




On the Average Case of MergeInsertion

Florian Stober¹ · Armin Weiß¹ 

Published online: 30 June 2020
© The Author(s) 2020

Abstract

MergeInsertion, also known as the Ford-Johnson algorithm, is a sorting algorithm which, up to today, for many input sizes achieves the best known upper bound on the number of comparisons. Indeed, it gets extremely close to the information-theoretic lower bound. While the worst-case behavior is well understood, only little is known about the average case. This work takes a closer look at the average case behavior. In particular, we establish an upper bound of $n \log n - 1.4005n + o(n)$ comparisons. We also give an exact description of the probability distribution of the length of the chain a given element is inserted into and use it to approximate the average number of comparisons numerically. Moreover, we compute the exact average number of comparisons for n up to 148. Furthermore, we experimentally explore the impact of different decision trees for binary insertion. To conclude, we conduct experiments showing that a slightly different insertion order leads to a better average case and we compare the algorithm to Manacher's combination of merging and MergeInsertion as well as to the recent combined algorithm with (1,2)-Insertionsort by Iwama and Teruyama.

Keywords MergeInsertion · Minimum-comparison sort · Average case analysis.

1 Introduction

Sorting a set of elements is an important operation frequently performed by many computer programs. Consequently there exist a variety of algorithms for sorting, each of which comes with its own advantages and disadvantages.

This article belongs to the Topical Collection: *Special Issue on International Workshop on Combinatorial Algorithms (IWOCA 2019)*

Guest Editors: Charles Colbourn, Roberto Grossi, Nadia Pisanti

The second author has been supported by the German Research Foundation (DFG) under grant DI 435/7-1.

✉ Armin Weiß
armin.weiss@fmi.uni-stuttgart.de

¹ Universität Stuttgart, FMI, Universitätsstr. 38, D-70569, Stuttgart, Germany

Here we focus on comparison based sorting and study a specific sorting algorithm known as MergeInsertion. It was discovered by Ford and Johnson in 1959 [5]. Before D. E. Knuth coined the term MergeInsertion in his study of the algorithm in his book “The Art of Computer Programming, Volume 3: Sorting and Searching” [8], it was known only as Ford-Johnson Algorithm, named after its creators. The one outstanding property of MergeInsertion is that the number of comparisons it requires is close to the information-theoretic lower bound of $\log(n!) \approx n \log n - 1.4427n$ (for sorting n elements). This sets it apart from many other sorting algorithms. MergeInsertion can be described in three steps: first pairs of elements are compared; in the second step the larger elements are sorted recursively; as a last step the elements belonging to the smaller half are inserted into the already sorted larger half using binary insertion.

In the worst case the number of comparisons of MergeInsertion is quite well understood [8] – it is $n \log n + b(n) \cdot n + o(n)$ where $b(n)$ oscillates between -1.415 and -1.3289 . Moreover, for many n MergeInsertion is proved to be the optimal algorithm in the worst case (in particular, for $n \leq 15$ [10, 11]). However, there are also n where it is not optimal [2, 9]. One reason for this is the oscillating linear term in the number of comparisons, which allowed Manacher [9] to show that for certain n it is more efficient to split the input into two parts, sort both parts with MergeInsertion, and then merge the two parts into one array.

Regarding the average case (with respect to a uniform distribution over all input permutations) not much is known: in [8] Knuth calculated the number of comparisons required on average for $n \in \{1, \dots, 8\}$; an upper bound of $n \log n - 1.3999n + o(n)$ has been given in [3, Theorem 6] (the proof can be found in [4, Theorem 6.5]). Most recently, Iwama and Teruyama [7] showed that in the average case MergeInsertion can be improved by combining it with their (1,2)-Insertion algorithm resulting in an upper bound of $n \log n - 1.4106n + O(\log n)$. This reduces the gap to the lower bound by around 25%. It is a fundamental open problem how close one can get to the information-theoretic lower bound of $n \log n - 1.4427n$ (see e. g. [7, 12]).

