Sorting with Recurrent Comparison Errors[*][†]

Barbara Geissmann[1], Stefano Leucci[2], Chih-Hung Liu[3], and Paolo Penna[4]

1   **Department of Computer Science, ETH Zürich, Switzerland**
    `barbara.geissmann@inf.ethz.ch`
2   **Department of Computer Science, ETH Zürich, Switzerland**
    `stefano.leucci@inf.ethz.ch`
3   **Department of Computer Science, ETH Zürich, Switzerland**
    `chih-hung.liu@inf.ethz.ch`
4   **Department of Computer Science, ETH Zürich, Switzerland**
    `paolo.penna@inf.ethz.ch`

—— Abstract ——

We present a sorting algorithm for the case of *recurrent* random comparison *errors*. The algorithm essentially achieves simultaneously good properties of previous algorithms for sorting n distinct elements in this model. In particular, it runs in $O(n^2)$ time, the *maximum dislocation* of the elements in the output is $O(\log n)$, while the *total dislocation* is $O(n)$. These guarantees are the best possible since we prove that even randomized algorithms cannot achieve $o(\log n)$ maximum dislocation with high probability, or $o(n)$ total dislocation in expectation, regardless of their running time.

1   Introduction

We study the problem of *sorting n distinct elements* under *recurrent* random comparison *errors*. In this classical model, each comparison is wrong with some fixed probability p, and correct with probability $1 - p$. The probability of errors are independent over all possible pairs of elements, but errors are recurrent: If the same comparison is repeated at any time during the computation, the result is always the same, i.e., always wrong or always correct.

In such a scenario not all sorting algorithms perform equally well in terms of the output, as some of them are more likely to produce "nearly sorted" sequences than others. To measure the quality of an output permutation in terms of sortedness, a common way is to consider the *dislocation* of an element, which is the difference between its position in the permutation and its true rank among all elements. Two criteria based on the dislocations of the elements are the *total dislocation* of a permutation, i.e., the sum of the dislocations of all n elements, and the *maximum dislocation* of any element in the permutation. Naturally, the running *time* remains an important criteria for evaluating sorting algorithms.

To the best of our knowledge, for recurrent random comparison errors, the best results with respect to *running time*, *maximum*, and *total dislocation* are achieved by the following two different algorithms:

28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 38; pp. 38:1–38:12
Leibniz International Proceedings in Informatics
LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Our and previous results. The constant c in [2] depends on both the error probability p, and the success probability of the algorithm. For example, for a success probability of $1 - 1/n$, the analysis in [2] yields $c = \frac{110525}{(1/2-p)^4}$. The total dislocation in [12] follows trivially by the maximum dislocation, and no further analysis is given.

	Braverman and Mossel [2]	Klein et al [12]	Ours
# Steps	$O(n^{3+24c})$	$O(n^2)$	$O(n^2)$
Maximal Dislocation	w.h.p. $O(\log n)$	w.h.p. $O(\log n)$	w.h.p. $O(\log n)$
Total Dislocation	w.h.p. $O(n)$	w.h.p. $O(n \log n)$	in exp. $O(n)$

≈ Braverman and Mossel [2] give an algorithm which guarantees maximum dislocation $O(\log n)$ and total dislocation $O(n)$, both with high probability. The main drawback of this algorithm seems to be its running time, as the constant exponent can be rather high;

≈ Klein et al. [12] give a much faster $O(n^2)$-time algorithm which guarantees maximum dislocation $O(\log n)$, with high probability. They however do not provide any upper bound on the total dislocation, which by the previous result is obviously $O(n \log n)$.

In this paper we investigate whether it is possible to guarantee *all* of these bounds together, that is, if there is an algorithm with running time $O(n^2)$, maximum dislocation $O(\log n)$, and total dislocation $O(n)$.

1.1 Our contribution

We propose a new algorithm whose performance guarantees are essentially the best of the two previous algorithms (see Table 1). Indeed, our algorithm WINDOW SORT takes $O(n^2)$ time and guarantees the maximum dislocation to be $O(\log n)$ with probability $1 - 1/n$ and the expected total dislocation to be $O(n)$. The main idea is to iteratively sort n elements by comparing each element with its neighbouring elements lying within a *window* and to halve the window size after every iteration. In each iteration, each element is assigned a rank based on the local comparisons, and then they are placed according to the computed ranks.

