turnip vs Cabbage: Different mtDNA Gene Order

- Gene order comparison:



Turnip vs Cabbage: Different mtDNA Gene Order

- Gene order comparison:



Turnip vs Cabbage: Different mtDNA Gene Order

- Gene order comparison:



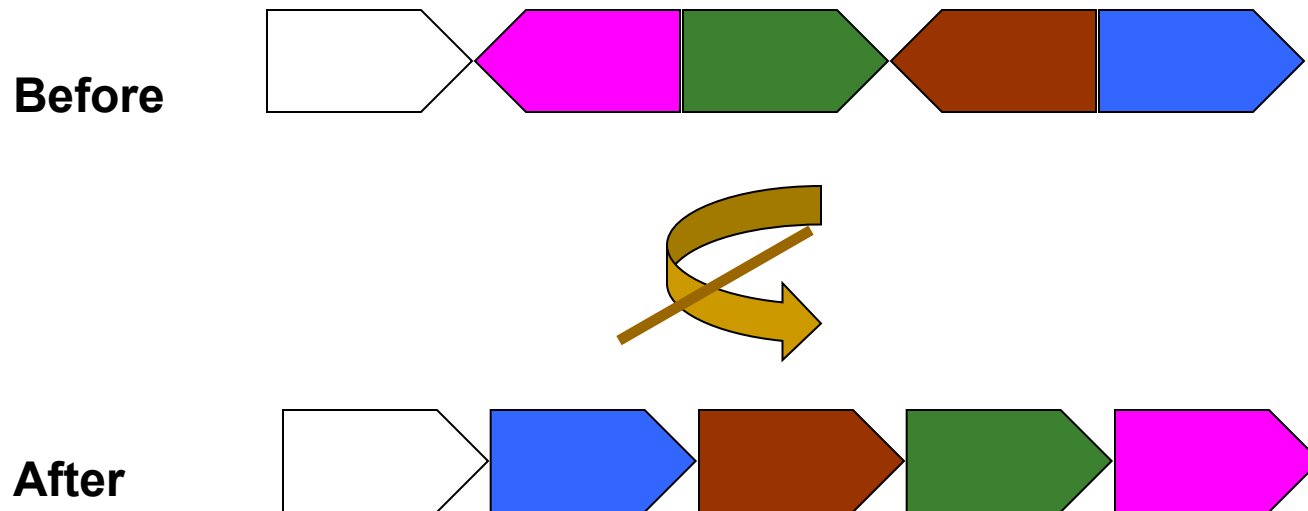
Turnip vs Cabbage: Different mtDNA Gene Order

- Gene order comparison:



Turnip vs Cabbage: Different mtDNA Gene Order

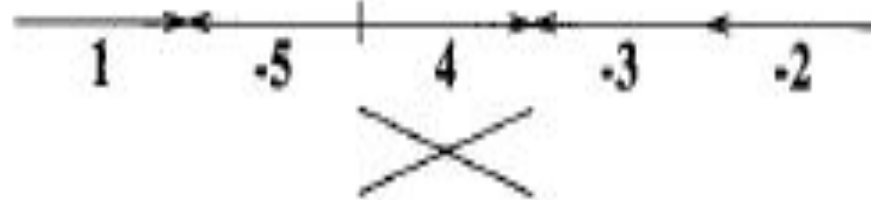
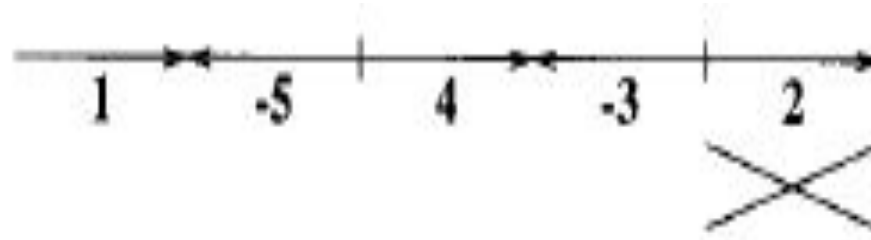
- Gene order comparison:



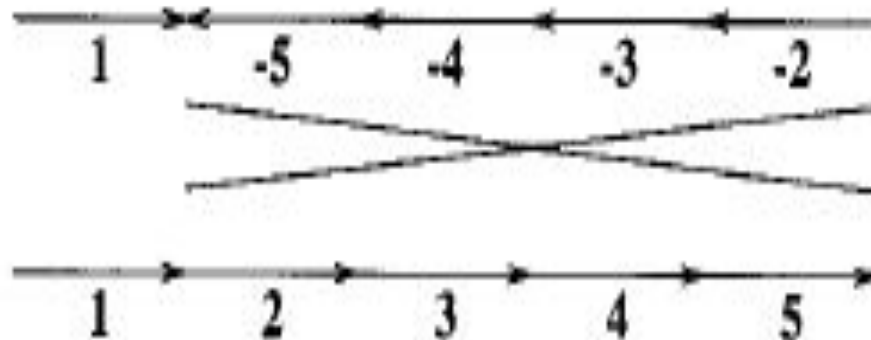
Evolution is manifested as the divergence in gene order

Transforming Cabbage into Turnip

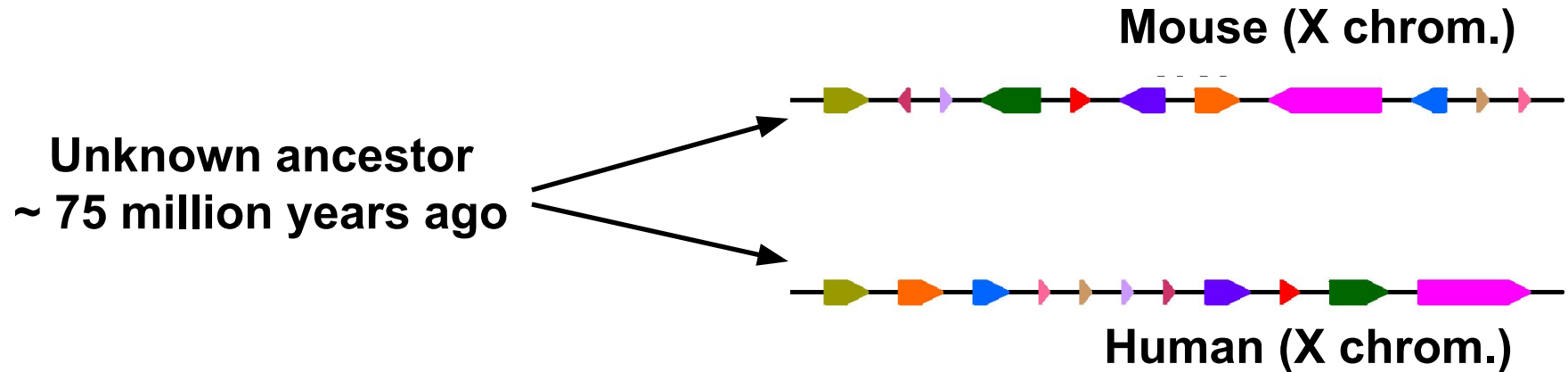
B. oleracea
(cabbage)



