Algorithms for the Swapping Puzzle

Let's dive into algorithms for solving swapping puzzles, such as the one on the course webpage.

Some questions to consider:

- What algorithm do you use?
- What are the best and worst-case runtimes for your algorithm with `n` pieces?
- Is the runtime related to the number of pieces that start in the correct position?
- Is the "greedy algorithm" (placing one piece correctly in each step) the best possible, or can we sometimes place two pieces by not placing one immediately?
- Can we precisely describe all best-possible algorithms?

Puzzle Model

We model the pieces in a puzzle as numbers $1, 2, 3, \ldots, n$, numbered according to the position the piece should occupy.

For example:

```
5 10 14 3 1 11 9 15 8 7 2 12 4 13 6 16  ->  1 2 3 4 5 6 7 8 9 10 11 12 13 14 1
```

The board can also be modeled as an array/list where the gray numbers are indices, starting with 1:

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
 5 10 14  3  1 11  9 15  8  7  2 12  4 13  6 16
```

A setup of the numbers $1, 2, 3, \ldots, n$ in an array of length n is also called a **permutation**.

The Greedy Algorithm and Its Analysis

Greedy Algorithm:

```
WHILE not all pieces in place:
    Choose a piece not in place
    Swap it with the piece in its place
```

For a puzzle with `n` pieces, what is the maximum number of swaps?

- For each swap, at least one piece comes into place, and no piece leaves its place. Therefore, at most `n` swaps are needed.

For a puzzle with `n` pieces, where `t` pieces are already in place, what are the maximum and minimum number of swaps?

- Maximum swaps: $n - t$
- Minimum swaps: $(n - t)/2$

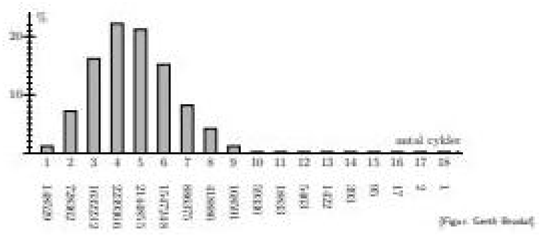
Cycles and Swaps

A permutation naturally leads to a collection of cycles. A piece (number) `t` points to the position where it should be, which is the position with index `t`.

For example:

```
9 3 8 7 1 2 12 5 10 6 4 11
```

The image below illustrates a puzzle to help conceptualize cycles and swaps.



Observation:

- Swapping two pieces in the same cycle increases the number of cycles by exactly one.
- Swapping two pieces in different cycles decreases the number of cycles by exactly one.

Lemma 5:

> The algorithm requires exactly $n - k$ swaps to solve a puzzle with n pieces and k cycles initially.

Observation:

> Piece in place <=> piece is in a cycle of length one.

Proof:

Since the algorithm always swaps two pieces from the same cycle (y comes immediately after x in the cycle), each swap increases the number of cycles by one (Lemma 4). Since we start with k cycles and end with n cycles (Lemma 3), we perform exactly $n - k$ swaps.

Theorem 1:

> To solve a puzzle with n pieces and k cycles initially, exactly $n - k$ swaps are required.

Conclusion:

A puzzle with n pieces and k cycles in the initial setup requires at least $n - k$ swaps, and it can always be done with $n - k$ swaps (e.g., via the greedy algorithm, which in each step divides a cycle of length t into two cycles of length $t - 1$ and 1).

Proof:

That $n - k$ swaps is sufficient (an upper bound) follows from Lemma 5. That it cannot be done with fewer swaps (a lower bound) follows because no swap can increase the number of cycles by more than one (Lemma 4), and one must start and end with the number of cycles.

Conclusion:

An algorithm uses the optimal number of swaps ($n - k$) if and only if each swap is with two pieces that are in the same cycle.

Expected Number of Cycles

According to Theorem 1, the number of cycles in the permutation is crucial.

Theorem (without proof here):

> The expected number of cycles in a random permutation is $H_n = \sum_{i=1}^{n} \frac{1}{i}$.

H_n is the n-th harmonic number, and if the permutations are randomly chosen so that all permutations are equally likely, the following theorem applies:

> The expected optimal number of swaps for a random puzzle is $n - \sum_{i=1}^{n} \frac{1}{i}$.

The n-th harmonic number can be approximated by:

$H_n \approx \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \frac{1}{252n^6} + \dots$