5 *Greedy Algorithms*

The algorithm USCHANGE in chapter 2 is an example of a greedy strategy: at each step, the cashier would only consider the largest denomination smaller than (or equal to) M. Since the goal was to minimize the number of coins returned to the customer, this seemed like a sensible strategy: you would never use five nickels in place of one quarter. A generalization of USCHANGE, BETTERCHANGE also used what seemed like the best option and did not consider any others, which is what makes an algorithm "greedy." Unfortunately, BETTERCHANGE actually returned incorrect results in some cases because of its short-sighted notion of "good." This is a common characteristic of greedy algorithms: they often return suboptimal results, but take very little time to do so. However, there are a lucky few greedy algorithms that find optimal rather than suboptimal solutions.

5.1 Genome Rearrangements

Waardenburg's syndrome is a genetic disorder resulting in hearing loss and pigmentary abnormalities, such as two differently colored eyes. The disease was named after the Dutch ophthalmologist who first noticed that people with two differently colored eyes frequently had hearing problems as well. In the early 1990s, biologists narrowed the search for the gene implicated in Waardenburg's syndrome to human chromosome 2, but its exact location remained unknown for some time. There was another clue that shed light on the gene associated with Waardenburg's syndrome, that drew attention to chromosome 2: for a long time, breeders scrutinized mice for mutants, and one of these, designated *splotch*, had pigmentary abnormalities like patches of white spots, similar to those in humans with Waardenburg's syndrome. Through breeding, the *splotch* gene was mapped to one of the mouse chro-

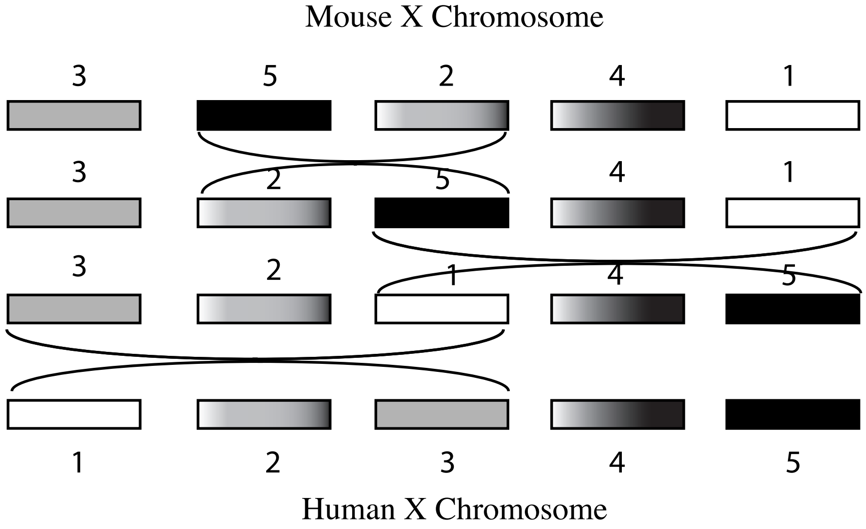

Figure 5.1 Transformation of the mouse gene order into the human gene order on the X chromosome (only the five longest synteny blocks are shown here).

mosomes. As gene mapping proceeded it became clear that there are groups of genes in mice that appear in the same order as they do in humans: these genes are likely to be present in the same order in a common ancestor of humans and mice—the ancient mammalian genome. In some ways, the human genome is just the mouse genome cut into about 300 large genomic fragments, called *synteny blocks*, that have been pasted together in a different order. Both sequences are just two different shufflings of the ancient mammalian genome. For example, chromosome 2 in humans is built from fragments that are similar to mouse DNA residing on chromosomes 1, 2, 3, 5, 6, 7, 10, 11, 12, 14, and 17. It is no surprise, then, that finding a gene in mice often leads to clues about the location of the related gene in humans.

Every genome rearrangement results in a change of gene ordering, and a series of these rearrangements can alter the genomic architecture of a species. Analyzing the rearrangement history of mammalian genomes is a challenging problem, even though a recent analysis of human and mouse genomes implies that fewer than 250 genomic rearrangements have occurred since the divergence of humans and mice approximately 80 million years ago. Every study of genome rearrangements involves solving the combinatorial puzzle of finding a series of rearrangements that transform one genome into another. Figure 5.1 presents a *rearrangement scenario* in which the mouse X chromosome is transformed into the human X chromosome.[1] The elementary rearrangement event in this scenario is the flipping of a genomic segment, called a *reversal*, or an *inversion*. One can consider other types of evolutionary events but in this book we only consider reversals, the most common evolutionary events.