B. campestris
(turnip)

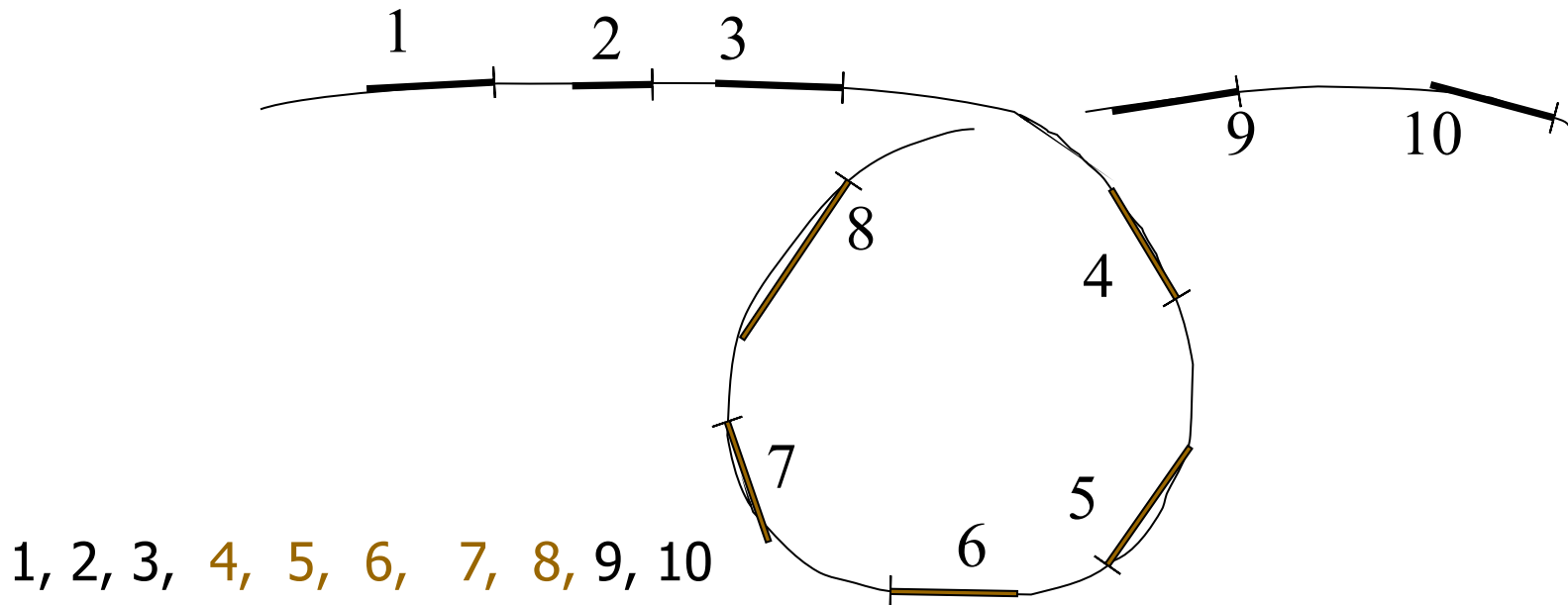


Genome rearrangements



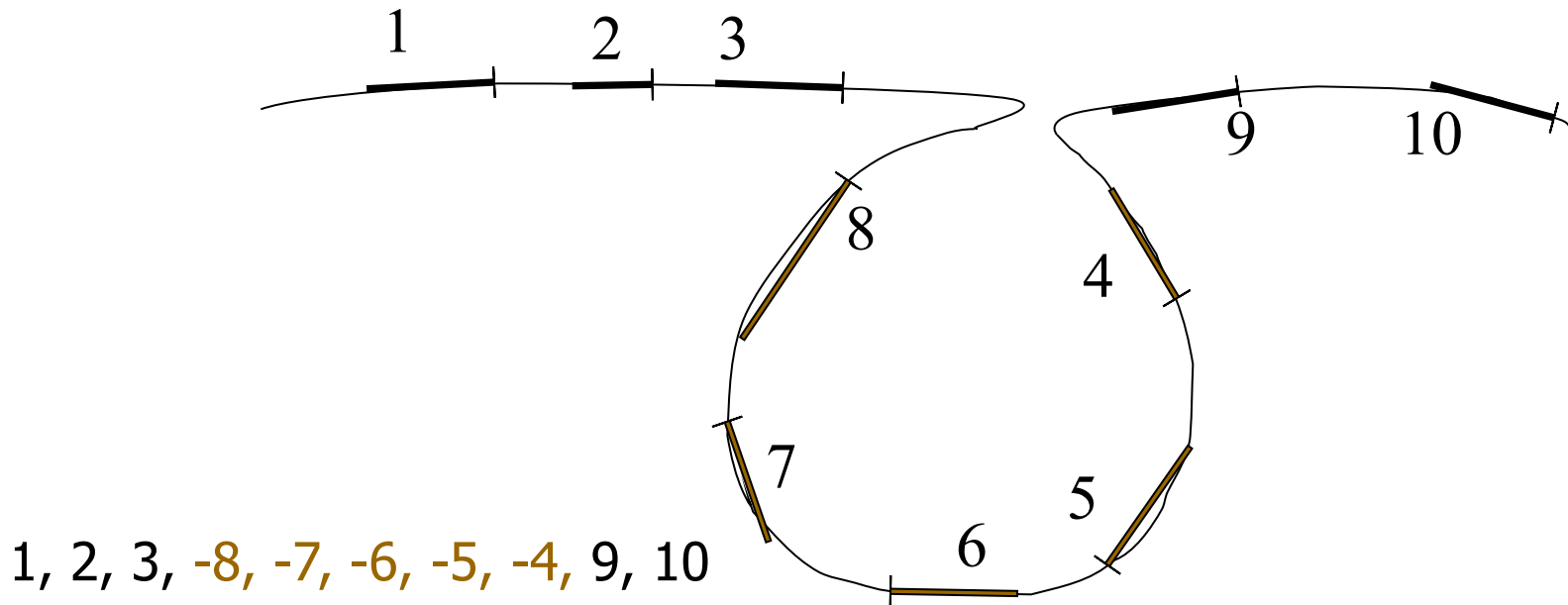
- What are the similarity blocks and how to find them?
- What is the architecture of the ancestral genome?
- What is the evolutionary scenario for transforming one genome into the other?

