GREEDY SORTING OF PERMUTATIONS BY REVERSALS

A PROJECT REPORT

Submitted by

BL.EN.U4AIE19007	APOORVA MANI
BL.EN.U4AIE19041	M. TANUJ
BL.EN.U4AIE19068	AISHWARYA.V

For the course

19BIO212 – Intelligence of Biological Systems - 4

Guided and Evaluated by

TRIPTI SINGH and AMRITA THAKUR

IN

ARTIFICIAL INTELLIGENCE ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

BANGALORE 560 035

ABSTRACT

Genome rearrangement is a research area capturing wide attention in molecular biology. The reversal distance problem is one of the most widely studied models of genome rearrangements in inferring the evolutionary relationship between two genomes at chromosome level. The problem of estimating reversal distance between two genomes is modeled as sorting by reversals. In our project, the objective is to find the shortest series of reversals that transforms one genome into another genome and we'll optimize the number of reversals. In this, we'll be analyzing the performances of various algorithms such as simple greedy sort by reversals, break-point reversal sort, improved break-point reversal sort for genomic rearrangement.

INTRODUCTION

Gene ordering is changed by a genome rearrangement event and the genomic architecture of a species is altered by a series of genome rearrangements. While the gene level evolution is measured by local mutations such as insertions, substitutions, and deletions, genetic evolution at the chromosome level is based on global rearrangement events. Reversals and translocations are the most common rearrangement events observed in mammalian genetic evolution. A reversal event changes the order of genes acting on the same chromosome while a translocation rearranges the genes swapping segments between two chromosomes.

Genome rearrangement by reversals alone has been considered a worthwhile study to understand evolutionary distance of different species at the chromosome level. Genome rearrangement by reversals provides a good method of studying evolutionary history between two species. However, finding a series of rearrangements that transforms one genome into another is a challenging combinatorial problem. Estimating the reversal distance between two genomes has been modeled as sorting by reversals problem and the problem has been proven to be NP-hard by Caprara in 1997.

SORTING BY REVERSALS

Genome rearrangement studies typically ignore the lengths of synteny blocks and represent chromosomes by signed permutations. Each block is labeled by an integer, which is assigned a positive/negative sign depending on the block’s direction. The ordering of blocks is represented by an ordering of these integers, with the endpoints of the chromosome represented by parentheses. The number of elements in a signed permutation is its length.

Mouse: (+1 -7 +6 -10 +9 -8 +2 -11 -3 +5 +4)

Human: (+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11)

Fig 1: human and mouse X chromosomes represented as permutations of length 11

We can model reversals by inverting the elements within an interval of a signed permutation, then switching the signs of any elements within the inverted

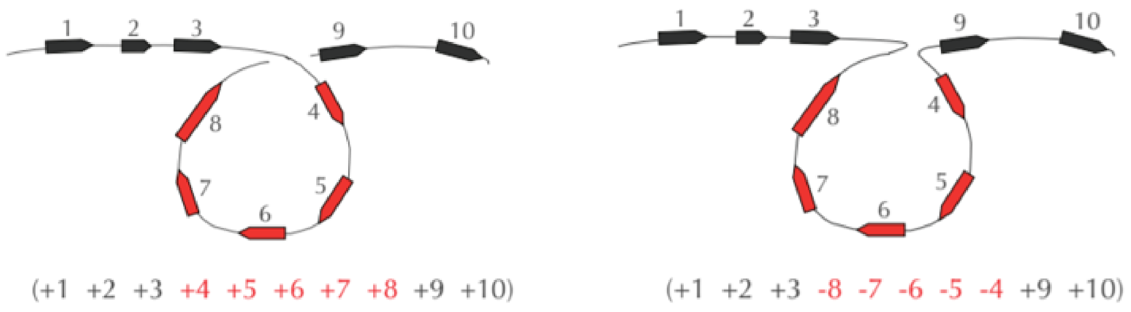


Fig2: A cartoon illustrating how a reversal breaks a chromosome in two places and inverts the segment between the two breakpoints.

"Reversal Distance" Problem:

Calculate the reversal distance between two signed permutations.

Input: Two signed permutations of equal length.

Output: The series of reversals (reversal distance) between these signed permutations.

We represented the human X chromosome by (+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11); this signed permutation, in which blocks are ordered from smallest to largest with positive directions, is called the identity signed permutation. The reason why we used the identity signed permutation of length 11 to represent the human X chromosome is that when comparing two genomes, we can label the synteny blocks in one of the genomes however we like. The block labeling for which the human X chromosome is the identity signed permutation automatically induces the representation of the mouse chromosome as (+1 7 +6 10 +9 8 +2 11 3 +5 +4). Of course, we could have instead encoded the mouse X chromosome as the identity signed permutation, which would have induced the encoding of the human X chromosome as (+1 +7 9 +11 +10 +3 2 6 +5 4 8) (Fig3).