Biologists are interested in the most parsimonious evolutionary scenario, that is, the scenario involving the smallest number of reversals. While there is no guarantee that this scenario represents an actual evolutionary sequence, it gives us a lower bound on the number of rearrangements that have occurred and indicates the similarity between two species.[2]

Even for the small number of synteny blocks shown, it is not so easy to verify that the three evolutionary events in figure 5.1 represent a *shortest* series of reversals transforming the mouse gene order into the human gene order on the X chromosome. The exhaustive search technique that we presented in the previous chapter would hardly work for rearrangement studies since the number of variants that need to be explored becomes enormous for more than ten synteny blocks. Below, we explore two greedy approaches that work to differing degrees of success.

5.2 Sorting by Reversals

In their simplest form, rearrangement events can be modeled by a series of reversals that transform one genome into another. The order of genes (rather, of synteny blocks) in a genome can be represented by a permutation[3]

[1] Extreme conservation of genes on X chromosomes across mammalian species provides an opportunity to study the evolutionary history of X chromosomes independently of the rest of the genomes, since the gene content of X chromosomes has barely changed throughout mammalian evolution. However, the order of genes on X chromosomes has been disrupted several times. In other words, genes that reside on the X chromosome stay on the X chromosome (but their order may change). All other chromosomes may exchange genes, that is, a gene can move from one chromosome to another.

[2] In fact, a sequence of reversals that transforms the X chromosome of mouse into the X chromosome of man does not even represent an evolutionary sequence, since humans are not descended from the present-day mouse. However, biologists believe that the architecture of the X chromosome in the human-mouse ancestor is about the same as the architecture of the human X chromosome.

[3] A permutation of a sequence of n numbers is just a reordering of that sequence. We will always use permutations of consecutive integers: for example, $2\,1\,3\,4\,5$ is a permutation of $1\,2\,3\,4\,5$.

$\pi = \pi_1\pi_2\cdots\pi_n$. The order of synteny blocks on the X chromosome in humans is represented in figure 5.1 by $(1, 2, 3, 4, 5)$, while the ordering in mice is $(3, 5, 2, 4, 1)$.[4]

A reversal $\rho(i, j)$ has the effect of reversing the order of synteny blocks

$$\pi_i\pi_{i+1}\cdots\pi_{j-1}\pi_j$$

In effect, this transforms

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

into

$$\pi\cdot\rho(i, j) = \pi_1\cdots\pi_{i-1}\underleftarrow{\pi_j\pi_{j-1}\cdots\pi_{i+1}\pi_i}\pi_{j+1}\cdots\pi_n$$

For example, if $\pi = 1\,2\,\underrightarrow{4\,3\,7\,5}\,6$, then $\pi\cdot\rho(3, 6) = 1\,2\,\underleftarrow{5\,7\,3\,4}\,6$. With this representation of a genome, and a rigorous definition of an evolutionary event, we are in a position to formulate the computational problem that mimics the biological rearrangement process.

Reversal Distance Problem:
Given two permutations, find a shortest series of reversals that transforms one permutation into another.

Input: Permutations π and σ.

Output: A series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ transforming π into σ (i.e., $\pi\cdot\rho_1\cdot\rho_2\cdots\rho_t = \sigma$), such that t is minimum.

We call t the *reversal distance between* π *and* σ, and write $d(\pi, \sigma)$ to denote the reversal distance for a given π and σ. In practice, one usually selects the second genome's order as a gold standard, and arbitrarily sets σ to be the *identity permutation* $1\,2\,\cdots\,n$. The Sorting by Reversals problem is similar to the Reversal Distance problem, except that it requires only one permutation as input.

[4] In reality, genes and synteny blocks have directionality, reflecting whether they reside on the direct strand or the reverse complement strand of the DNA. In other words, the synteny block order in an organism is really represented by a *signed* permutation. However, in this section we ignore the directionality of the synteny blocks for simplicity.