Our algorithm is inspired by Klein et al.'s algorithm [12] which distributes elements into *buckets* according to their computed ranks, compares each element with elements in neighboring buckets to obtain a new rank, and halves the range of a bucket iteratively. Note however that the two algorithms operate in a different way, essentially because of the following key difference between *bucket* and *window*. The number of elements in a bucket is not fixed, since the computed rank of several elements could be the same. In a window, instead, the number of elements is *fixed*. This property is essential in the analysis of the total dislocation of WINDOW SORT, but introduces a potential *offset* between the computed rank and the computed position of an element. Our analysis consists in showing that such an offset is sufficiently small, which we do by considering a number of "delicate" conditions that the algorithm should maintain throughout its execution with sufficiently high probability.

We first describe a standard version of our algorithm which achieves the afore mentioned bounds for any error probability $p < 1/32$. We then improve this result to $p < 1/16$ by using the idea of shrinking the window size at a different rate. An experimental evaluation of our algorithms –which, due to space limitations, can be found in the full version of the paper– shows that the performance of the standard version is significantly better than the theoretical guarantees. In particular, the experiments suggest that the expected total dislocation is $O(n)$ for $p < 1/5$, while the maximum dislocation is $O(\log n)$ for $p < 1/4$.

In addition, we prove that no sorting algorithm can guarantee the maximum dislocation to be $o(\log n)$ with high probability, and no sorting algorithm can guarantee the expected total dislocation to be $o(n)$.

1.2 Further Related Work on Sorting with Comparison Errors

Computing with errors is often considered in the framework of a two-person game called *Rényi-Ulam Game*: The *responder* thinks of an object in the search space, and the *questioner* has to find it by asking questions to which the responder provides answers. However, some of the answers are incorrect on purpose; the responder is an *adversarial lier*. These games have been extensively studied in the past on various kinds of search spaces, questions, and errors; see Pelc's survey [14] and Cicalese's monograph [3].

Feige et al. [7] studied several comparison based algorithms with independent random errors where the error probability of a comparison is less than half, the repetitions of an comparison can obtain different outcomes, and all the comparisons are independent. They required the reported solution to be correct with a probability $1 - q$, where $0 < q < 1/2$, and proved that for sorting, $O(n \log(n/q))$ comparisons suffice, which gives also the running time. In the same model, sorting by random swaps represented as Markovian processes have been studied under the question of the number of inversions (reversed pairs) [8, 9], which is within a constant factor of the total dislocation [15]. Karp and Kleinberg [11] studied a noisy version of the classic binary search problem, where elements cannot be compared directly. Instead, each element is associated with a coin that has an unknown probability of observing heads when tossing it and these probabilities increase when going through the sorted order.

For recurring errors, Coppersmith and Rurda [4] studied a simple algorithm that gives a 5-approximation on the weighted feedback arc set in tournaments (FAST) problem if the weights satisfy probability constraints. The algorithm consists of ordering the elements based on computed ranks, which for unweighted FAST are identical to our computed ranks. Damaschke [6] also gave a *subquadratic* time algorithm returning a sequence with $O(k)$ inversions when at most k errors can occur. Alonso et al. [1] and Hadjicostas and Lakshamanan [10] studied Quicksort and recursive Mergesort, respectively, with random comparison errors.

Paper organization

The rest of this paper is organized as follows. We present the Window Sort algorithm in Section 2 and analyze the maximum and total dislocation in Section 3 and Section 4, respectively. Then, we explain how to modify Window Sort to allow larger error probabilities in Section 5. Additionally, we provide a lower bounds on both the maximum and average dislocation for any sorting algorithm (due to space limitations the corresponding proofs are omitted and can be found in the full version of the paper).

2 Window Sort

Window Sort consists of multiple iterations of the same procedure: Starting with a permutation σ and a *window size* w, we compare each element x in σ with its left $2w$ and right $2w$ adjacent elements (if they exist) and count its *wins*, i.e., the number of times a comparison outputs x as the larger element. Then, we obtain the *computed rank* for each element based on its *original position* in σ and its wins: if $\sigma(x)$ denotes the original position of x in σ, the computed rank of x equals $\max\{0, \sigma(x) - 2w\}$ plus the number of its wins. And we get a new permutation σ' by placing the elements ordered by their computed ranks.

Algorithm 1: WINDOW SORT (on a permutation σ on n elements)

Initialization: The initial window size is $w = n/2$. Each element x has two variables $wins(x)$ and $computed\_rank(x)$ which are set to zero.

repeat

    1. **foreach** x at position $l = 1, 2, 3, \ldots, n$ in σ **do**

        foreach y whose position in σ is in $[l - 2w, l - 1]$ *or* in $[l + 1, l + 2w]$ **do**

            if $x > y$ **then**

                $wins(x) = wins(x) + 1$

        $computed\_rank(x) = \max\{l - 2w, 0\} + wins(x)$

    2. Place the elements into σ' ordered by non-decreasing $computed\_rank$, break ties arbitrarily.

    3. Set all $wins$ to zero, $\sigma = \sigma'$, and $w = w/2$.

until $w < 1$;

Finally, we set $w' = w/2$ and start a new iteration on σ' with window size w'. In the very first iteration, $w = n/2$. We formalize WINDOW SORT in Algorithm 1.