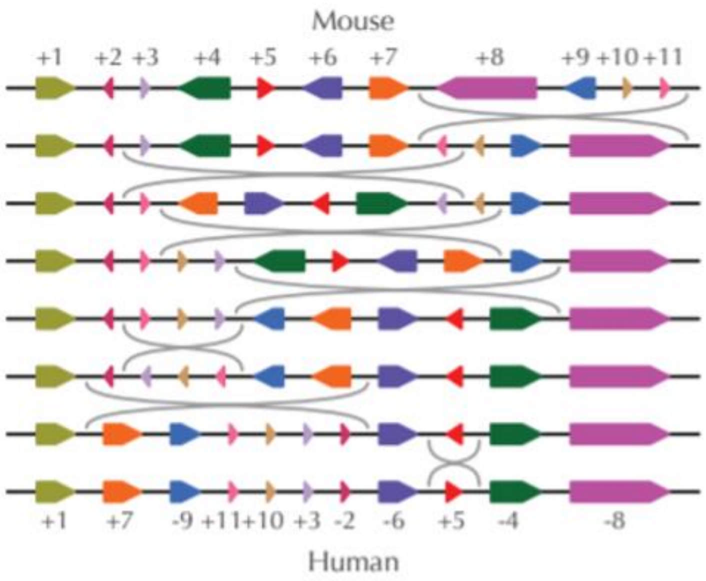


Fig3: Encoding the mouse X chromosome as the identity signed permutation implies encoding the human X chromosome as (+1 +7 9 +11 +10 +3 2 6 +5 4 8).

A simplified restatement of the reversal distance problem…

"Sorting by Reversals" Problem:

Compute the reversal distance between a signed permutation and the identity signed permutation.

Input: A signed permutation P.

Output: The reversal distance $d_{\mathrm{rev}}(P)$.

Example:

$$\begin{array}{lll}
\Pi_0 & = 2, \underline{4, 3}, 5, 8, 7, 6, 1 & \rho(2,3) \\
\Pi_1 & = 2, 3, 4, 5, \underline{8, 7, 6}, 1 & \rho(5,7) \\
\Pi_2 & = \underline{2, 3, 4, 5, 6, 7, 8}, 1 & \rho(1,7) \\
\Pi_3 & = \underline{8, 7, 6, 5, 4, 3, 2, 1} & \rho(1,8) \\
\Pi_4 & = 1, 2, 3, 4, 5, 6, 7, 8 &
\end{array}$$

$$d(\Pi) = 4$$

Here, the number of reversals is 3. But, is 4 the minimum number of reversals? Can it be done in 3?

Greedy Algorithm for Sorting by Reversals

When sorting the permutation, $\Pi = 1, 2, 3, 6, 4, 5$, one notices that the first three elements are already in order. So, it does not make sense to break them apart. The length of the already sorted prefix of Π is denoted a $prefix(\Pi) = 3$

This inspires the following simple greedy algorithm

```
while $prefix(\Pi) < len(\Pi)$:
    perform a reversal $\rho(prefix(\Pi) + 1, k)$ such that $prefix(\Pi)$ increases by
at least 1.
```

Such a reversal must always exist. Finding, k , is as simple as finding the index of the minimum value of the remaining unsorted part.
The number of steps to sort any permutation of length is at most (n -1).

Algorithm:

```
GREEDYSORTING($P$)
    $approxReversalDistance \leftarrow 0$
    for $k \leftarrow 1$ to $|P|$
        if element $k$ is not sorted
            apply the $k$-sorting reversal to $P$
            $approxReversalDistance \leftarrow approxReversalDistance + 1$
            if $k$-th element of $P$ is $-k$
                apply the reversal flipping the $k$-th element of $P$
                $approxReversalDistance \leftarrow approxReversalDistance + 1$
    return $approxReversalDistance$
```

Example:

$$\begin{aligned}
\Pi_1 &: 6, 1, 2, 3, 4, 5 \\
\rho(1, 2) &: 1, 6, 2, 3, 4, 5 \\
\rho(2, 3) &: 1, 2, 6, 3, 4, 5 \\
\rho(3, 4) &: 1, 2, 3, 6, 4, 5 \\
\rho(4, 5) &: 1, 2, 3, 4, 6, 5 \\
\rho(5, 6) &: 1, 2, 3, 4, 5, 6
\end{aligned}$$

But there is a solution with far fewer flips. The same sequence sorted with two reversals.

$$\begin{aligned}\Pi &: 6,1,2,3,4,5\\ \rho(1,6) &: 5,4,3,2,1,6\\ \rho(1,5) &: 1,2,3,4,5,6\end{aligned}$$

This isn't greedy method but yet it beats a greedy approach handily. So GreedyReversalSort(π) is correct (as a sorting algorithm), but non-optimal.

New Concept: Adjacencies & Breakpoints

The consecutive elements (pi pi+1) in signed permutation P = (p1 ... pn) form an adjacency if pi+1 pi is equal to 1. By definition, for any positive element k < n, both (k k + 1) and ((k + 1) k) are adjacencies. If pi+1 pi is not equal to 1, then we say that (pi pi+1) is a breakpoint.