Reversals



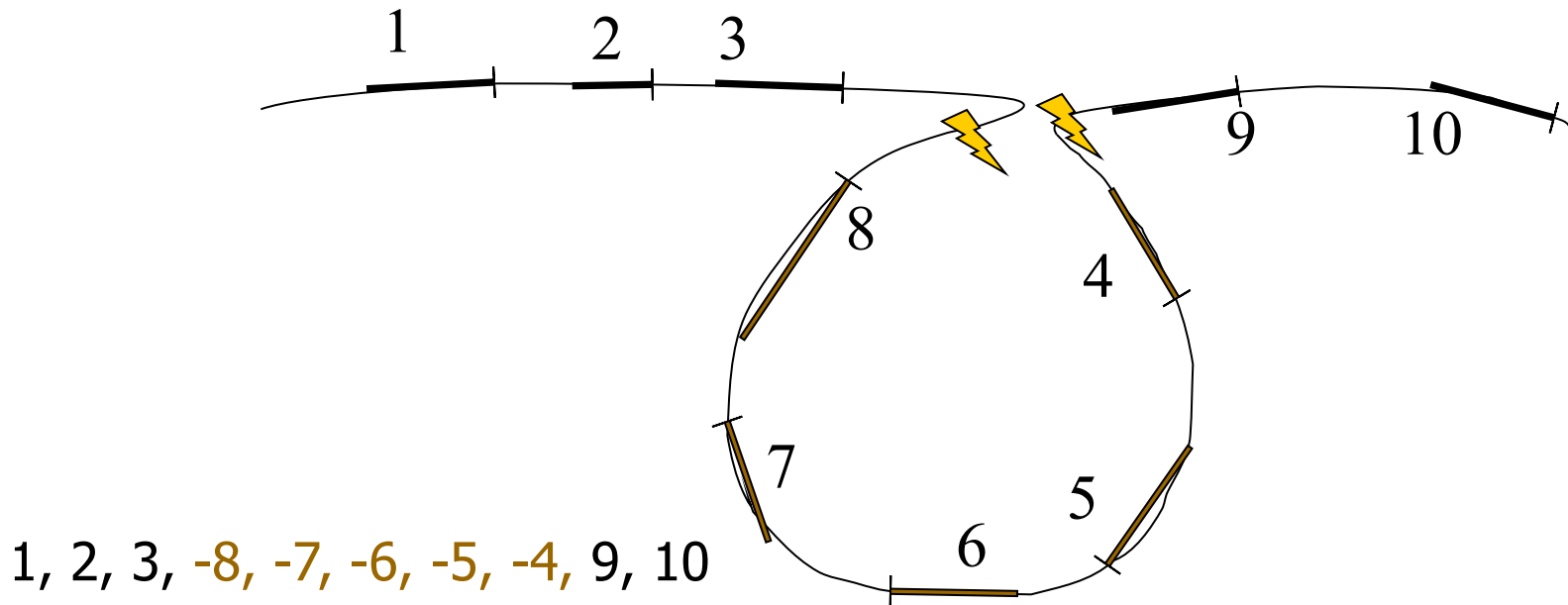
- Blocks represent conserved genes.

Reversals



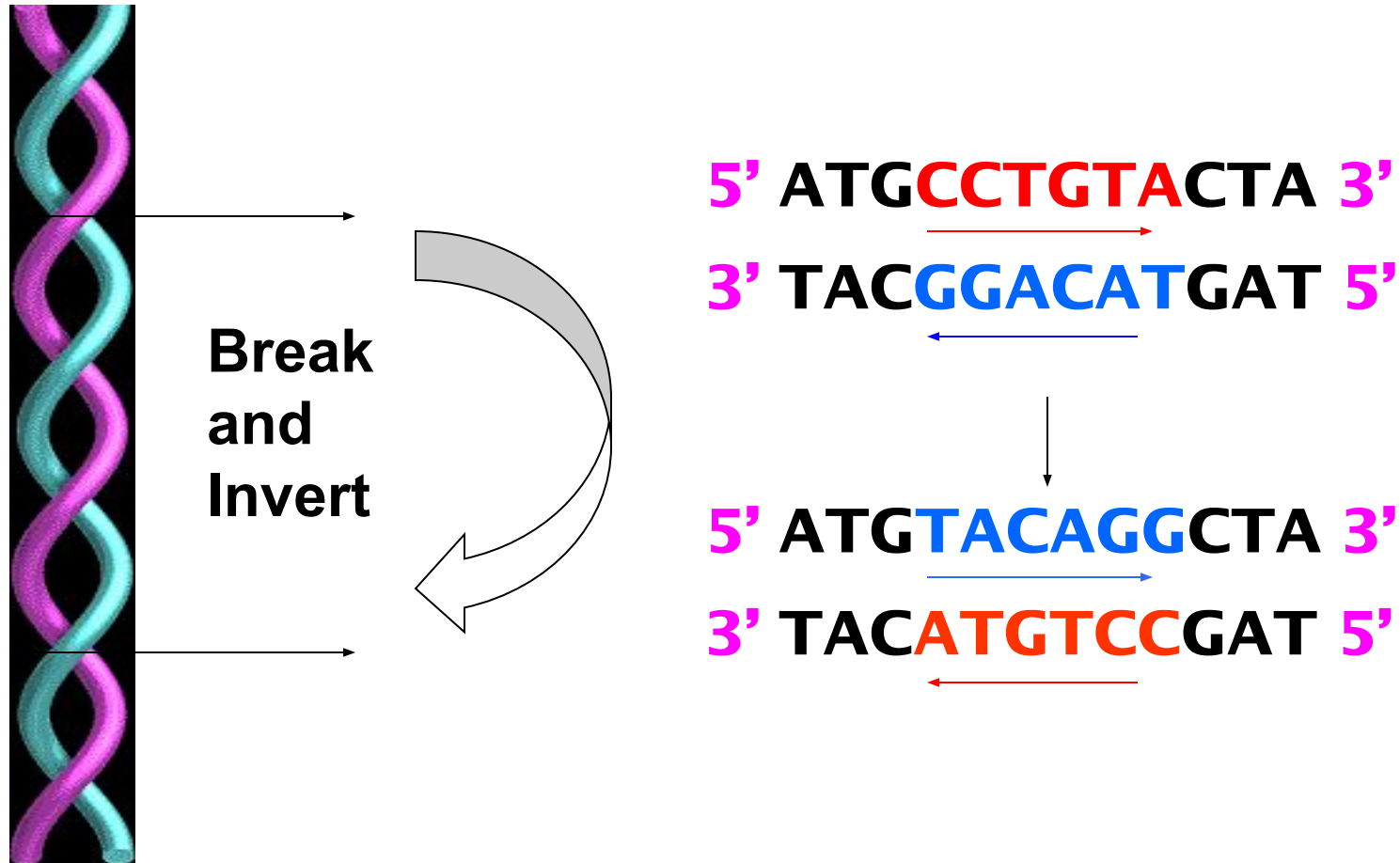
- Blocks represent conserved genes.
- In the course of evolution or in a clinical context, blocks 1,...,10 could be misread as 1, 2, 3, -8, -7, -6, -5, -4, 9, 10.