In the following, w.l.o.g. we assume to sort elements $\{1, \ldots, n\}$, i.e., we refer to both an element x and its rank by x. Let σ denote the permutation of the elements at the beginning of the current iteration of WINDOW SORT and let σ' denote the permutation obtained after this iteration (i.e., the permutation on which the next iteration performs). Similarly, let w and $w' = w/2$ denote the window size of the current and the next iteration. Furthermore, let π denote the sorted permutation. We define four important terms for an element x in σ:

- *Current/Original position*: The position of x in σ: $\sigma(x)$
- *Computed rank*: The current position of x minus $2w$ (zero if negative) plus its number of wins: $computed\_rank(x)$
- *Computed position*: The position of x in σ': $\sigma'(x)$

▶ **Theorem 1.** WINDOW SORT *takes $O(n^2)$ time.*

Proof. Consider the three steps in Algorithm 1. The number of comparisons in an iteration in the outer loop is $4w$, for w the current size of the window. Therefore, the first step needs $O(nw)$ time. For the second step we could apply for instance Counting Sort (see e.g. [5]), which takes $O(n)$ time, since all computed ranks lie between zero and n. Thus, the total running time is upper bounded by $\sum_{i=0}^{\infty} O(\frac{4n^2}{2^i}) = O(8n^2)$. ◀

2.1   Preliminaries

We first introduce a condition on the errors in comparisons between an element x and a fixed subset of elements which depends only on the window size w.

▶ **Definition 2.** We define $ERRORS(x, w)$ as the set of errors among the comparisons between x and every $y \in [x - 4w, x + 4w]$.

▶ **Theorem 3.** WINDOW SORT *returns a sequence of maximum dislocation at most $9w^\star$ whenever the initial comparisons are such that*

$$|ERRORS(x, w)| \leq w/4 \tag{1}$$

hold for all elements x and for all $w = n/2, n/4, \ldots, 2w^\star$.

The proof of this theorem follows in the end of this section. In the analysis, we shall prove the following:

- If the computed rank of each element is close to its (true) rank, then the dislocation of each elements is small (Lemma 7).
- The computed rank of each element is indeed close to its (true) rank if the number of errors involving the element under consideration is small (Lemma 4).
- The number of positions an element can move in further iterations is small (Lemma 8).

We now introduce a condition that implies Theorem 3: Throughout the execution of WINDOW SORT we would like every element x to satisfy the following condition:

(∗) For window size w, the dislocation of x is at most w.

We also introduce two further conditions, which essentially relax the requirement that all elements satisfy **(∗)**. The first condition justifies the first step of our algorithm, while the second condition restricts the range of elements that get compared with x in some iteration:

(●) For window size w, element x is larger (smaller) than all the elements lying apart by more than $2w$ positions left (right) of x's original position.

(○) For window size w, x and its left $2w$ and right $2w$ adjacent elements satisfy condition **(∗)**.

Note that if **(∗)** holds for all elements, then **(●)** and **(○)** also hold for all elements. For elements that satisfy both **(●)** and **(○)**, the computed rank is close to the true rank if there are few errors in the comparisons:

▶ **Lemma 4.** *For every window size w, if an element x satisfies satisfy both **(●)** and **(○)**, then the absolute difference between the computed rank and its true rank is bounded by*

$$|computed\_rank(x) - x| \leq |ERRORS(x, w)| \ .$$

Proof. This follows immediately from condition **(●)**. ◀

We now consider the difference between the computed rank and the computed position of an element, which we define as the *offset* of this element. Afterwards, we consider the difference between the original position and the computed position of an element.

▶ **Fact 5.** *Observe that by Step 1 of the algorithm it holds that, for every permutation σ, every window size w and every element x, the difference between $\sigma(x)$ and the computed rank of x is at most $2w$, $|computed\_rank(x) - \sigma(x)| \leq 2w$.*

▶ **Lemma 6.** *For any permutation of n elements and for each element x, if the difference between the computed rank and x is at most m for every element, then the difference between the computed position and x is at most $2m$ for every element.*

The proof of this lemma is analogue to the proof of Lemma 7 below.