Breakpoints occur between neighbouring non-adjacent elements

$$\Pi = 1, \mid 9, \mid \underline{3,4,} \mid \underline{7,8,} \mid 2, \mid \underline{6,5}$$

- There are 5 breakpoints in our permutation between pairs (1,9), (9,3), (4,7), (8,2) and (2,6).
- There are 3 adjacencies in our permutation between pairs (3,4), (7,8), (6,5).

We define b(π) as the number of breakpoints in permutation(π).

Extending Permutations

One can place two elements, $\pi_0 = 0$ and $\pi_{n+1} = n+1$ at the beginning and end of π respectively.

$$\begin{gathered}1, \mid 9, \mid \underline{3,4,} \mid \underline{7,8,} \mid 2, \mid \underline{6,5}\\ \downarrow\\ \Pi = 0\, 1, \mid 9, \mid \underline{3,4,} \mid \underline{7,8,} \mid 2, \mid \underline{6,5,} \mid 10\end{gathered}$$

- An additional breakpoint was created after extending
- An extended permutation of length *n* can have at most breakpoints
- (*n*-1) between the original elements plus 2 for the extending elements

- Breakpoints are the targets for sorting by reversals.
- Once they are removed, the permutation is sorted.
- Each "useful" reversal eliminates at least 1, and at most 2
- breakpoints.
- Consider the following application of GreedyReversalSort(Extend(π))

$$
\begin{array}{lll}
\Pi = & 2, 3, 1, 4, 6, 5 & \\
 & 0 \mid 2, 3 \mid 1 \mid 4 \mid 6, 5 \mid 7 & b(\Pi) = 5 \\
 & 0, \overline{1 \mid 3, 2} \mid 4 \mid 6, 5 \mid 7 & b(\Pi) = 4 \\
 & 0, 1, \overline{2, 3}, 4 \mid 6, 5 \mid 7 & b(\Pi) = 2 \\
 & 0, 1, 2, 3, 4, \overline{5, 6}, 7 & b(\Pi) = 0
\end{array}
$$

Sorting By Reversals: A second Greedy Algorithm

```
BreakpointReversalSort(π):

1. while $b(\pi) > 0$:
2.     Among all possible reversals, choose reversal ρ
   minimizing $b(\pi)$
3.     $\Pi \leftarrow \Pi \cdot \rho(i, j)$
4.     output $\Pi$
5. return
```

The greedy concept here is to reduce as many breakpoints as possible at each step. Does it always terminate? What if no reversal reduces the number of breakpoints?

New Concept: Strips

Strip is 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 extension strips (with 0 and n+1)

$$\overrightarrow{0, 1}, \overleftarrow{9}, \overleftarrow{4, 3}, \overrightarrow{7, 8}, \overleftarrow{2}, \overrightarrow{5, 6}, \overrightarrow{10}$$

1) Choose the decreasing strip with the smallest element k in π.
    - It'll always be the rightmost element of that strip
2) Find k-1 in the permutation
    - It'll always be flanked by a breakpoint
3) Reverse the segment between k and k-1

If permutation p contains a least one decreasing strip, then there exists a reversal r which decreases the number of breakpoints (i.e. *b(p*r)<b(p)*).

Example 1:

Consider $\Pi = 1, 4, 6, 5, 7, 8, 3, 2$

$$\overrightarrow{0, 1}, \mid \overleftarrow{4}, \mid \overleftarrow{6, 5}, \mid \overrightarrow{7, 8}, \mid \overleftarrow{3, 2}, \mid \overrightarrow{9} \qquad b(p) = 5$$

$$\overrightarrow{0, 1, 2, 3}, \mid \overleftarrow{8, 7}, \mid \overrightarrow{5, 6}, \mid \overleftarrow{4}, \mid \overrightarrow{9} \qquad b(p) = 4$$

$$\overrightarrow{0, 1, 2, 3, 4}, \mid \overleftarrow{6, 5}, \mid \overrightarrow{7, 8, 9} \qquad b(p) = 2$$

$$\overrightarrow{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} \qquad b(p) = 0$$

$$d(\Pi) = 3$$

Example 2:

$$\overrightarrow{0, 1, 2}, \mid \overrightarrow{5, 6, 7}, \mid \overrightarrow{3, 4}, \mid \overrightarrow{8, 9} \qquad b(p) = 3$$

- If there is no decreasing strip, there may be no strip-reversal that reduces the number of $b(\Pi \cdot \rho(i,j)) \geq b(\Pi)$ breakpoints (i.e. for any reversal).
- However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.
- Then the number of breakpoints will be reduced in the following steps.