The goal of this work is to study the number of comparisons required in the average case. Usually, the average case of a sorting algorithm refers to a uniform distribution of all permutations of distinct elements. Here, we follow this convention. In particular, we analyze the insertion step of MergeInsertion in greater detail. In general, MergeInsertion achieves its good performance by inserting elements in a specific order that in the worst case causes each element to be inserted into a sorted list of $2^k - 1$ elements (thus, using exactly k comparisons). When looking at the average case elements are often inserted into less than $2^k - 1$ elements which is slightly cheaper. By calculating those small savings we seek to achieve our goal of a better upper bound on the average case. Our results can be summarized as follows:

- We derive an exact formula for the probability distribution into how many elements a given element is inserted (Theorem 2). This is the crucial first step in order to obtain better bounds for the average case of MergeInsertion.
- We experimentally examine different decision trees for binary insertion. We obtain the best result when assigning shorter decision paths to positions located further to the left (Section 3.1).

- We use Theorem 2 in order to compute quite precise numerical estimates for the average number of comparisons for n up to roughly 15000 (Section 4.1).
- We compute the exact average number of comparisons for n up to 148 – thus, going much further than [8] (Section 4.2).
- We improve the bound of [3, 4] to $n \log n - 1.4005n + o(n)$ (Theorem 3). This partially answers a conjecture from [12] which asks for an in-place algorithm with $n \log n - 1.4n$ comparisons on average and $n \log n - 1.3n$ comparisons in the worst case. Although MergeInsertion is not in-place, the techniques from [3, 4] or [12] can be used to make it so.
- We evaluate a slightly different insertion order decreasing the gap between the lower bound and the average number of comparisons of MergeInsertion by roughly 30% for $n \approx 2^k/3$ (Section 5.2).
- Our experiments show that splitting the input into two parts, sorting them separately, and then merging the sorted parts – as proposed by [9] to improve the worst case –, also leads to an improvement in the average-case for certain input sizes (Section 5.3).
- We compare MergeInsertion to the recent combination by Iwama and Teruyama [7] showing that, in fact, their combined algorithm is still better than the analysis and with the different insertion order can be further improved (Section 5.4).

2 Preliminaries

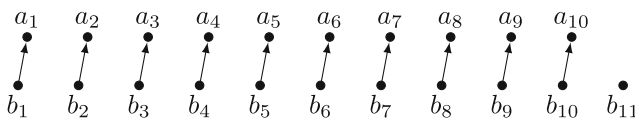
Throughout, we assume that the input consists of n distinct elements. The average case complexity is the mean number of comparisons over all input permutations of n elements.

We use the notation *falling factorial* $x^n = x(x - 1) \cdots (x - n + 1) = \prod_{i=0}^{n-1} (x - i)$ and *rising factorial* $x^{\bar{n}} = x(x + 1) \cdots (x + n - 1) = \prod_{i=0}^{n-1} (x + i)$ for $x, n \in \mathbb{N}$. Notice that we have $x^n = \frac{x!}{(x-n)!}$ for $x \geq n$ and $x^{\bar{n}} = \frac{(x+n-1)!}{(x-1)!}$.

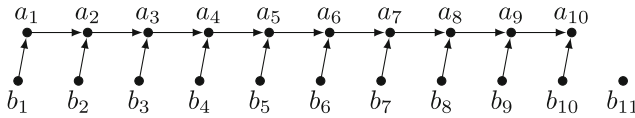
2.1 Description of MergeInsertion

The MergeInsertion algorithm consists of three phases: pairwise comparison, recursion, and insertion. Accompanying the explanations we give an example where $n = 21$. We call such a set of relations between individual elements a configuration.

1. **Pairwise comparison.** The elements are grouped into $\lfloor \frac{n}{2} \rfloor$ pairs. Each pair is sorted using one comparison. After that, the elements are called a_1 to $a_{\lfloor \frac{n}{2} \rfloor}$ and b_1 to $b_{\lceil \frac{n}{2} \rceil}$ with $a_i > b_i$ for all $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$.