▶ **Lemma 7.** *For every permutation σ and window size w, the offset of every element x is at most $2w$, $|computed\_rank(x) - \sigma'(x)| \leq 2w$.*

Proof. Let the computed rank of x be k. The computed position $\sigma'(x)$ is larger than the number of elements with computed rank smaller than k, and at most the number of elements with computed ranks at most k. By Fact 5, every element y with $\sigma(y) < k - 2w$ has a computed rank smaller than k, and every element y with $\sigma(y) > k + 2w$ has a computed rank larger than k. ◀

▶ **Lemma 8.** *Consider a generic iteration of the algorithm with permutation σ and a window size w. In this iteration, the position of each element changes by at most $4w$. Moreover, the position of each element changes by at most $8w$ until the algorithm terminates.*

Proof. By Fact 5, Lemma 7 and triangle inequality,

$$|\sigma(x) - \sigma'(x)| \leq |computed\_rank(x) - \sigma(x)| + |computed\_rank(x) - \sigma'(x)| \leq 2w + 2w = 4w.$$

Since w is halved after every iteration, the final difference is at most $\sum_{i=0}^{\infty} \frac{4w}{2^i} = 8w$.   ◀

Finally, we conclude Theorem 3 and show that the dislocation of an element is small if the number of errors is small:

Proof of Theorem 3. Consider an iteration of the algorithm with current window size w. We show that, if (∗) holds for all elements in the current iteration, then (1) implies that (∗) also holds for all elements in the next iteration, i.e., when the window size becomes $w/2$. In order for (∗) to hold for the next iteration, the computed *position* of each element should differ from the true rank by at most $w/2$,

$$|\sigma'(x) - x| \leq w/2 \ .$$

By Lemma 6, it is sufficient to require that the computed *rank* of each element differs from its true rank by at most $w/4$,

$$|computed\_rank(x) - x| \leq w/4$$

By Lemma 4, the above inequality follows from the hypothesis $|ERRORS(x, w)| \leq w/4$.

We have thus shown that after the iteration with window size $2w^*$, all elements have dislocation at most w^*. By Lemma 8, the subsequent iterations will move each element by at most $8w^*$ positions.   ◀

▶ Remark. If we care only about the maximum dislocation, then we could obtain a better bound of w by simply stopping the algorithm at the iteration where the window size is w (for a w which guarantees the condition above with high probability). In order to bound also the total dislocation, we let the algorithm continue all the way until window size $w = 1$. This will allow us to show that the total dislocation is linear in expectation.

3   Maximum Dislocation

In this section we give a bound on the maximum dislocation of an element after running WINDOW SORT on n elements. We prove that it is a function of n and of the probability p that a single comparison fails. Our main result is the following:

▶ **Theorem 9.** *For a set of n elements, with probability $1 - 1/n$, the maximum dislocation after running* WINDOW SORT *is $9 \cdot f(p) \cdot \log n$ where*

$$f(p) = \begin{cases} \frac{400p}{(1-32p)^2} & \text{for} \quad 1/64 < p < 1/32, \\ \frac{4}{\ln\left(\frac{1}{32p}\right) - (1-32p)} & \text{for} \quad 1/192 < p \leq 1/64, \\ 6 & \text{for} \quad p \leq 1/192. \end{cases}$$

It is enough to prove that the condition in Theorem 3 holds for all $w \geq 2f(p) \log n$ with probability at least $1 - 1/n$.

▶ **Lemma 10.** *For every fixed element x and for every fixed window size $w \geq 2f(p) \log n$, the probability that*

$$|ERRORS(x, w)| > w/4 \tag{2}$$

is at most $1/n^3$.

By the union bound, the probability that (2) holds for some x and for some w is at most $1/n$. That is, the condition of Theorem 3 holds with probability at least $1 - 1/n$ for all $w \geq 2w^\star = 2f(p) \log n$, which then implies Theorem 9.

3.1 Proof of Lemma 10

Since each comparison fails with probability p independently of the other comparisons, the probability that the event in (2) happens is equal to the probability that at least $w/4$ errors occur in $8w$ comparisons. We denote such probability as $\Pr(w)$, and show that $\Pr(w) \leq 1/n^3$.

We will make use of the following standard Chernoff Bounds (see for instance in [13]):

▶ **Theorem 11** (Chernoff Bounds). *Let $X_1, \cdots, X_n$ be independent Poisson trials with $\Pr(X_i) = p_i$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathrm{E}[X]$. Then the following bounds hold:*

$$(i) \text{ For } 0 < \delta < 1, \qquad \Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{\mu\delta^2}{3}}, \tag{3}$$

$$(ii) \text{ For any } \delta > 0, \qquad \Pr(X \geq (1+\delta)\mu) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu, \tag{4}$$

$$(iii) \text{ For } R \geq 6\mu, \qquad \Pr(X \geq R) \leq 2^{-R}. \tag{5}$$

▶ **Lemma 12.** *The probability $\Pr(w)$ (at least $w/4$ errors occur in $8w$ comparisons) satisfies*

$$\Pr(w) \leq \begin{cases} e^{-\frac{w(1-32p)^2}{384p}} & \text{for} \quad 1/64 < p < 1/32, \\ \left(\frac{e^{\frac{1-32p}{32p}}}{\left(\frac{1}{32p}\right)^{\frac{1}{32p}}}\right)^{8wp} & \text{for} \quad 1/192 < p \leq 1/64, \\ 2^{-\frac{w}{4}} & \text{for} \quad p \leq 1/192. \end{cases}$$