Sorting by Reversals Problem:
Given a permutation, find a shortest series of reversals that transforms it into the identity permutation.

Input: Permutation π.

Output: A series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ transforming π into the identity permutation such that t is minimum.

In this case, we call t the *reversal distance of* π and denote it as $d(\pi)$. When sorting a permutation $\pi = 1\,2\,3\,6\,4\,5$, it hardly makes sense to move the already-sorted first three elements of π. If we define *prefix*(π) to be the number of already-sorted elements of π, then a sensible strategy for sorting by reversals is to increase *prefix*(π) at every step. This approach sorts π in 2 steps: $1\,2\,3\,\underline{6\,4}\,5 \to 1\,2\,3\,4\,\underline{6\,5} \to 1\,2\,3\,4\,5\,6$. Generalizing this leads to an algorithm that sorts a permutation by repeatedly moving its ith element to the ith position.[5]

```
SIMPLEREVERSALSORT($\pi$)
for $i \leftarrow 1$ to $n - 1$
    $j \leftarrow$ position of element $i$ in $\pi$ (i.e., $\pi_j = i$)
    if $j \neq i$
        $\pi \leftarrow \pi \cdot \rho(i, j)$
        output $\pi$
    if $\pi$ is the identity permutation
        return
```

SIMPLEREVERSALSORT is an example of a greedy algorithm that chooses the "best" reversal at every step. However, the notion of "best" here is rather short-sighted—simply increasing *prefix*(π) does not guarantee the smallest number of reversals. For example, SIMPLEREVERSALSORT takes five steps to sort 6 1 2 3 4 5:

$$\underline{6\,1}\,2\,3\,4\,5 \to 1\,\underline{6\,2}\,3\,4\,5 \to 1\,2\,\underline{6\,3}\,4\,5 \to 1\,2\,3\,\underline{6\,4}\,5 \to 1\,2\,3\,4\,\underline{6\,5} \to 1\,2\,3\,4\,5\,6$$

However, the same permutation can be sorted in just two steps:

$$\underline{6\,1\,2\,3\,4\,5} \to \underline{5\,4\,3\,2\,1}\,6 \to 1\,2\,3\,4\,5\,6.$$

[5] Note the superficial similarity of this algorithm to SELECTIONSORT in chapter 2.

Therefore, we can confidently say that SIMPLEREVERSALSORT is not a correct algorithm, in the strict sense of chapter 2. In fact, despite its commonsense appeal, SIMPLEREVERSALSORT is a terrible algorithm since it takes $n-1$ steps to sort the permutation $\pi = n\,1\,2\,\ldots\,(n-1)$ even though $d(\pi) = 2$.

Even before biologists faced genome rearrangement problems, computer scientists studied the related Sorting by Prefix Reversals problem, also known as the Pancake Flipping problem: given an arbitrary permutation π, find $d_{pref}(\pi)$, which is the minimum number of reversals of the form $\rho(1, i)$ sorting π. The Pancake Flipping problem was inspired by the following "real-life" situation described by (the fictitious) Harry Dweighter:

> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to a table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips that I will ever have to use to rearrange them?

An analog of SIMPLEREVERSALSORT will sort every permutation by at most $2(n-1)$ prefix reversals. For example, one can sort $1\,2\,3\,6\,4\,5$ by 4 prefix reversals ($\underline{1\,2\,3\,6}\,4\,5 \rightarrow \underline{6\,3\,2\,1\,4\,5} \rightarrow \underline{5\,4\,1\,2\,3}\,6 \rightarrow \underline{3\,2\,1}\,4\,5\,6 \rightarrow 1\,2\,3\,4\,5\,6$) but it is not clear whether there exists an even shorter series of prefix reversals to sort this permutation. William Gates, an undergraduate student at Harvard in the mid-1970s, and Christos Papadimitriou, a professor at Harvard in the mid-1970s, now at Berkeley, made the first attempt to solve this problem and proved that any permutation can be sorted by at most $\frac{5}{3}(n+1)$ prefix reversals. However, the Pancake Flipping problem remains unsolved.