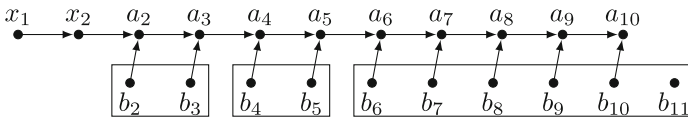
- Recursion.** The $\lfloor \frac{n}{2} \rfloor$ larger elements, i. e., a_1 to $a_{\lfloor \frac{n}{2} \rfloor}$ are sorted recursively. Then all elements (the $\lfloor \frac{n}{2} \rfloor$ larger ones as well as the corresponding smaller ones) are renamed accordingly such that $a_i < a_{i+1}$ and $a_i > b_i$ still holds.



- Insertion.** The $\lceil \frac{n}{2} \rceil$ small elements, i. e., the b_i , are inserted into the main chain using binary insertion. The term “main chain” describes the set of elements containing a_1, \dots, a_{t_k} as well as the b_i that have already been inserted.

The elements are inserted in batches starting with b_3, b_2 . In the k -th batch the elements $b_{t_k}, b_{t_k-1}, \dots, b_{t_k-1+1}$ where $t_k = \frac{2^{k+1} + (-1)^k}{3}$ are inserted in that order. Elements b_j where $j > \lceil \frac{n}{2} \rceil$ (which do not exist) are skipped. Note that technically b_1 is the first batch; but inserting b_1 does not need any comparison.

Because of the insertion order, every element b_i which is part of the k -th batch is inserted into at most $2^k - 1$ elements; thus, it can be inserted by binary insertion using at most k comparisons.



Regarding the average number of comparisons $F(n)$ we make the following observations: the first step always requires $\lfloor \frac{n}{2} \rfloor$ comparisons. The recursion step does not do any comparisons by itself but depends on the other steps. The average number of comparisons $G(n)$ required in the insertion step is not obvious. It will be studied closer in following chapters. Following [8], we obtain the recurrence (which is the same as for the worst-case number of comparisons)

$$F(n) = \lfloor \frac{n}{2} \rfloor + F\left(\lfloor \frac{n}{2} \rfloor\right) + G\left(\lceil \frac{n}{2} \rceil\right). \tag{1}$$

3 Average Case Analysis of the Insertion Step

The aim of this section is to compute the probability distributions where the elements end up in the insertion step and into how many elements they are inserted. For this we focus on the insertion of one batch of elements, i. e., the elements b_{t_k} down to b_{t_k-1+1} . We assume that all elements of previous batches, i. e., b_1 to b_{t_k-1} , have already been inserted and together with the corresponding a_i they constitute the main chain and have been renamed to x_1 to x_{2t_k-1} such that $x_i < x_{i+1}$. The situation is shown in Fig. 1.

We will look at the element b_{t_k+i} and want to answer the following questions: what is the probability of it being inserted between x_j and x_{j+1} ? And what is the probability of it being inserted into a specific number of elements?

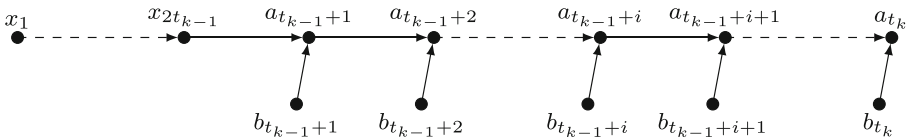


Fig. 1 Configuration where a single batch of elements remains to be inserted

We can ignore batches that are inserted after the batch we are looking at since those do not affect the probabilities we wish to obtain.