Reversals and Breakpoints



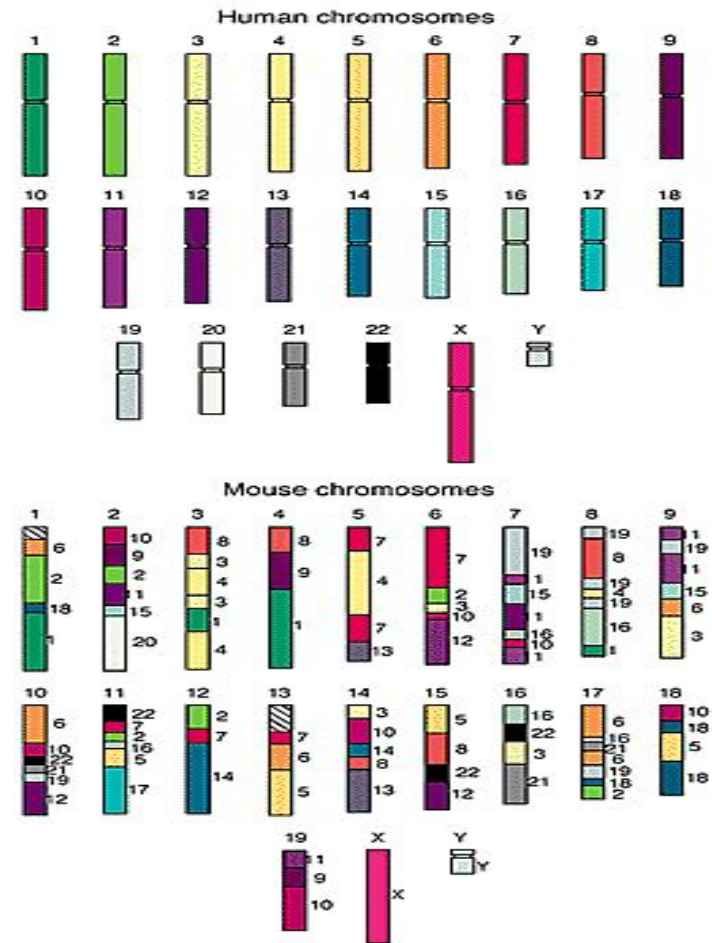
The reversion introduced two *breakpoints* ⚡ (disruptions in order).

Reversals: Example



Comparative Genomic Architectures: Mouse vs Human Genome

- Humans and mice have similar genomes, but their genes are ordered differently
- ~245 rearrangements
 - Reversals
 - Fusions
 - Fissions
 - Translocation



Waardenburg's Syndrome: Mouse Provides Insight into Human Genetic Disorder

- Waardenburg's syndrome is characterized by pigmentary dysphasia
- Gene implicated in the disease was linked to human chromosome 2 but it was not clear where exactly it is located on chromosome 2