$\overrightarrow{0,1,2,}\mid\underleftarrow{\overrightarrow{5,6,7,}}\mid\overrightarrow{3,4,}\mid\overrightarrow{8,9}$	$b(p)=3$	$\rho(3,5)$
$\overrightarrow{0,1,2,}\mid\overleftarrow{7,6,5,}\mid\overrightarrow{3,4,}\mid\overrightarrow{8,9}$	$b(p)=3$	$\rho(6,7)$
$\overrightarrow{0,1,2,}\mid\overleftarrow{7,6,5,4,3,}\mid\overrightarrow{8,9}$	$b(p)=2$	$\rho(3,7)$
$\overrightarrow{0,1,2,3,4,5,6,7,8,9}$	$b(p)=0$	$Done!$

1. With each reversal, one can remove at most 2 breakpoints.
2. If there is any decreasing strip there exists a reversal that will remove at least one breakpoint.
3. If breakpoints remain and there is no decreasing strip one can be created by reserving any remaining strip.

An optimal algorithm would remove 2 breakpoints at every step. The last reversal always removes 2 breakpoints, thus if the number of breakpoints is odd, even the optimal algorithm must make at least one reversal that removes only 1 breakpoint.

Algorithm:

ImprovedBreakpointReversalSort(π)

```
while b(π) > 0
    if π has a decreasing strip
        Among all possible reversals, choose reversal ρ that minimizes b(π • ρ)
    else
        Choose a reversal ρ that flips an increasing strip in π
    π ← π • ρ
output π
return
```

COMPARING GREEDY ALGORITHMS

Simple Reversal Sort

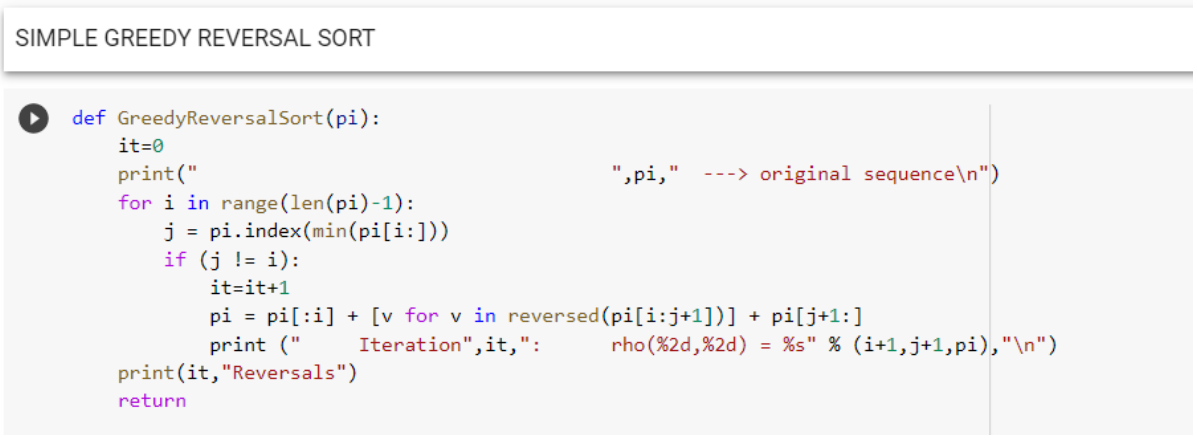
- Attempts to extend the prefix(π) at each step
- Approximation ratio $\frac{n-1}{b(\Pi)/2}$ steps
-

Improved Breakpoint Reversal Sort

- Attempts to reduce the number of breakpoints at each step
- Approximation ratio $\frac{b(\Pi)}{b(\Pi)/2} = 2$ steps

CODE

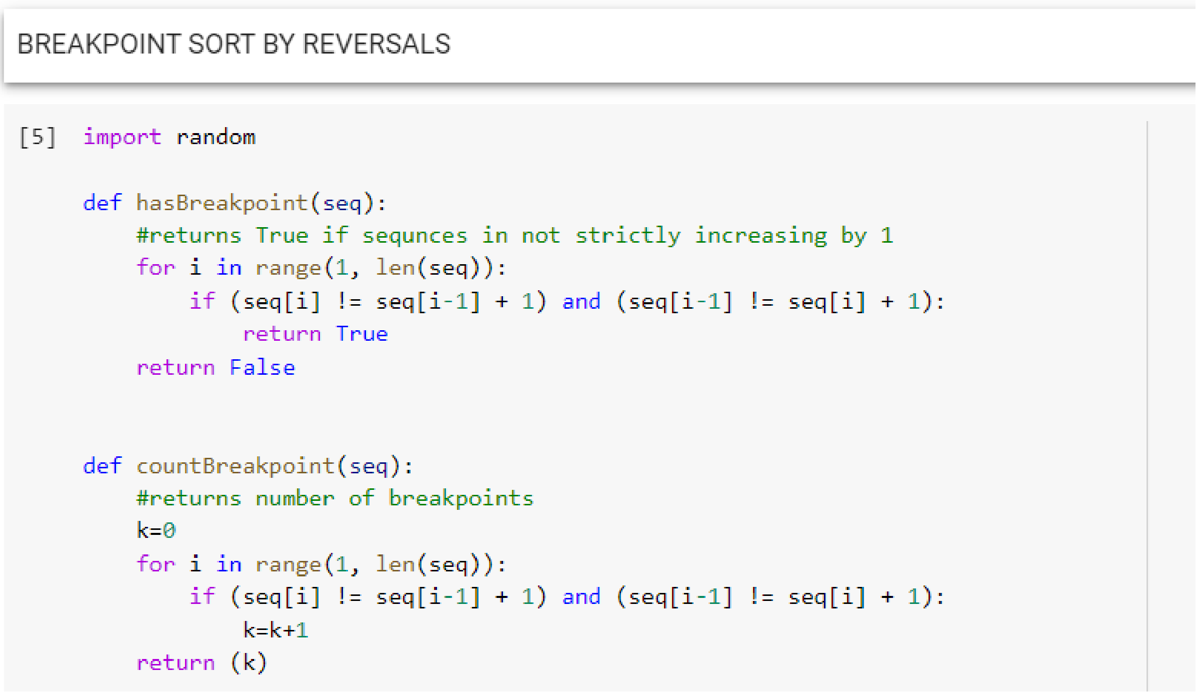
Simple Greedy Reversal Sort:



SIMPLE GREEDY REVERSAL SORT

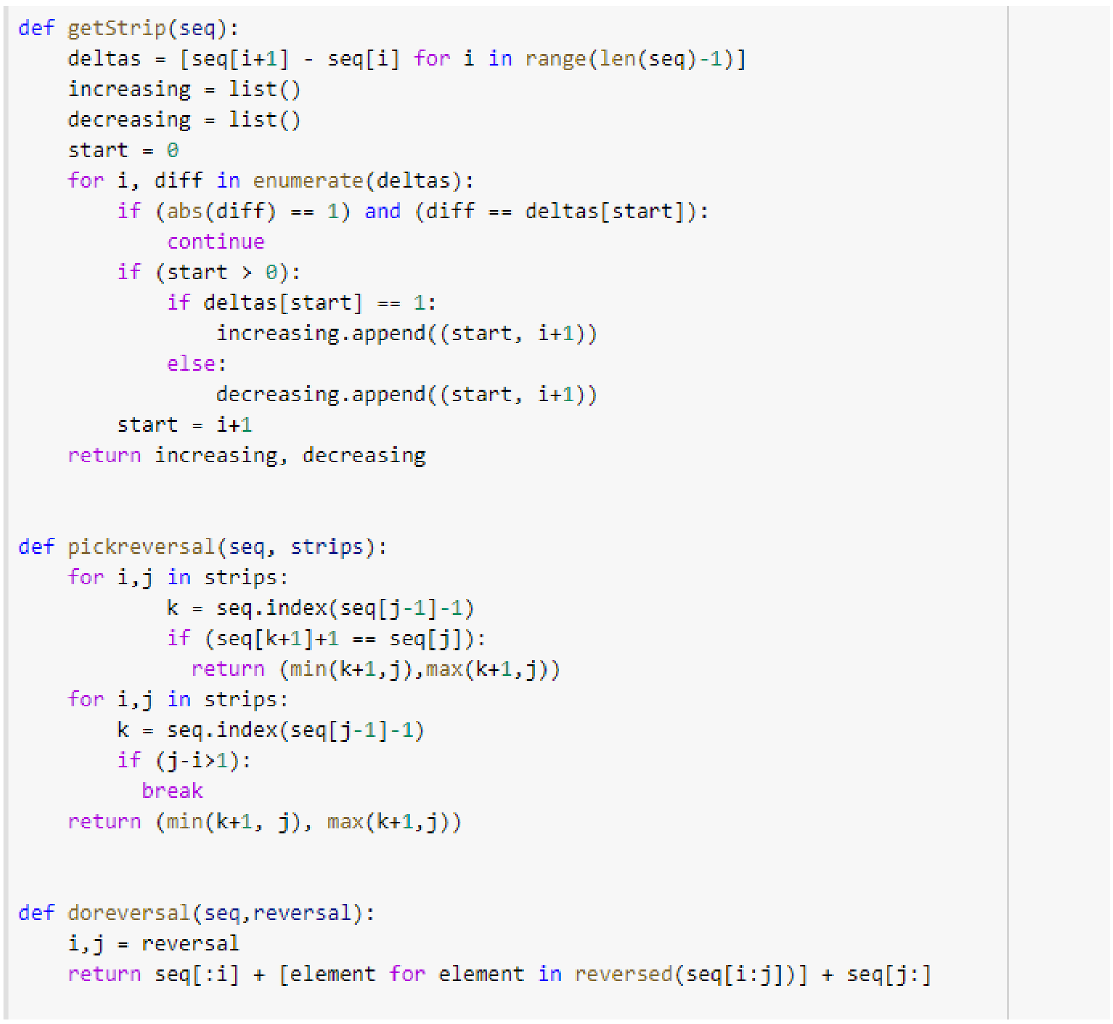
```python
def GreedyReversalSort(pi):
    it=0
    print("                                  ",pi,"  ---> original sequence\n")
    for i in range(len(pi)-1):
        j = pi.index(min(pi[i:]))
        if (j != i):
            it=it+1
            pi = pi[:i] + [v for v in reversed(pi[i:j+1])] + pi[j+1:]
            print ("     Iteration",it,":      rho(%2d,%2d) = %s" % (i+1,j+1,pi),"\n")
    print(it,"Reversals")
    return
```