First we define a probability space for the process of inserting one batch of elements: let Ω_k be the set of all possible outcomes (i. e., linear extensions) when sorting the partially ordered elements shown in Fig. 1 by inserting b_{t_k} to $b_{t_{k-1}+1}$. Each $\omega \in \Omega_k$ can be viewed as a function that maps an element e to its final position, i. e., $\omega(e) \in \{1, 2, \dots, 2t_k\}$. While the algorithm mandates a specific order for inserting the elements $b_{t_{k-1}+1}$ to b_{t_k} during the insertion step, using a different order does not change the outcome, i. e., the elements are still sorted correctly. We will use this observation frequently in order to simplify calculating the likelihood of relations between individual elements.

Let us look at where an element will end up after it has been inserted. Unfortunately, not all positions are equally likely. For this purpose we define the random variable X_i as follows. To simplify notation we define $x_{t_{k-1}+j} := a_j$ for $t_{k-1} < j \leq t_k$ (hence, the main chain consists of x_1, \dots, x_{2^k}).

$$X_i : \omega \mapsto \begin{cases} 0 & \text{if } \omega(b_{t_{k-1}+i}) < \omega(x_1) \\ j & \text{if } \omega(x_j) < \omega(b_{t_{k-1}+i}) < \omega(x_{j+1}) \text{ for } j \in \{1, \dots, 2^k - 2\} \\ 2^k - 1 & \text{if } \omega(x_{2^k-1}) < \omega(b_{t_{k-1}+i}). \end{cases}$$

Be aware that X_i actually also depends on k , but for a simpler notation we omit a second index k . We are interested in the probabilities $P(X_i = j)$. These values follow a simple pattern. For $k = 4$ they are given in Table 1 and for $k = 6$ and $i = t_k - t_{k-1}$ depicted in Fig. 2.

Theorem 1 *The probability of $b_{t_{k-1}+i}$ being inserted between x_j and x_{j+1} is given by*

$$P(X_i = j) = \begin{cases} 2^{2i-2} \left(\frac{(t_{k-1}+i-1)!}{(t_{k-1})!} \right)^2 \frac{(2t_{k-1})!}{(2t_{k-1}+2i-1)!} & \text{if } 0 \leq j \leq 2t_{k-1} \\ 2^{4t_{k-1}-2j+2i-2} \left(\frac{(t_{k-1}+i-1)!}{(j-t_{k-1})!} \right)^2 \frac{(2j-2t_{k-1})!}{(2t_{k-1}+2i-1)!} & \text{if } 2t_{k-1} < j < 2t_{k-1}+i \\ 0 & \text{otherwise} \end{cases}$$

Proof For an arbitrary k we can calculate the probabilities $P(X_i = j)$ by induction on i . The base cases for the induction are that $i = 1$ or $i = j - 2t_{k-1}$. We start with $P(X_1 = j)$. This corresponds to the insertion of $b_{t_{k-1}+1}$ into $x_1, \dots, x_{2t_{k-1}}$. The probability of all positions is uniformly distributed, so $P(X_1 = j) = \frac{1}{2t_{k-1}+1}$ for $0 \leq j \leq 2t_{k-1}$.

Table 1 Values of $P(X_i = j)$ for $k = 4$

i	1	2	3	4	5	6
$P(X_i = 0)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 1)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 2)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 3)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 4)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 5)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 6)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 7)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 8)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 9)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 10)$	$\frac{1}{11}$	$\frac{1}{11} \cdot \frac{12}{13}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{11} \cdot \frac{12}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 11)$	0	$\frac{1}{13}$	$\frac{1}{13} \cdot \frac{14}{15}$	$\frac{1}{13} \cdot \frac{14}{15} \cdot \frac{16}{17}$	$\frac{1}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{13} \cdot \frac{14}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 12)$	0	0	$\frac{1}{15}$	$\frac{1}{15} \cdot \frac{16}{17}$	$\frac{1}{15} \cdot \frac{16}{17} \cdot \frac{18}{19}$	$\frac{1}{15} \cdot \frac{16}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 13)$	0	0	0	$\frac{1}{17}$	$\frac{1}{17} \cdot \frac{18}{19}$	$\frac{1}{17} \cdot \frac{18}{19} \cdot \frac{20}{21}$
$P(X_i = 14)$	0	0	0	0	$\frac{1}{19}$	$\frac{1}{19} \cdot \frac{20}{21}$
$P(X_i = 15)$	0	0	0	0	0	$\frac{1}{21}$