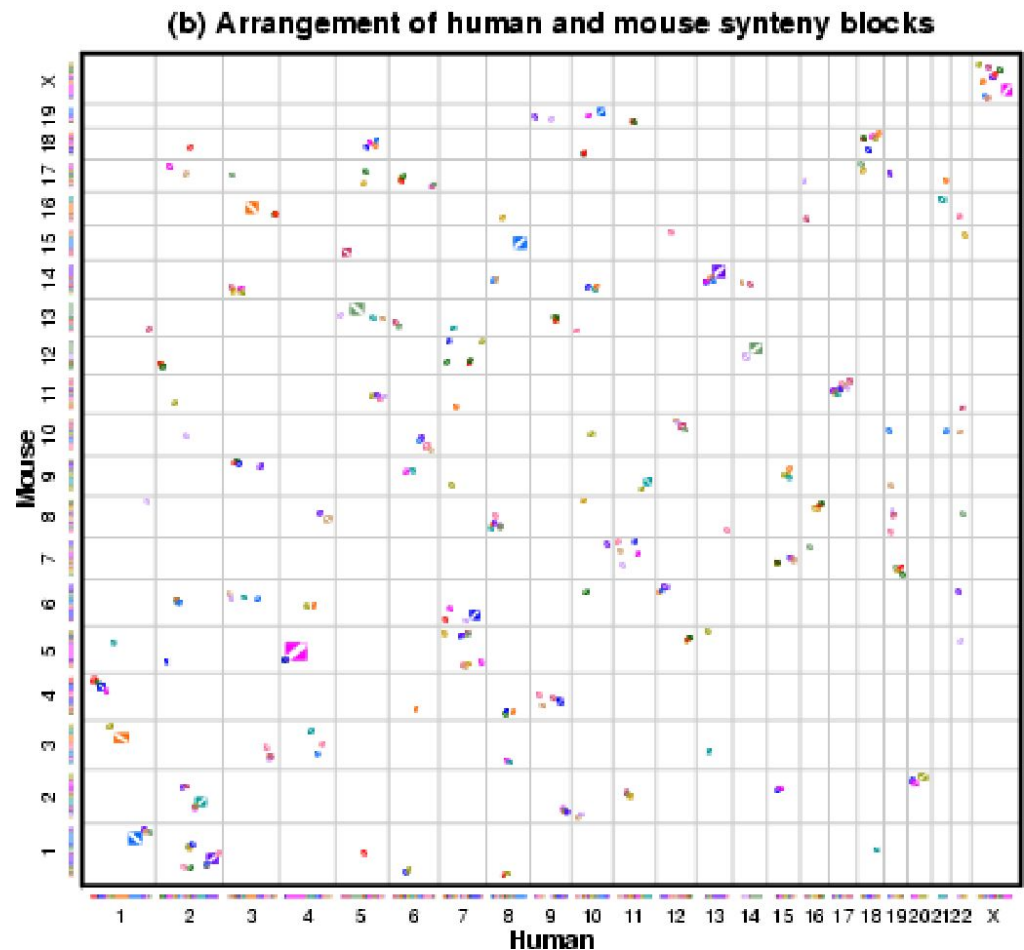


Waardenburg's syndrome and plotch mice

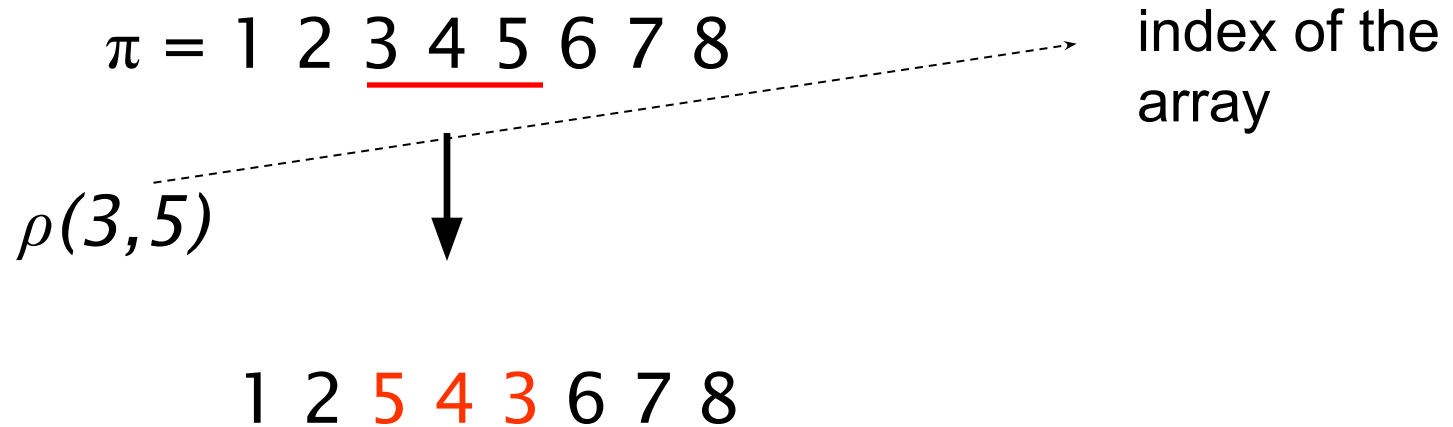
- A breed of mice had similar symptoms caused by the same type of gene as in humans
- Scientists succeeded in identifying location of gene responsible for disorder in mice
- Finding the gene in mice gives clues to where the same gene is located in humans

Comparative Genomic Architecture of Human and Mouse Genomes

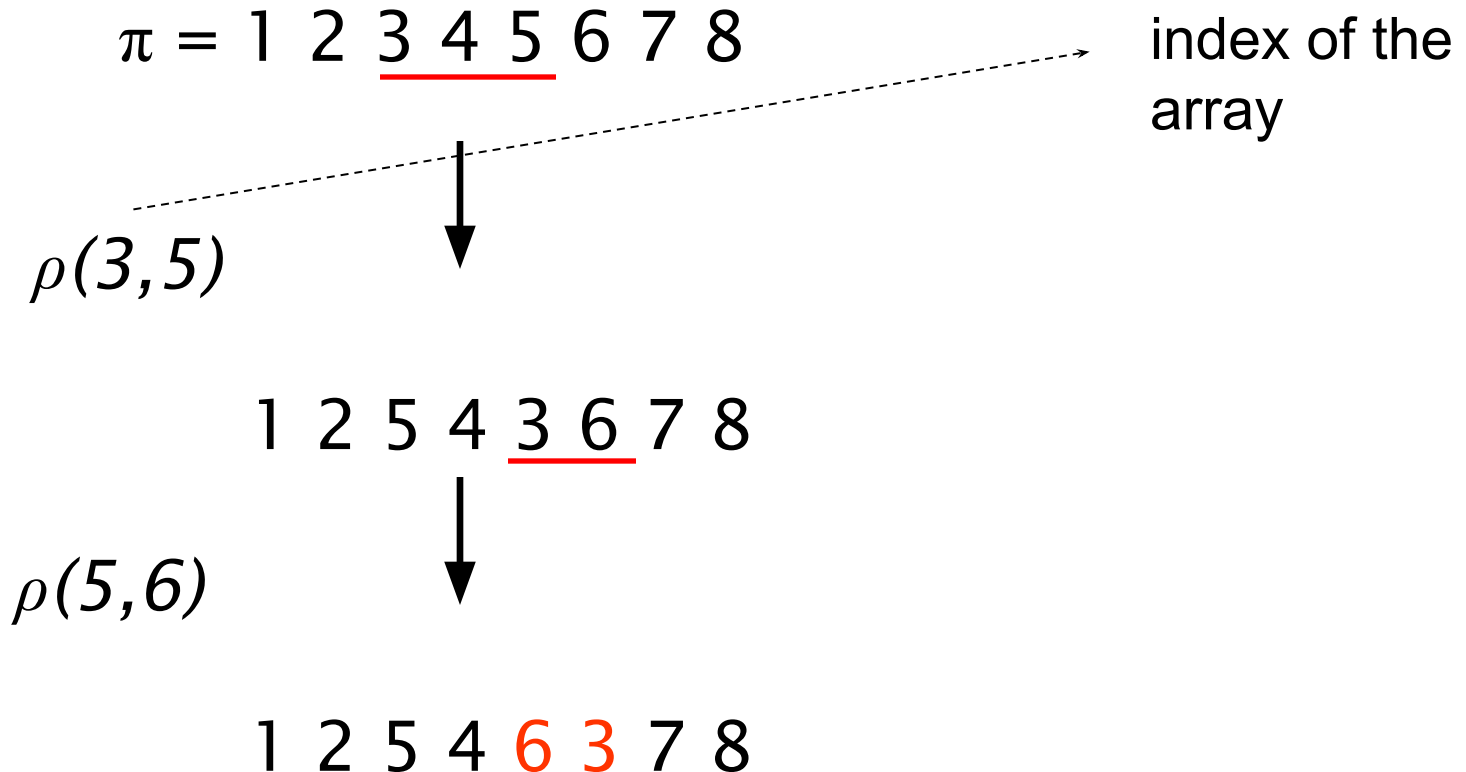
To locate where corresponding gene is in humans, we have to analyze the relative architecture of human and mouse genomes