Breakpoint Sort by Reversals:

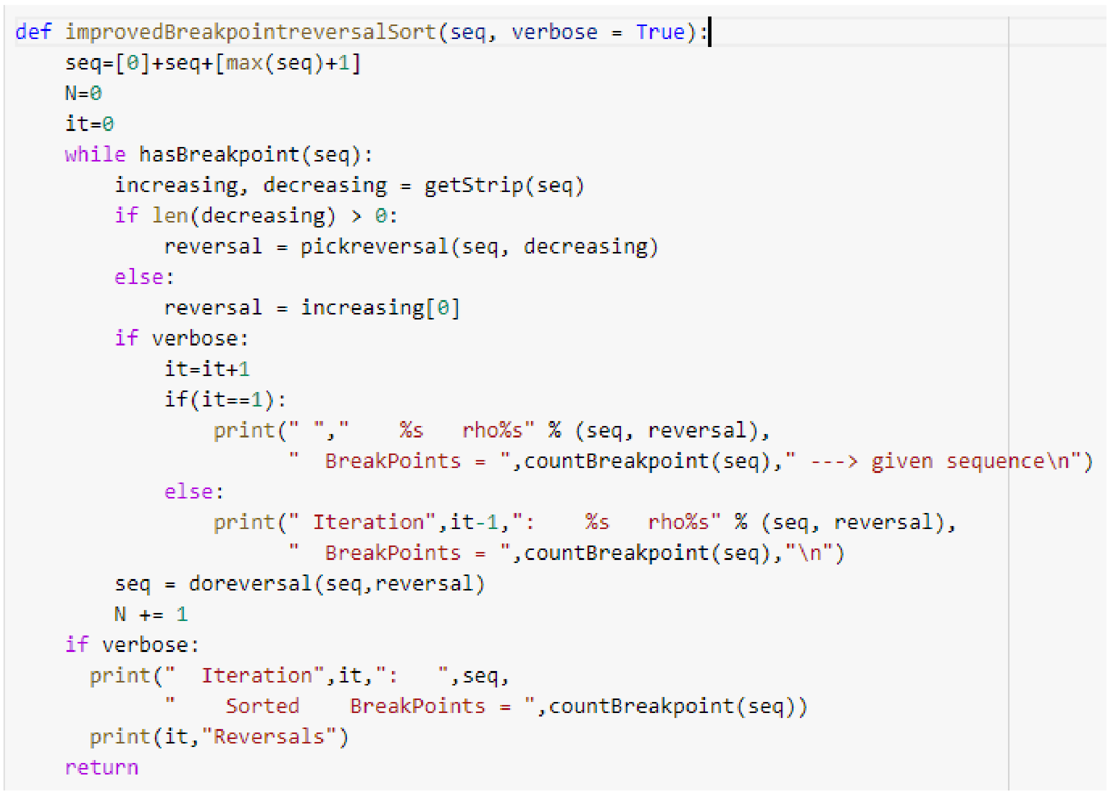


BREAKPOINT SORT BY REVERSALS

```python
[5]  import random

     def hasBreakpoint(seq):
         #returns True if sequnces in not strictly increasing by 1
         for i in range(1, len(seq)):
             if (seq[i] != seq[i-1] + 1) and (seq[i-1] != seq[i] + 1):
                 return True
         return False


     def countBreakpoint(seq):
         #returns number of breakpoints
         k=0
         for i in range(1, len(seq)):
             if (seq[i] != seq[i-1] + 1) and (seq[i-1] != seq[i] + 1):
                 k=k+1
         return (k)
```