Now let $i > 1$ and $j = i + 2t_{k-1}$. As mentioned above, we can compute this probability by looking at a different insertion order: we assume that the elements $b_{t_{k-1}+1}, \dots, b_{t_{k-1}+i-1}$ have been already inserted into the main chain and we want to insert $b_{t_{k-1}+i}$. Now we have a uniform distribution over all positions where $b_{t_{k-1}+i}$ can end up. There are $2t_{k-1} + 2i - 2$ elements known to be smaller than $a_{t_{k-1}+i}$. These

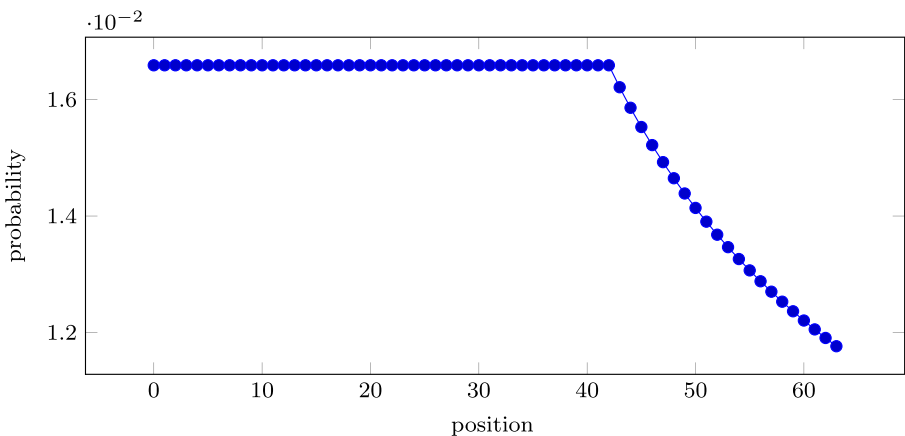


Fig. 2 Probability distribution of $X_{t_k - t_{k-1}}$ (when inserting b_{t_k}) for $k = 6$

are $x_1, \dots, x_{2t_{k-1}}$ and $a_{t_{k-1}+1}, \dots, a_{t_{k-1}+i-1}$ as well as the corresponding b 's. The number of elements known to be smaller than $a_{t_{k-1}+i-1}$ is one less: $2t_{k-1} + 2i - 3$. As a result the probability that $b_{t_{k-1}+i}$ is inserted between $a_{t_{k-1}+i-1}$ and $a_{t_{k-1}+i}$ is $P(X_i = 2t_{k-1} + i - 1) = \frac{1}{2t_{k-1}+2i-1}$. This constitutes the induction base for the case $2t_{k-1} < j$.

For $i > 1$ and $j < i + 2t_{k-1}$ we can express $P(X_i = j)$ in terms of $P(X_{i-1} = j)$. We know already that $P(X_i = 2t_{k-1} + i - 1) = \frac{1}{2t_{k-1}+2i-1}$. The probability that $b_{t_{k-1}+i}$ ends up in one of the positions in front of $a_{t_{k-1}+i-1}$ is consequently $P(0 \leq X_i < 2t_{k-1} + i - 1) = \frac{2t_{k-1}+2i-2}{2t_{k-1}+2i-1}$. If we know that $b_{t_{k-1}+i}$ is inserted into one of those other positions, then it is inserted into exactly the same elements as $b_{t_{k-1}+i-1}$ (i. e., the situation is completely symmetric in $b_{t_{k-1}+i-1}$ and $b_{t_{k-1}+i}$). Thus, we have $P(X_i = j \mid X_i < 2t_{k-1} + i - 1) = P(X_{i-1} = j)$ and so we can write $P(X_i = j) = \frac{2t_{k-1}+2i-2}{2t_{k-1}+2i-1} P(X_{i-1} = j)$. This leads to