5.3 Approximation Algorithms

In chapter 2 we mentioned that, for many problems, efficient polynomial algorithms are still unknown and unlikely ever to be found. For such problems, computer scientists often find a compromise in *approximation algorithms* that produce an approximate solution rather than an optimal one.[6] The *approximation ratio of algorithm $\mathcal{A}$ on input π* is defined as $\frac{\mathcal{A}(\pi)}{OPT(\pi)}$, where $\mathcal{A}(\pi)$ is the solution produced by the algorithm $\mathcal{A}$ and $OPT(\pi)$ is the correct (optimal) solution of the problem.[7] The *approximation ratio, or performance guarantee of algorithm $\mathcal{A}$* is defined as its maximum approximation ratio over *all* inputs of size n, that is, as

$$\max_{|\pi|=n} \frac{\mathcal{A}(\pi)}{OPT(\pi)}.$$

We assume that $\mathcal{A}$ is a *minimization* algorithm, i.e., an algorithm that attempts to minimize its objective function. For maximization algorithms, the approximation ratio is

$$\min_{|\pi|=n} \frac{\mathcal{A}(\pi)}{OPT(\pi)}.$$

In essence, an approximation algorithm gives a worst-case scenario of just how far off an algorithm's output can be from some hypothetical perfect algorithm. The approximation ratio of SIMPLEREVERSALSORT is at least $\frac{n-1}{2}$, so a biologist has no guarantee that this algorithm comes anywhere close to the correct solution. For example, if n is 1001, this algorithm could return a series of reversals that is as large as 500 times the optimal. Our goal is

[6] Approximation algorithms are only relevant to problems that have a numerical objective function like minimizing the number of coins returned to the customer. A problem that does not have such an objective function (like the Partial Digest problem) does not lend itself to approximation algorithms.

[7] Technically, an approximation algorithm is not correct, in the sense of chapter 2, since there exists some input that returns a suboptimal (incorrect) output. The approximation ratio gives one an idea of just how incorrect the algorithm can be.

$$0\ \underleftarrow{2\ 1}\ \underrightarrow{3\ 4\ 5}\ \underleftarrow{8\ 7\ 6}\ 9$$

Figure 5.2 Breakpoints, adjacencies, and strips for permutation $2\,1\,3\,4\,5\,8\,7\,6$ (extended by 0 and 9 on the ends). Strips with more than one element are divided into decreasing strips (←) and increasing strips (→). The boundary between two nonconsecutive elements (in this case, 02, 13, 58, and 69) is a breakpoint; breakpoints demarcate the boundaries of strips.

to design approximation algorithms with better performance guarantees, for example, an algorithm with an approximation ratio of 2, or even better, 1.01. Of course, an algorithm with an approximation ratio of 1 (by definition, a correct and optimal algorithm) would be the acme of perfection, but such algorithms can be hard to find. As of the writing of this book, the best known algorithm for sorting by reversals has a performance guarantee of 1.375.

5.4 Breakpoints: A Different Face of Greed

We have described a greedy algorithm that attempts to maximize *prefix*(π) in every step, but any chess player knows that greed often leads to wrong decisions. For example, the ability to take a queen in a single step is usually a good sign of a trap. Good chess players use a more sophisticated notion of greed that evaluates a position based on many subtle factors rather than simply on the face value of a piece they can take.

The problem with SIMPLEREVERSALSORT is that *prefix*(π) is a naive measure of our progress toward the identity permutation, and does not accurately reflect how difficult it is to sort a permutation. Below we define *breakpoints* that can be viewed as "bottlenecks" for sorting by reversals. Using the number of breakpoints, rather than *prefix*(π), as the basis of greed leads to a better algorithm for sorting by reversals, in the sense that it produces a solution that is closer to the optimal one.