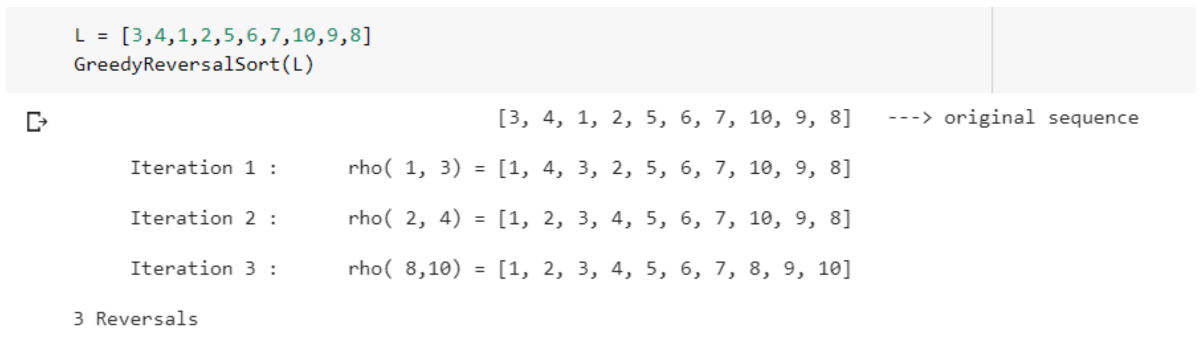
```python
def getStrip(seq):
    deltas = [seq[i+1] - seq[i] for i in range(len(seq)-1)]
    increasing = list()
    decreasing = list()
    start = 0
    for i, diff in enumerate(deltas):
        if (abs(diff) == 1) and (diff == deltas[start]):
            continue
        if (start > 0):
            if deltas[start] == 1:
                increasing.append((start, i+1))
            else:
                decreasing.append((start, i+1))
        start = i+1
    return increasing, decreasing


def pickreversal(seq, strips):
    for i,j in strips:
            k = seq.index(seq[j-1]-1)
            if (seq[k+1]+1 == seq[j]):
              return (min(k+1,j),max(k+1,j))
    for i,j in strips:
        k = seq.index(seq[j-1]-1)
        if (j-i>1):
          break
    return (min(k+1, j), max(k+1,j))


def doreversal(seq,reversal):
    i,j = reversal
    return seq[:i] + [element for element in reversed(seq[i:j])] + seq[j:]
```



```python
def improvedBreakpointreversalSort(seq, verbose = True):
    seq=[0]+seq+[max(seq)+1]
    N=0
    it=0
    while hasBreakpoint(seq):
        increasing, decreasing = getStrip(seq)
        if len(decreasing) > 0:
            reversal = pickreversal(seq, decreasing)
        else:
            reversal = increasing[0]
        if verbose:
            it=it+1
            if(it==1):
                print(" ","    %s    rho%s" % (seq, reversal),
                      "  BreakPoints = ",countBreakpoint(seq)," ---> given sequence\n")
            else:
                print(" Iteration",it-1,":    %s    rho%s" % (seq, reversal),
                      "  BreakPoints = ",countBreakpoint(seq),"\n")
        seq = doreversal(seq,reversal)
        N += 1
    if verbose:
      print("  Iteration",it,":   ",seq,
            "    Sorted    BreakPoints = ",countBreakpoint(seq))
      print(it,"Reversals")
    return
```

RESULTS

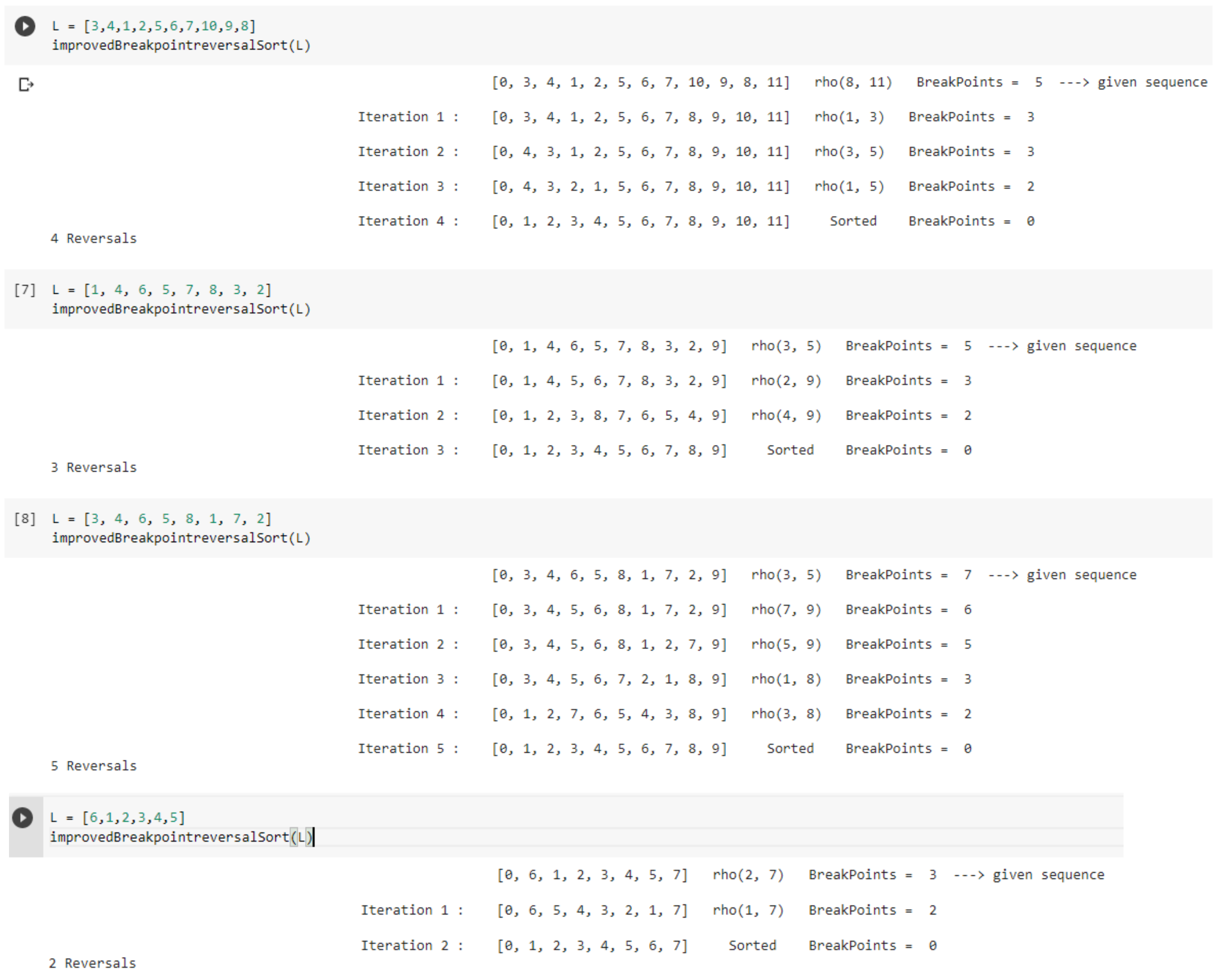
Simple Greedy Reversal Sort:



```
L = [3,4,1,2,5,6,7,10,9,8]
GreedyReversalSort(L)
```

```
                                   [3, 4, 1, 2, 5, 6, 7, 10, 9, 8]   ---> original sequence

        Iteration 1 :      rho( 1, 3) = [1, 4, 3, 2, 5, 6, 7, 10, 9, 8]

        Iteration 2 :      rho( 2, 4) = [1, 2, 3, 4, 5, 6, 7, 10, 9, 8]

        Iteration 3 :      rho( 8,10) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

   3 Reversals
```