$$P(X_i = j) = \begin{cases} \left(\prod_{l=1}^{i-1} 2t_{k-1} + 2l \right) \cdot \left(\prod_{l=1}^i 2t_{k-1} + 2l - 1 \right)^{-1} & \text{if } 0 \leq j \leq 2t_{k-1} \\ \left(\prod_{l=j-2t_{k-1}+1}^{i-1} 2t_{k-1} + 2l \right) \cdot \left(\prod_{l=j-2t_{k-1}+1}^i 2t_{k-1} + 2l - 1 \right)^{-1} & \text{if } 2t_{k-1} < j < 2t_{k-1} + i \\ 0 & \text{otherwise.} \end{cases}$$

By simplifying this equation, we obtain Theorem 1. □

Next, our aim is to compute the probability that b_i is inserted into a particular number of elements. This is of special interest because the difference between average and worst case comes from the fact that sometimes we insert into less than $2^k - 1$ elements. For that purpose we define the random variable Y_i :

$$Y_i : \omega \mapsto \left| \left\{ v \in \{x_1, \dots, x_{2^k}\} \cup \{b_{t_{k-1}+i+1}, \dots, b_{t_k}\} \mid \omega(v) < \omega(a_{t_{k-1}+i}) \right\} \right|.$$

Thus, Y_i describes the number of elements into which $b_{t_{k-1}+i}$ is inserted (for $1 \leq i \leq t_k - t_{k-1}$), i. e., the length of the main chain up to $a_{t_{k-1}+i}$ at this point of time. As for the X_i , we omit an additional index k to keep notation simple. The elements in the main chain when inserting $b_{t_{k-1}+i}$ are x_1 to $x_{2t_{k-1}+i-1}$ and those elements out of $b_{t_{k-1}+i+1}, \dots, b_{t_k}$ which have been inserted in front of $a_{t_{k-1}+i}$ (which is $x_{2t_{k-1}+i}$). Let us start with an easy observation which we will use later.

Lemma 1 *For $q < u$ and arbitrary m , we have $P(Y_q < m) \geq P(Y_u < m)$.*

Proof Consider the situation $Y_u < m$ meaning that $b_{t_{k-1}+u}$ is inserted into less than m elements. We distinguish two cases: either $b_{t_{k-1}+u}$ is inserted in front of $a_{t_{k-1}+u-1}$ or between $a_{t_{k-1}+u-1}$ and $a_{t_{k-1}+u}$. In the first case, we have $Y_{u-1} = Y_u$, in the second case we have $Y_{u-1} = Y_u - 1$; thus, in any case $Y_{u-1} < m$. Therefore, we have $P(Y_{u-1} < m) \geq P(Y_u < m)$ and the lemma follows by induction. □

Theorem 2 For $1 \leq i \leq t_k - t_{k-1}$ and $2t_{k-1} + i - 1 \leq j \leq 2^k - 1$ the probability $P(Y_i = j)$, that $b_{t_{k-1}+i}$ is inserted into j elements is given by

$$P(Y_i = j) = 2^{j-2t_{k-1}-i+1} \frac{(2t_k - i - j - 1)!}{(j - 2t_{k-1} - i + 1)!(2^k - j - 1)!} \frac{(i + j)!}{(2t_k - 1)!} \frac{(t_k - 1)!}{(t_{k-1} + i - 1)!}.$$

Proof For the proof we introduce a set of random variables $\tilde{Y}_{i,q}$ counting the elements in $\{b_{t_{k-1}+i+1}, \dots, b_{t_{k-1}+i+q}\}$ which are inserted in front of $a_{t_{k-1}+i}$ into the main chain (see Fig. 3):

$$\tilde{Y}_{i,q} : \omega \mapsto |\{v \in \{b_{t_{k-1}+i+1}, \dots, b_{t_{k-1}+i+q}\} \mid \omega(v) < \omega(a_{t_{k-1}+i})\}|.$$