It will be convenient for us to extend the permutation $\pi_1 \cdots \pi_n$ by $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ on the ends. To be clear, we do not move π_0 or π_{n+1} during the process of sorting. We call a pair of neighboring elements π_i and π_{i+1}, for $0 \le i \le n$, an *adjacency* if π_i and π_{i+1} are consecutive numbers; we call the pair a *breakpoint* if not. The permutation in figure 5.2 has five adjacencies ($2\,1$, $3\,4$, $4\,5$, $8\,7$, and $7\,6$) and four breakpoints ($0\,2$, $1\,3$, $5\,8$, and $6\,9$). A permutation on n elements may have as many as $n+1$ breakpoints (e.g., the permutation $0\,6\,1\,3\,5\,7\,2\,4\,8$ on seven elements has eight breakpoints) and as few as 0 (the identity permutation $0\,1\,2\,3\,4\,5\,6\,7\,8$).[8] Every breakpoint corresponds to a pair of elements π_i and π_{i+1} that are neighbors in π but not in the identity permutation. In fact, the identity permutation is the only permutation with no breakpoints at all. Therefore, the nonconsecutive elements π_i and π_{i+1} forming a breakpoint must be separated in the process of transforming π to the identity, and we can view sorting by reversals as the process of eliminating breakpoints. The observation that every reversal can eliminate *at most* two breakpoints (one on the left end and another on the right end of the reversal) immediately implies that $d(\pi) \geq \frac{b(\pi)}{2}$, where $b(\pi)$ is the number of breakpoints in π. The algorithm BreakpointReversalSort eliminates as many breakpoints as possible in every step in order to reach the identity permutation.

```
BreakpointReversalSort($\pi$)
while $b(\pi) > 0$
    Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
    $\pi \leftarrow \pi \cdot \rho$
    output $\pi$
return
```

One problem with this algorithm is that it is not clear why BreakpointReversalSort is a better approximation algorithm than SimpleReversalSort. Moreover, it is not even obvious yet that BreakpointReversalSort terminates! How can we be sure that removing some breakpoints does not introduce others, leading to an endless cycle?

We define a *strip* in a permutation π as an interval between two consecutive breakpoints, that is, as any maximal segment without breakpoints (see figure 5.2). For example, the permutation 0 2 1 3 4 5 8 7 6 9 consists of five strips: 0, 2 1, 3 4 5, 8 7 6, and 9. Strips can be further divided into *increasing* strips (3 4 5) and *decreasing* strips (2 1) and (8 7 6). Single-element strips can be considered to be either increasing or decreasing, but it will be convenient to

[8] We remind the reader that we extend permutations by 0 and $n+1$ on their ends, thus introducing potential breakpoints in the beginning and in the end.

define them as decreasing (except for elements 0 and $n+1$ which will always be classified as increasing strips).

We present the following theorems, first to show that endless cycles of breakpoint removal cannot happen, and then to show that the approximation ratio of the algorithm is 4. While the notion of "theorem" and "proof" might seem overly formal for what is, at heart, a biological problem, it is important to consider that we have modeled the biological process in mathematical terms. We are proving analytically that the algorithm meets certain expectations. This notion of proof without experimentation is very different from what a biologist would view as proof, but it is just as important when working in bioinformatics.

Theorem 5.1 *If a permutation π contains a decreasing strip, then there is a reversal ρ that decreases the number of breakpoints in π, that is, $b(\pi \cdot \rho) < b(\pi)$.*

Proof: Among all decreasing strips in π, choose the strip containing the smallest element k ($k = 3$ for permutation $\underrightarrow{0\,1\,2}\,\underleftarrow{7\,6\,5}\,\underleftarrow{8}\,\underleftarrow{4\,3}\,\underrightarrow{9}$). Element $k-1$ in π cannot belong to a decreasing strip, since otherwise we would choose a strip ending at $k-1$ rather than a strip ending at k. Therefore, $k-1$ belongs to an increasing strip; moreover, it is easy to see that $k-1$ terminates this strip (for permutation $\underrightarrow{0\,1\,2}\,\underleftarrow{7\,6\,5}\,\underleftarrow{8}\,\underrightarrow{4\,3}\,\underrightarrow{9}$, $k-1=2$ and 2 is at the right end of the increasing strip $0\,1\,2$). Therefore elements k and $k-1$ correspond to two breakpoints, one at the end of the decreasing strip ending with k and the other at the end of the increasing strip ending in $k-1$. Reversing the segment between k and $k-1$ brings them together, as in $0\,1\,2\,\underline{7\,6\,5\,8\,4\,3}\,9 \rightarrow \underrightarrow{0\,1\,2\,3\,4}\,\underleftarrow{8}\,\underrightarrow{5\,6\,7}\,\underrightarrow{9}$, thus reducing the number of breakpoints in π. □