Proof. Let the random variable X denote the number of errors in the outcome of $8w$ comparisons. Clearly, $\mathrm{E}[X] = 8wp$, and

$$\Pr(w) = \Pr\left[X \geq \frac{w}{4}\right] = \Pr\left[X \geq \frac{\mathrm{E}[X]}{32p}\right] = \Pr\left[X \geq \left(1 + \frac{1-32p}{32p}\right)\mathrm{E}[X]\right].$$

Let $\delta = \frac{1-32p}{32p}$. If $1/64 < p < 1/32$, then $0 < \delta < 1$, and by Theorem 11, case (i), we have

$$\Pr(w) \leq e^{-\frac{1}{3}\delta^2\mu} \leq e^{-\frac{w\cdot(1-32p)^2}{384p}}.$$

Similarly, if $p \leq 1/64$, then $\delta \geq 1$, and by Theorem 11, case (ii), we have

$$\Pr(w) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu \leq \left(\frac{e^{\frac{1-32p}{32p}}}{\left(\frac{1}{32p}\right)^{\frac{1}{32p}}}\right)^{8wp}.$$

If $p \leq 1/192$, then $w/4 \geq 48wp = 6\,\mathrm{E}[X]$, and by Theorem 11 case (iii), $\Pr(w) \leq 2^{-\frac{w}{4}}$. ◀

▶ **Lemma 13.** *If $w \geq 2\,f(p) \log n$, with $n \geq 1$ and $f(p)$ as in Theorem 9, then $\Pr(w) \leq 1/n^3$.*

Proof. We show the first case, the other two are similar. If $1/64 < p < 1/32$, by Lemma 12,

$$\Pr(w) \leq e^{-\frac{w\cdot(1-32p)^2}{384p}} \leq e^{-\frac{800}{384}\log n} \leq e^{-3\lg n} \leq 1/n^3. \qquad ◀$$

4   Total Dislocation

In this section, we prove that WINDOW SORT orders n elements such that their total dislocation is linear in n times a factor which depends only on p:

▶ **Theorem 14.** *For a set of n elements, the expected total dislocation after running* WINDOW SORT *is at most* $n \cdot 60\, f(p) \log f(p)$.

The key idea is to show that for an element x, only $O(w)$ elements adjacent to its (true) rank matter in all upcoming iterations. If this holds, it is sufficient to keep the following *weak invariant* for an element x throughout all iterations:

(♣) This invariant consists of three conditions that have to be satisfied:
   (a) x satisfies condition **(∗)**.
   (b) All elements with original position in $[x - 12w, x + 12w]$ satisfy condition **(∗)**.
   (c) All elements with original position in $[x - 10w, x + 10w]$ satisfy condition **(•)**.

Note that if x satisfies **(♣)**, all elements lying in $[x - 10w, x + 10w]$ satisfy both **(•)** and **(○)**.

The rest of this section is structured as follows: First we derive several properties of the weak invariant, then we prove an $n \log \log n$ bound on the expected total dislocation, and finally we extend the proof to achieve the claimed linear bound.

4.1   Properties of the Weak Invariant

We start with the key property of the weak invariant **(♣)** for some element x.

▶ **Lemma 15.** *Let σ be the permutation of n elements and w be the window size of some iteration in* WINDOW SORT*. If the weak invariant* **(♣)** *holds for an element x in σ and the computed rank of every element y with $\sigma(y) \in [x - 10w, x + 10w]$ differs from y by at most $w/4$, then* **(♣)** *still holds for x in the permutation σ' of next iteration with window size $w/2$.*

Proof. Consider the set X of all elements y with *computed_rank*$(y) \in [x - 8w, x + 8w]$. Their computed ranks differ from their original positions by at most $2w$. Thus, all these elements are in the set $Y \supseteq X$ of all elements whose original positions are in $[x - 10w, x + 10w]$. By the assumption of the lemma, for each element $y \in Y$, $|computed\_rank(y) - y| \leq w/4$. Using the same reasoning as in the proof of Lemma 7, we conclude that

$$\text{for each element } y \in X, \ |\sigma'(y) - y| \leq w/2. \tag{A}$$

Consider the set Z of all elements y with $\sigma'(y) \in [x - 6w, x + 6w]$. By Lemma 7, their computed ranks lie in $[x - 8w, x + 8w]$, thus $Z \subseteq X$, and by (A), $|\sigma'(y) - y| \leq w/2$ for each $y \in Z$. Thus, the second condition of **(♣)** holds for the next iteration.