By setting $q = t_k - t_{k-1} - i$, we obtain

$$Y_i = \tilde{Y}_{i,t_k-t_{k-1}-i} + 2t_{k-1} + i - 1. \tag{2}$$

Clearly we have $P(\tilde{Y}_{i,0} = j) = 1$ if $j = 0$ and $P(\tilde{Y}_{i,0} = j) = 0$ otherwise. For $q > 0$ there are two possibilities:

1. $\tilde{Y}_{i,q-1} = j - 1$ and $X_{i+q} < 2t_{k-1} + i$: out of $\{b_{t_{k-1}+i+1}, \dots, b_{t_{k-1}+i+q-1}\}$ there have been $j - 1$ elements inserted before $a_{t_{k-1}+i}$ and $b_{t_{k-1}+i+q}$ is inserted before $a_{t_{k-1}+i}$.
2. $\tilde{Y}_{i,q-1} = j$ and $X_{i+q} \geq 2t_{k-1} + i$: out of $\{b_{t_{k-1}+i+1}, \dots, b_{t_{k-1}+i+q-1}\}$ there have been j elements inserted before $a_{t_{k-1}+i}$ and $b_{t_{k-1}+i+q}$ is inserted after $a_{t_{k-1}+i}$.

From these we can calculate $P(\tilde{Y}_{i,q} = j)$ via

$$P(\tilde{Y}_{i,q} = j) = P(X_{i+q} < 2t_{k-1} + i \wedge \tilde{Y}_{i,q-1} = j - 1) + P(X_{i+q} \geq 2t_{k-1} + i \wedge \tilde{Y}_{i,q-1} = j).$$

Using Bayes' theorem we obtain the following recurrence:

$$P(\tilde{Y}_{i,q} = j) = P(X_{i+q} < 2t_{k-1} + i \mid \tilde{Y}_{i,q-1} = j - 1) \cdot P(\tilde{Y}_{i,q-1} = j - 1) + P(X_{i+q} \geq 2t_{k-1} + i \mid \tilde{Y}_{i,q-1} = j) \cdot P(\tilde{Y}_{i,q-1} = j)$$

The probability $P(X_{i+q} < 2t_{k-1} + i \mid \tilde{Y}_{i,q-1} = j - 1)$ can be obtained by looking at Fig. 3 and counting elements. As we explained before (Theorem 1), we can compute this probability by assuming a different insertion order: if all the b_ℓ for $\ell < t_{k-1} + i + q$ are already inserted into the main chain, then we have a uniform distribution for the position where $b_{t_{k-1}+i+q}$ is inserted. In this situation, $b_{t_{k-1}+i+q}$ is inserted into $2t_{k-1} + 2i + 2q - 2$ elements (namely a_ℓ and b_ℓ for $1 \leq \ell < t_{k-1} + i + q$). On the other

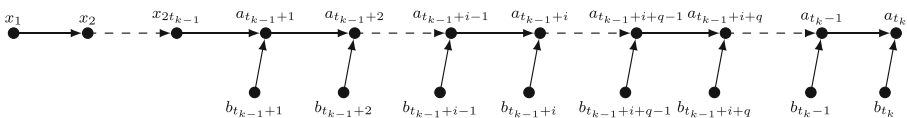


Fig. 3 Configuration where one batch of $t_k - t_{k-1}$ elements remains to be inserted. The elements $b_{t_{k-1}+i}$ and $b_{t_{k-1}+i+q}$ are drawn

hand, under the condition $\tilde{Y}_{i,q-1} = j - 1$, we know that $2t_{k-1} + 2i + j - 2$ of these are smaller than $a_{t_{k-1}+i}$ (namely, the a_ℓ and b_ℓ with $1 \leq \ell < t_{k-1} + i$, $b_{t_{k-1}+i}$, and $j - 1$ elements out of $\{b_{t_{k-1}+i+1}, \dots, b_{t_{k-1}+i+q-1}\}$). Thus, we obtain the probability $\frac{2t_{k-1}+2i+j-1}{2t_{k-1}+2i+2q-1}$. We can calculate $P(X_{i+q} \geq 2t_{k-1} + i \mid \tilde{Y}_{i,q-1} = j)$ similarly leading to