For example, BREAKPOINTREVERSALSORT may perform the following four steps when run on the input $(0\,8\,2\,7\,6\,5\,1\,4\,3\,9)$ in order to reduce the number of breakpoints:

$$\begin{array}{ll}
(\underrightarrow{0}\;\underleftarrow{8}\;\underleftarrow{2}\;\underleftarrow{7\;6\;5}\;\underleftarrow{1}\;\underleftarrow{4}\;\underleftarrow{3}\;\underrightarrow{9}) & b(\pi)=6 \\
(\underrightarrow{0}\;\underleftarrow{2}\;\underleftarrow{8\;7\;6\;5}\;\underleftarrow{1}\;\underleftarrow{4}\;\underleftarrow{3}\;\underrightarrow{9}) & b(\pi)=5 \\
(\underrightarrow{0}\;\underrightarrow{2\;\;3\;\;4}\;\underleftarrow{1}\;\underrightarrow{5\;\;6\;\;7\;\;8\;\;9}) & b(\pi)=3 \\
(\underrightarrow{0}\;\underleftarrow{4\;\;3\;\;2\;\;1}\;\underrightarrow{5\;\;6\;\;7\;\;8\;\;9}) & b(\pi)=2 \\
(\underrightarrow{0\;\;1\;\;2\;\;3\;\;4\;\;5\;\;6\;\;7\;\;8\;\;9}) & b(\pi)=0
\end{array}$$

In this case, BREAKPOINTREVERSALSORT steadily reduces the number of breakpoints in every step of the algorithm. In other cases, (e.g., the permutation $(\underrightarrow{0\,1}\,\underrightarrow{5\,6\,7}\,\underrightarrow{2\,3\,4}\,\underrightarrow{8\,9})$ without decreasing strips), no reversal reduces the number of breakpoints. In order to overcome this, we can simply find any increasing strip (excluding π_0 and π_{n+1}, of course) and flip it. This creates a decreasing strip and we can proceed.

```
IMPROVEDBREAKPOINTREVERSALSORT($\pi$)
while $b(\pi) > 0$
    if $\pi$ has a decreasing strip
        Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
    else
        Choose a reversal $\rho$ that flips an increasing strip in $\pi$
    $\pi \leftarrow \pi \cdot \rho$
    output $\pi$
return
```

The theorem below demonstrates that such "no progress" situations do not happen too often in the course of IMPROVEDBREAKPOINTREVERSALSORT. In fact, the theorem quanitifes exactly how often those situations could possibly occur and provides an approximation ratio guarantee.

Theorem 5.2 IMPROVEDBREAKPOINTREVERSALSORT *is an approximation algorithm with a performance guarantee of at most 4.*

Proof: Theorem 5.1 implies that as long as π has a decreasing strip, IMPROVEDBREAKPOINTREVERSALSORT reduces the number of breakpoints in π. On the other hand, it is easy to see that if all strips in π are increasing, then there might not be a reversal that reduces the number of breakpoints. In this case IMPROVEDBREAKPOINTREVERSALSORT finds a reversal ρ that reverses an increasing strip(s) in π. By reversing an increasing strip, ρ creates a decreasing strip in π implying that IMPROVEDBREAKPOINTREVERSALSORT will be able to reduce the number of strips at the next step. Therefore, for every "no progress" step, IMPROVEDBREAKPOINTREVERSALSORT will make progress at the next step which means that IMPROVEDBREAKPOINTREVERSALSORT eliminates at least one breakpoint in every two steps. In the worst-case scenario, the number of steps in IMPROVEDBREAKPOINTREVERSALSORT is at most $2b(\pi)$ and its approximation ratio is at most $\frac{2b(\pi)}{d(\pi)}$. Since $d(\pi) \geq \frac{b(\pi)}{2}$, IMPROVEDBREAKPOINTREVERSALSORT has a performance guarantee[9] bounded above by $\frac{2b(\pi)}{d(\pi)} \leq \frac{2b(\pi)}{\frac{b(\pi)}{2}} = 4$. □

[9] To be clear, we are not claiming that IMPROVEDBREAKPOINTREVERSALSORT will take four times as long, or use four times as much memory as an (unknown) optimal algorithm. We