Reversals: Example



Reversals: Example



Reversals and Gene Orders

- Gene order is represented by a permutation π :

$$\pi = \pi_1 \text{ ----- } \pi_{i-1} \pi_i \pi_{i+1} \text{ ----- } \pi_{j-1} \pi_j \pi_{j+1} \text{ ----- } \pi_n$$

$\rho(i,j)$

$$\pi_1 \text{ ----- } \pi_{i-1} \pi_j \pi_{j-1} \text{ ----- } \pi_{i+1} \pi_i \pi_{j+1} \text{ ----- } \pi_n$$

- Reversal $\rho(i, j)$ reverses (flips) the elements from i to j in π

Reversal Distance Problem

- Goal: Given two permutations, find the shortest series of reversals that transforms one into another
- Input: Permutations π and σ
- Output: A series of reversals ρ_1, \dots, ρ_t transforming π into σ , such that t is minimum
- t - reversal distance between π and σ
- $d(\pi, \sigma)$ - smallest possible value of t , given π and σ

Sorting By Reversals Problem

- Goal: Given a permutation, find a shortest series of reversals that transforms it into the identity permutation $(1\ 2\ \dots\ n)$
- Input: Permutation π
- Output: A series of reversals ρ_1, \dots, ρ_t transforming π into the identity permutation such that t is minimum

Sorting By Reversals: Example

- $t = d(\pi)$ - reversal distance of π
- Example :

$$\pi = \begin{array}{cccccccccc} \underline{3} & \underline{4} & 2 & 1 & 5 & 6 & 7 & 10 & 9 & 8 \\ 4 & 3 & 2 & 1 & 5 & 6 & 7 & \underline{10} & \underline{9} & \underline{8} \\ \underline{4} & \underline{3} & \underline{2} & \underline{1} & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

So $d(\pi) = 3$

Sorting by reversals: 5 steps

Step 0:	π	2	<u>-4</u>	<u>-3</u>	5	-8	-7	-6	1
Step 1:		2	3	4	5	<u>-8</u>	<u>-7</u>	<u>-6</u>	1
Step 2:		2	3	4	5	6	7	8	<u>1</u>
Step 3:		2	3	4	5	6	7	8	-1
Step 4:		<u>-8</u>	<u>-7</u>	<u>-6</u>	<u>-5</u>	<u>-4</u>	<u>-3</u>	<u>-2</u>	<u>-1</u>
Step 5:	γ	1	2	3	4	5	6	7	8

Sorting by reversals: 4 steps

Step 0: π 2 -4 -3 5 -8 -7 -6 1

Step 1: 2 3 4 5 -8 -7 -6 1

Step 2: -5 -4 -3 -2 -8 -7 -6 1

Step 3: -5 -4 -3 -2 -1 6 7 8

Step 4: γ 1 2 3 4 5 6 7 8

Sorting by reversals: 4 steps

Step 0: π 2 -4 -3 5 -8 -7 -6 1

Step 1: 2 3 4 5 -8 -7 -6 1

Step 2: -5 -4 -3 -2 -8 -7 -6 1

Step 3: -5 -4 -3 -2 -1 6 7 8

Step 4: γ 1 2 3 4 5 6 7 8

What is the reversal distance for this permutation? Can it be sorted in 3 steps?

Pancake Flipping Problem

- The chef is sloppy; he prepares an unordered stack of pancakes of different sizes
- The waiter wants to rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom)
- He does it by flipping over several from the top, repeating this as many times as necessary



Christos Papadimitriou and Bill Gates flip pancakes

Pancake Flipping Problem: Formulation

- Goal: Given a stack of n pancakes, what is the minimum number of flips to rearrange them into perfect stack?
- Input: Permutation π
- Output: A series of prefix reversals ρ_1, \dots, ρ_t transforming π into the identity permutation such that t is minimum

Pancake Flipping Problem: Greedy Algorithm

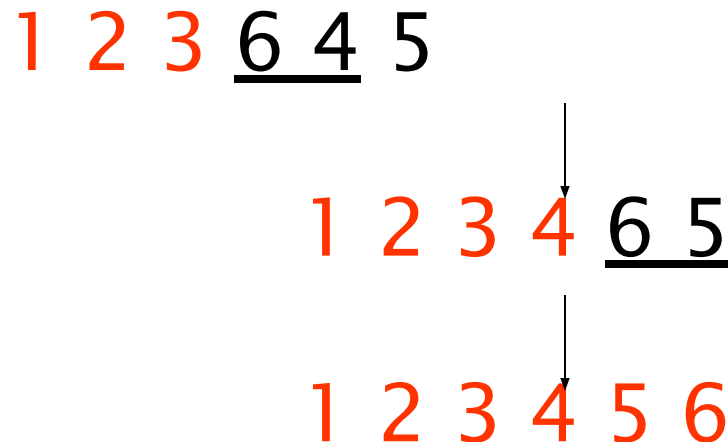
- Greedy approach: 2 prefix reversals at most to place a pancake in its right position, $2n - 2$ steps total at most
- William Gates and Christos Papadimitriou showed in the mid-1970s that this problem can be solved by at most $\frac{5}{3}(n + 1)$ *prefix reversals*

Sorting By Reversals: A Greedy Algorithm

- If sorting permutation $\pi = 1\ 2\ 3\ 6\ 4\ 5$, the first three elements are already in order so it does not make any sense to break them.
- The length of the already sorted prefix of π is denoted $prefix(\pi)$
 - $prefix(\pi) = 3$
- This results in an idea for a greedy algorithm: increase $prefix(\pi)$ at every step

Greedy Algorithm: An Example

- Doing so, π can be sorted



- Number of steps to sort permutation of length n is at most $(n - 1)$

Greedy Algorithm: Pseudocode

SimpleReversalSort(π)

```
1 for  $i \leftarrow 1$  to  $n - 1$ 
2    $j \leftarrow$  position of element  $i$  in  $\pi$  (i.e.,  $\pi_j = i$ )
3   if  $j \neq i$ 
4      $\pi \leftarrow \pi * \rho(i, j)$ 
5   output  $\pi$ 
6 if  $\pi$  is the identity permutation
7   return
```