Breakpoint Sort by Reversals:



```
L = [3,4,1,2,5,6,7,10,9,8]
improvedBreakpointreversalSort(L)
```

```
                             [0, 3, 4, 1, 2, 5, 6, 7, 10, 9, 8, 11]   rho(8, 11)   BreakPoints =  5  ---> given sequence

              Iteration 1 :  [0, 3, 4, 1, 2, 5, 6, 7, 8, 9, 10, 11]   rho(1, 3)   BreakPoints =  3

              Iteration 2 :  [0, 4, 3, 1, 2, 5, 6, 7, 8, 9, 10, 11]   rho(3, 5)   BreakPoints =  3

              Iteration 3 :  [0, 4, 3, 2, 1, 5, 6, 7, 8, 9, 10, 11]   rho(1, 5)   BreakPoints =  2

              Iteration 4 :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]    Sorted     BreakPoints =  0
4 Reversals
```

```
[7]  L = [1, 4, 6, 5, 7, 8, 3, 2]
     improvedBreakpointreversalSort(L)
```

```
                             [0, 1, 4, 6, 5, 7, 8, 3, 2, 9]   rho(3, 5)   BreakPoints =  5  ---> given sequence

              Iteration 1 :  [0, 1, 4, 5, 6, 7, 8, 3, 2, 9]   rho(2, 9)   BreakPoints =  3

              Iteration 2 :  [0, 1, 2, 3, 8, 7, 6, 5, 4, 9]   rho(4, 9)   BreakPoints =  2

              Iteration 3 :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]    Sorted     BreakPoints =  0
3 Reversals
```

```
[8]  L = [3, 4, 6, 5, 8, 1, 7, 2]
     improvedBreakpointreversalSort(L)
```

```
                             [0, 3, 4, 6, 5, 8, 1, 7, 2, 9]   rho(3, 5)   BreakPoints =  7  ---> given sequence

              Iteration 1 :  [0, 3, 4, 5, 6, 8, 1, 7, 2, 9]   rho(7, 9)   BreakPoints =  6

              Iteration 2 :  [0, 3, 4, 5, 6, 8, 1, 2, 7, 9]   rho(5, 9)   BreakPoints =  5

              Iteration 3 :  [0, 3, 4, 5, 6, 7, 2, 1, 8, 9]   rho(1, 8)   BreakPoints =  3

              Iteration 4 :  [0, 1, 2, 7, 6, 5, 4, 3, 8, 9]   rho(3, 8)   BreakPoints =  2

              Iteration 5 :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]    Sorted     BreakPoints =  0
5 Reversals
```

```
L = [6,1,2,3,4,5]
improvedBreakpointreversalSort(L)
```

```
                             [0, 6, 1, 2, 3, 4, 5, 7]   rho(2, 7)   BreakPoints =  3  ---> given sequence

              Iteration 1 :  [0, 6, 5, 4, 3, 2, 1, 7]   rho(1, 7)   BreakPoints =  2

              Iteration 2 :  [0, 1, 2, 3, 4, 5, 6, 7]    Sorted     BreakPoints =  0
2 Reversals
```