We continue with the third condition. Consider the set $T \subseteq Z$ of all elements y with $\sigma'(y) \in [x - 5w, x + 5w]$. By the assumptions of the lemma, $y \in [x - 5w - w/2, x + 5w + w/2]$ and $\sigma(y) \in [x - 5w - 3w/2, x + 5w + 3w/2]$ for all $y \in T$. It is sufficient to show that every element in T is larger (or smaller) than all elements whose computed positions are smaller than $x - 6w$ (or larger than $x + 6w$), the rest follows from the second condition. We show the former case, the latter is symmetric. We distinguish three subcases: elements $y \in T$ with $\sigma'(y) < x - 6w$ and with $\sigma(y)$ (i) smaller than $x - 12$, (ii) between $x - 12$ and $x - 10w - 1$, or (iii) between $x - 10w$ and $x - 4w - 1$.

(i) This case follows immediately from the third condition of (♣).

(ii) This case follows immediately from the second condition of (♣).

(iii) By the assumption of our lemma, $|computed\_rank(y) - y| \le w/4$. Thus, if the computed rank r of such an element y is smaller than $x - 6w$, then $y < x - 6w + w/4$. Otherwise, if $r \ge kx - 6w$, then by (A), $|y - \sigma'(y)| \le w/2$. Thus, $y < x - 6w + w/2$.

Since we assume (♣) for x, $\sigma(x) \in [x-w, x+w]$ and $computed\_rank(x) \in [x-3w, x+3w]$. By Lemma 7, $\sigma'(x) \in [k-5w, k+5w]$, and thus $x \in Z$, which implies that the first condition of (♣) will still be satisfied for x for the next iteration. This concludes the proof. ◀

Next, we adopt Lemma 13 to analyze the probability of keeping the weak invariant for an element x and an arbitrary window size through several iteration of WINDOW SORT.

▶ **Lemma 16.** *Consider an iteration of* WINDOW SORT *on a permutation σ on n elements such that the window size is $w \ge 2\,f(p)\log w$, where $f(p)$ is defined as in Theorem 9. If the weak invariant* (♣) *for an element x holds, then with probability at least $1 - 42/w^2$,* (♣) *still holds for x when the window size is $f(p)\log w$ (after some iterations of* WINDOW SORT*).*

Proof. By Lemma 15, the probability that (♣) fails for x before the next iteration is $(20w + 1) \cdot \Pr(w)$. Let $r = \log(\frac{w}{2\,f(p)logw})$, then the probability that (♣) fails for x during the iterations from window size w to window size $f(p) \cdot \log w$ is

$$\sum_{i=0}^{r} \left(\frac{20w}{2^i} + 1 \right) \cdot \Pr\left(\frac{w}{2^i} \right) \le \sum_{i=0}^{r} \left(\frac{21w}{2^i} \right) \cdot \Pr\left(2\,f(p)\log w \right) \le 42w \cdot \Pr\left(2\,f(p)\log w \right),$$

where the first inequality is by fact that $\Pr(w)$ increases when w decreases. By Lemma 13, $\Pr(2\,f(p)\log w) \le 1/w^3$, leading to the statement. ◀

4.2 Double Logarithmic Factor (Main Idea)

Given that WINDOW SORT guarantees maximum dislocation at most $9\,f(p)\log n$ with probability at least $(1 - 1/n)$ (Theorem 9), this trivially implies that the expected total dislocation is at most $O(f(p)\log n)$. More precisely, the expected dislocation is at most

$$(1/n) \cdot n \cdot n + (1 - 1/n) \cdot n \cdot 9\,f(p)\log n \le n \cdot (1 + 9\,f(p)\log n),$$

since a fraction $1/n$ of the elements is dislocated by at most n, while the others are dislocated by at most $9 \cdot f(p)\log n$.

We next describe how to improve this to $O(f(p)\log\log n)$ by considering in the analysis *two phases* during the execution of the algorithm:

- **Phase 1:** The first phase consists of the iterations up to window size $w = f(p)\log n$. With probability at least $(1 - 1/n)$ all elements satisfy (∗) during this phase.
- **Phase 2:** The second phase consists of the executions up to window size $w' = f(p)\log w$. If all elements satisfied (∗) at the end of the previous phase, then the probability that a fixed element violates (♣) during this second phase is at most $42/w^2$.