Analyzing SimpleReversalSort

- SimpleReversalSort does not guarantee the smallest number of reversals and takes five steps on $\pi = 6\ 1\ 2\ 3\ 4\ 5$:

- Step 1: 1 6 2 3 4 5
- Step 2: 1 2 6 3 4 5
- Step 3: 1 2 3 6 4 5
- Step 4: 1 2 3 4 6 5
- Step 5: 1 2 3 4 5 6

Analyzing SimpleReversalSort (cont'd)

- But it can be sorted in two steps:
 $\pi = 6 \ 1 \ 2 \ 3 \ 4 \ 5$
 - Step 1: $5 \ 4 \ 3 \ 2 \ 1 \ 6$
 - Step 2: $1 \ 2 \ 3 \ 4 \ 5 \ 6$
- So, SimpleReversalSort(π) is not optimal
- Optimal algorithms are unknown for many problems; approximation algorithms are used

Approximation Algorithms

- These algorithms find approximate solutions rather than optimal solutions
- The approximation ratio of an algorithm A on input π is:

$$A(\pi) / \text{OPT}(\pi)$$

where

$A(\pi)$ - solution produced by algorithm A
 $\text{OPT}(\pi)$ - optimal solution of the problem

Approximation Ratio/Performance Guarantee

- Approximation ratio (**performance guarantee**) of algorithm A: max approximation ratio of all inputs of size n
 - For algorithm A that minimizes objective function (minimization algorithm):
 - $\max_{|\pi| = n} A(\pi) / \text{OPT}(\pi)$

Approximation Ratio/Performance Guarantee

- Approximation ratio (**performance guarantee**) of algorithm A: max approximation ratio of all inputs of size n
 - For algorithm A that minimizes objective function (minimization algorithm):
 - $\max_{|\pi| = n} A(\pi) / \text{OPT}(\pi)$
 - For maximization algorithm:
 - $\min_{|\pi| = n} A(\pi) / \text{OPT}(\pi)$

Adjacencies and Breakpoints

$$\pi = \pi_1 \pi_2 \pi_3 \dots \pi_{n-1} \pi_n$$

- A pair of elements π_i and π_{i+1} are **adjacent** if

$$\pi_{i+1} = \pi_i \pm 1$$

- For example:

$$\pi = 1 \quad 9 \quad \underline{3} \quad \underline{4} \quad \underline{7} \quad \underline{8} \quad 2 \quad \underline{6} \quad 5$$

- (3, 4) or (7, 8) and (6,5) are adjacent pairs

Breakpoints: An Example

There is a **breakpoint** between any adjacent element that are non-consecutive:

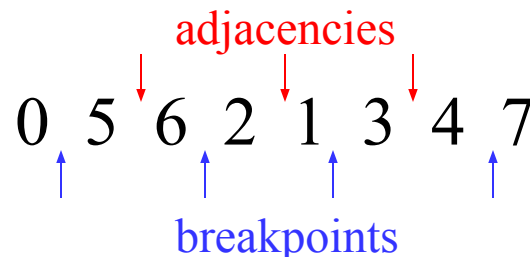
$$\pi = 1 \mid 9 \mid 3 \mid 4 \mid 7 \mid 8 \mid 2 \mid 6 \mid 5$$

- Pairs (1,9), (9,3), (4,7), (8,2) and (2,5) form breakpoints of permutation π
- $b(\pi)$ - # breakpoints in permutation π

Adjacency & Breakpoints

- An **adjacency** - a pair of adjacent elements that are **consecutive**
- A **breakpoint** - a pair of adjacent elements that are **not consecutive**

$\pi = 5 \ 6 \ 2 \ 1 \ 3 \ 4 \longrightarrow$ Extend π with $\pi_0 = 0$ and $\pi_7 = 7$



Extending Permutations

- We put two elements $\pi_0 = 0$ and $\pi_{n+1} = n+1$ at the ends of π

Example:

$$\pi = 1 \mid 9 \mid 3 \mid 4 \mid 7 \mid 8 \mid 2 \mid 6 \mid 5$$



Extending with 0 and 10

$$\pi = 0 \mid 1 \mid 9 \mid 3 \mid 4 \mid 7 \mid 8 \mid 2 \mid 6 \mid 5 \mid 10$$

Note: A new breakpoint was created after extending

Reversal Distance and Breakpoints

- Each reversal eliminates at most 2 breakpoints.

$$\pi = 2 \ 3 \ 1 \ 4 \ 6 \ 5$$

$$0 \mid \underline{2 \ 3} \mid \underline{1} \mid 4 \mid 6 \ 5 \mid 7$$

$$b(\pi) = 5$$

$$0 \ 1 \mid \underline{3 \ 2} \mid 4 \mid 6 \ 5 \mid 7$$

$$b(\pi) = 4$$

$$0 \ 1 \ 2 \ 3 \ 4 \mid \underline{6 \ 5} \mid 7$$

$$b(\pi) = 2$$

$$0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7$$

$$b(\pi) = 0$$

Reversal Distance and Breakpoints

- Each reversal eliminates at most 2 breakpoints.
- This implies:

$$\text{reversal distance} \geq \# \text{breakpoints} / 2$$

$$\pi = 2 \ 3 \ 1 \ 4 \ 6 \ 5$$

$$0 \mid \underline{2 \ 3} \mid 1 \mid 4 \mid 6 \ 5 \mid 7$$

$$b(\pi) = 5$$

$$0 \ 1 \mid \underline{3 \ 2} \mid 4 \mid 6 \ 5 \mid 7$$

$$b(\pi) = 4$$

$$0 \ 1 \ 2 \ 3 \ 4 \mid \underline{6 \ 5} \mid 7$$

$$b(\pi) = 2$$

$$0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7$$

$$b(\pi) = 0$$

Sorting By Reversals: A Better Greedy Algorithm

BreakPointReversalSort(π)

- 1 **while** $b(\pi) > 0$
- 2 Among all possible reversals, choose reversal ρ minimizing $b(\pi \bullet \rho)$
- 3 $\pi \leftarrow \pi \bullet \rho(i, j)$
- 4 **output** π
- 5 **return**

Sorting By Reversals: A Better Greedy Algorithm

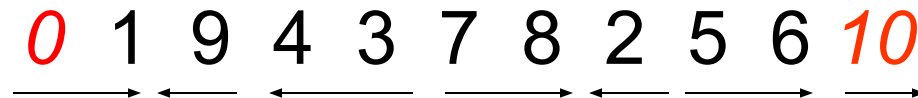
BreakPointReversalSort(π)