More precisely, by Theorem 9 and the proof of Theorem 3, the probability that (∗) holds for all elements when the window size is $f(p)\log n$ is at least $(1 - 1/n)$. We thus restart our analysis with $w = f(p)\log n$ and the corresponding permutation σ. Assume an element x satisfies (♣). By Lemma 16, the probability that (♣) fails for x before the window size is $f(p)\log w$ is at most $42/w^2$. By Lemma 8, an element moves by at most $8w$ positions from its original position, which is at most w apart from its true rank. Therefore, the expected dislocation of an element x is at most

$$(1/n) \cdot n + 42/w^2 \cdot 9w + 9\,f(p)\log w = O(1) + 9\,f(p)\log(f(p)\log n),$$

where the equality holds for sufficiently large n because $w = f(p)\log n$.

4.3   Linear Dislocation (Proof of Theorem 14)

In this section, we apply a simple idea to decrease the upper bound on the expected total dislocation after running WINDOW SORT on n elements to $60\,f(p)\log f(p)$. We recurse the analysis from the previous Section 4.2 for several *phases*: Roughly speaking, an *iteration* in WINDOW SORT halves the window size, a *phase* of iterations logarithmizes the window size.

- **Phase 1**: Iterations until the window size is $f(p)\log n$.
- **Phase 2:** Subsequent iterations until the window size is $f(p)\cdot\log(f(p)\log n)$.
- **Phase 3:** Subsequent iterations until the window size is $f(p)\cdot\log(f(p)\cdot\log(f(p)\log n))$.
- . . .

We bound the expected dislocation of an element x, and let w_i denote the window size after the i-th phase. We have $w_0 = n$, $w_1 = f(p)\log n$, $w_2 = f(p)\log(f(p)\log n)$, and

$$w_{i+1} = f(p)\log w_i, \tag{6}$$

if $i \geq 1$ and $w_i \geq 2\,f(p)\log w_i$. Any further phase would just consist of a single iteration. In the remaining of this section, we only consider phases i for which Equation (6) is true, and we call them the *valid phases*.

By Lemma 16, if the weak invariant (♣) holds for x and window size w_{i-1}, the probability that it still holds for window size w_i is at least $1 - 42/w_{i-1}^2$. Similarly to the analysis in the Section 4.2, we get that a valid phase $i \geq 1$ contributes to the expected dislocation of x by

$$42/w_{i-1}^2 \cdot 9w_{i-1} = 378/w_{i-1}. \tag{7}$$

If we stop our analysis after c valid phases, then by (7) and Lemma 8, the expected dislocation of any element x is at most

$$\sum_{i=0}^{c-1} 378/w_i + 9w_c \leq 378/w_c + 9w_c. \tag{8}$$

The inequality holds since $\frac{w_{i-1}}{w_i} \leq 2$ for $1 < i < c$. We next define c such that phase c is valid and w_c only depends on $f(p)$. The term $\frac{w_{i-1}}{f(p)\log w_{i-1}} \geq 2$ holds for every valid phase i and decreases with increasing i. For instance for $w = 6f(p)\log f(p)$:

$$\frac{w}{f(p)\log w} = \frac{6\,f(p)\log f(p)}{f(p)\log(6\,f(p)\log f(p))} \geq \frac{6\log f(p)}{3\log f(p)} \geq 2.$$

Therefore, if we choose c such that $w_{c-1} \geq 6\,f(p)\log f(p) > w_c$, we can use that $f(p) \geq 6$ and upper bound w_c by

$$w_c = f(p)\log w_{c-1} \geq f(p)\log(6\,f(p)\log f(p)) \geq 6\log(36\log 6) \geq 39. \tag{9}$$

Equations (8) and (9) and Lemma 8 imply the following:

▶ **Lemma 17.** *The expected dislocation of each element x after running* WINDOW SORT *is at most* $378/w_c + 9w_c < 10 + 9w_c \leq 10w_c \leq 60\,f(p)\log f(p)$.

This immediately implies Theorem 14.

5 Extension

The reason why we require the error probability p to be smaller than $1/32$ is to analyze the probability that at most $w/4$ errors occur in $8w$ comparisons, for $w \geq 1$. This bound on the number of errors appears since we halve the window size in every iteration. If we let the window size shrink by another rate $1/2 < \alpha < 1$, the limit of p will also change:

First, the running time of the adapted WINDOW SORT will become $O(\frac{1}{1-\alpha}n^2)$. Second, for any permutation σ and window size w, in order to maintain condition $(*)$ for an element x, its computed position should differ from x by at most αw, and thus $computed\_rank(x)$ should differ from x by at most $\alpha w/2$.

Our new issue is thus the probability that at most $\alpha w/2$ errors occur in $8w$ comparisons: Since the expected number of errors is $8wp$, we have $\frac{\alpha w}{2} = \frac{\alpha}{16p} \cdot 8wp = (1 + \frac{\alpha - 16p}{16p}) \cdot 8wp$, and by the reasoning of Lemma 12, we have $\frac{\alpha - 16p}{16p} > 0$, thus $p < \alpha/16$. (Note that $f(p)$ should change accordingly.)

Finally, the number of windows for the weak invariant should also change accordingly. Let m be the number of windows that matter for the weak invariant ($m = 12$ when $\alpha = 1/2$). According to the analysis in Section 4.1, we have $m - 6 \geq \alpha m$, implying that $m \geq \frac{6}{1-\alpha}$. Of course, the constant inside the linear expected total dislocation will also change accordingly.

▶ **Theorem 18.** *For an error probability $p < \alpha/16$, where $1/2 < \alpha < 1$, modified* WINDOW SORT *on n elements takes $O(\frac{1}{1-\alpha}n^2)$ time, has maximum dislocation $9\,g(p,\alpha)\log n$ with probability $1 - 1/n$, and expected total dislocation $n \cdot (9 + \frac{2}{1-\alpha}) \cdot 6\,g(p,\alpha)\log g(p,\alpha)$, where*

$$
g(p,\alpha) = \begin{cases} \frac{100p}{(\alpha - 16p)^2} & for \quad \alpha/32 < p < \alpha/16 \ , \\[2mm] \frac{4}{(\ln(\alpha/16p)) - (\alpha - 16p)} & for \quad \alpha/96 < p \leq \alpha/32 \ , \\[2mm] 6 & for \quad p \leq \alpha/96 \ . \end{cases}
$$

6 A lower bound on the maximum dislocation

In this section we prove a lower bound on both the maximum and the average dislocation that can be achieved w.h.p. by any sorting algorithm.

The following lemma – whose proof is omitted – is a key ingredient in our lower bounds:

▶ **Lemma 19.** *Let $x, y \in S$ with $x < y$. Let A be any (possibly randomized) algorithm. On a random instance, the probability that A returns a permutation in which elements x and y appear the wrong relative order is at least $\frac{1}{2}\left(\frac{p}{1-p}\right)^{2(y-x)-1}$.*

As a first consequence of the previous lemma, we obtain the following:

▶ **Theorem 20.** *No (possibly randomized) algorithm can achieve maximum dislocation $o(\log n)$ with high probability.*

Proof. By Lemma 19, any algorithm, when invoked on a random instance, must return a permutation ρ in which elements 1 and $h = \lfloor \frac{\log n}{2\log 1-p/p} \rfloor$ appear in the wrong order with a probability larger than $\frac{1}{n}$. When this happens, at least one of the following two conditions holds: (i) the position of element 1 in ρ is at least $\lceil \frac{h}{2} \rceil$; or (ii) the position of element h in ρ is at most $\lfloor \frac{h}{2} \rfloor$. In any case, the maximum dislocation must be at least $\frac{h}{2} - 1 = \Omega(\log n)$. ◀

Finally, we are also able to prove a lower bound to the total dislocation (proof omitted due to space limitations).

▶ **Theorem 21.** *No (possibly randomized) algorithm can achieve expected total dislocation $o(n)$.*

─── **References** ───

1   Laurent Alonso, Philippe Chassaing, Florent Gillet, Svante Janson, Edward M Reingold, and René Schott. Quicksort with unreliable comparisons: a probabilistic analysis. *Combinatorics, Probability and Computing*, 13(4-5):419–449, 2004.

2   Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms*, pages 268–276, 2008. `arXiv:0707.1051`.

3   Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013.

4   Don Coppersmith, Lisa K. Fleischer, and Atri Rurda. Ordering by weighted number of wins gives a good ranking for weighted tournaments. *ACM Trans. Algorithms*, 6(3):55:1–55:13, July 2010. `doi:10.1145/1798596.1798608`.

5   Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

6   Peter Damaschke. The solution space of sorting with recurring comparison faults. In *Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, pages 397–408, 2016.

7   Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. `doi:10.1137/S0097539791195877`.

8   Tomáš Gavenčiak, Barbara Geissmann, and Johannes Lengler. Sorting by swaps with noisy comparisons. In *To appear in: Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '17, 2017.

9   B. Geissmann and P. Penna. Sort well with energy-constrained comparisons. *ArXiv e-prints*, October 2016. `arXiv:1610.09223`.

10   Petros Hadjicostas and KB Lakshmanan. Recursive merge sort with erroneous comparisons. *Discrete Applied Mathematics*, 159(14):1398–1417, 2011.

11   Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 881–890, 2007.

12   Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proceedings of the19th Annual European Symposium on Algorithm*, pages 736—-747, 2011.

13   Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

14   Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

15   R. L. Graham Persi Diaconis. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):262–268, 1977.