- 1 **while** $b(\pi) > 0$
- 2 Among all possible reversals, choose reversal ρ minimizing $b(\pi \bullet \rho)$
- 3 $\pi \leftarrow \pi \bullet \rho(i, j)$
- 4 **output** π
- 5 **return**

Problem: this algorithm may work forever

Strips

- Strip: an interval between two consecutive breakpoints in a permutation
 - Decreasing strip: *strip* of elements in decreasing order (e.g. 6 5 and 3 2).
 - Increasing strip: *strip* of elements in increasing order (e.g. 7 8)



- A single-element strip can be declared either increasing or decreasing. We will choose to declare them as decreasing with exception of the strips with 0 and $n+1$

Reducing the Number of Breakpoints

Theorem 1:

If permutation π contains at least one decreasing strip, then there exists a reversal ρ which decreases the number of breakpoints (i.e. $b(\pi \bullet \rho) < b(\pi)$)

Things To Consider

- For $\pi = 1\ 4\ 6\ 5\ 7\ 8\ 3\ 2$

$0\ 1\ |4|\ 6\ 5\ |7\ 8|\ 3\ 2\ |9\quad b(\pi) = 5$

- Choose decreasing strip with the smallest element k in π ($k = 2$ in this case)

Things To Consider (cont'd)

- For $\pi = 1\ 4\ 6\ 5\ 7\ 8\ 3\ 2$

$0\ 1\ |4|\ 6\ 5\ |7\ 8|\ 3\ 2\ |9\quad b(\pi) = 5$

- Choose decreasing strip with the smallest element k in π ($k = 2$ in this case)

Things To Consider (cont'd)

- For $\pi = 1\ 4\ 6\ 5\ 7\ 8\ 3\ 2$

$0\ 1\ |\ 4\ |\ 6\ 5\ |\ 7\ 8\ |\ 3\ 2\ |\ 9\quad b(\pi) = 5$

- Choose decreasing strip with the smallest element k in π ($k = 2$ in this case)
- Find $k - 1$ in the permutation

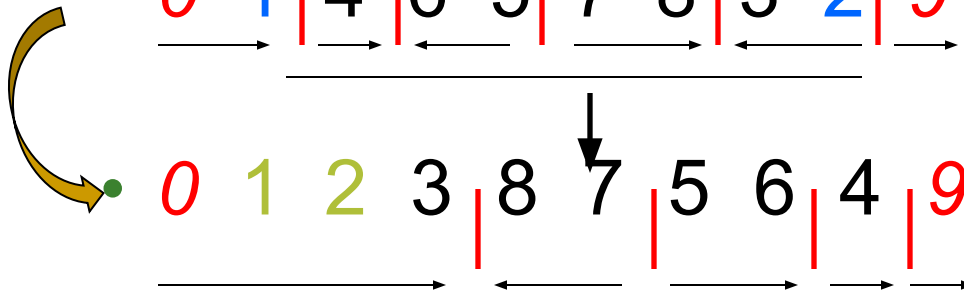
Things To Consider (cont'd)

- For $\pi = 1\ 4\ 6\ 5\ 7\ 8\ 3\ 2$

$0\ 1\ |\ 4\ |\ 6\ 5\ |\ 7\ 8\ |\ 3\ 2\ |\ 9\quad b(\pi) = 5$

- Choose decreasing strip with the smallest element k in π ($k = 2$ in this case)
- Find $k - 1$ in the permutation
- Reverse the segment between k and $k-1$:

$0\ 1\ |\ 4\ |\ 6\ 5\ |\ 7\ 8\ |\ 3\ 2\ |\ 9\quad b(\pi) = 5$



Reducing the Number of Breakpoints Again

- If there is no decreasing strip, there may be no reversal ρ that reduces the number of breakpoints (i.e. $b(\pi \bullet \rho) \geq b(\pi)$ for any reversal ρ).
- By reversing an increasing strip (# of breakpoints stay unchanged), we will create a decreasing strip at the next step. Then the number of breakpoints will be reduced in the next step (theorem 1).

Things To Consider (cont'd)

- There are no decreasing strips in π , for:

$$\begin{array}{l} \pi = \underline{0} \quad \underline{1} \quad \underline{2} \mid \underline{5} \quad \underline{6} \quad \underline{7} \mid \underline{3} \quad \underline{4} \mid \underline{8} \quad b(\pi) = 3 \\ \pi \bullet \rho(6,7) = \underline{0} \quad \underline{1} \quad \underline{2} \mid \underline{5} \quad \underline{6} \quad \underline{7} \mid \underline{4} \quad \underline{3} \mid \underline{8} \quad b(\pi) = 3 \end{array}$$

- ✓ $\rho(6,7)$ does not change the # of breakpoints
- ✓ $\rho(6,7)$ creates a decreasing strip thus guaranteeing that the next step will decrease the # of breakpoints.

ImprovedBreakpointReversalSort

ImprovedBreakpointReversalSort(π)

```
1 while  $b(\pi) > 0$ 
2   if  $\pi$  has a decreasing strip
3   Among all possible reversals, choose reversal  $\rho$ 
      that minimizes  $b(\pi \bullet \rho)$ 
4   else
5     Choose a reversal  $\rho$  that flips an increasing strip in  $\pi$ 
6      $\pi \sqsubseteq \pi \bullet \rho$ 
7   output  $\pi$ 
8 return
```

ImprovedBreakpointReversalSort: Performance Guarantee

- *ImprovedBreakPointReversalSort* is an approximation algorithm with a performance guarantee of at most 4
 - It eliminates at least one breakpoint in every two steps; at most $2b(\pi)$ steps
 - Approximation ratio: $2b(\pi) / d(\pi)$
 - Optimal algorithm eliminates at most 2 breakpoints in every step: $d(\pi) \geq b(\pi) / 2$
 - Performance guarantee:
 - $(2b(\pi) / d(\pi)) \geq [2b(\pi) / (b(\pi) / 2)] = 4$

When should we use Greedy Algorithms?

When should we use Greedy Algorithms?

- **Simple and easy to understand**
- **Fast and efficient** (compared to other techniques)
- **Provides a good enough solution** (we have how good today for one problem)
- **Can be used as a building block for other algorithms:** it can be used as a starting point for developing more complex algorithms.
- **Useful for a variety of problems:** optimization problems, including knapsack problems, scheduling problems, and routing problems