$$P(\tilde{Y}_{i,q} = j) = \frac{2t_{k-1} + 2i + j - 1}{2t_{k-1} + 2i + 2q - 1} \cdot P(\tilde{Y}_{i,q-1} = j - 1) + \frac{2q - j - 1}{2t_{k-1} + 2i + 2q - 1} \cdot P(\tilde{Y}_{i,q-1} = j). \tag{3}$$

Resolving the recurrence leads to

$$P(\tilde{Y}_{i,q} = j) = \frac{(2q - j)^q}{2^{q-j} j!} \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j)^{2q-j}}. \tag{4}$$

By substituting $q := t_k - t_{k-1} - i$ and $j := j + 2t_{k-1} + i - 1$ according to (2), we obtain the closed form for $P(Y_i = j)$ shown in Theorem 2.

In order to prove (4), notice that in the case $j = 0$ (respectively $j = q$), we have $P(\tilde{Y}_{i,q-1} = j - 1) = 0$ (respectively $P(\tilde{Y}_{i,q-1} = j) = 0$). Hence, in these cases we have only one term to consider and it is an easy exercise to show (4) by induction. For the case $0 < j < q$, we also proceed by induction ¹:

$$\begin{aligned} &P(\tilde{Y}_{i,q} = j) \\ &\stackrel{(3)+\text{induction}}{=} \frac{2t_{k-1} + 2i + j - 1}{2t_{k-1} + 2i + 2q - 1} \cdot \frac{(2q - j - 1)^{q-1}}{2^{q-j} (j - 1)!} \cdot \frac{2^{q-1} (t_{k-1} + i)^{\overline{q-1}}}{(2t_{k-1} + 2i + j - 1)^{2q-j-1}} \\ &\quad + \frac{2q - j - 1}{2t_{k-1} + 2i + 2q - 1} \cdot \frac{(2q - j - 2)^{q-1}}{2^{q-j-1} j!} \cdot \frac{2^{q-1} (t_{k-1} + i)^{\overline{q-1}}}{(2t_{k-1} + 2i + j)^{2q-j-2}} \\ &= (2t_{k-1} + 2i + j - 1) \cdot \frac{(2q - j - 1)^{q-1}}{2^{q-j} (j - 1)!} \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j - 1)^{2q-j+1}} \\ &\quad + (2q - j - 1) \cdot \frac{(2q - j - 2)^{q-1}}{2^{q-j-1} j!} \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j)^{2q-j}} \\ &= \left(\frac{(2q - j - 1)^{q-1}}{2^{q-j} (j - 1)!} + \frac{(q - j)(2q - j - 1)^{q-1}}{2^{q-j-1} j!} \right) \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j)^{2q-j}} \\ &= \left(\frac{j}{2q - j} + \frac{2(q - j)}{2q - j} \right) \cdot \frac{(2q - j)^{q-1}}{2^{q-j} j!} \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j)^{2q-j}} \\ &= \frac{(2q - j)^{q-1}}{2^{q-j} j!} \cdot \frac{2^q (t_{k-1} + i)^{\bar{q}}}{(2t_{k-1} + 2i + j)^{2q-j}}. \end{aligned}$$

□

¹For a automated verification of the inductive step we computed the quotient of the first and the last line in the following computation using the Mathematica query: FullSimplify[((2^j/j!) ((2 q - j)!/(q - j)!) ((2 t + 2 i + j - 1)!/(2 t + 2 i + 2 q - 1)!) ((t + i - 1 + q)!/(t + i - 1)!)) / (((2 t + 2 i + j - 1)/(2 t + 2 i + 2 q - 1)) ((2 q - j - 1)!/(2^(q - j) (j - 1)! (q - j)!)) 2^(q - 1) ((2 t + 2 i + j - 2)!/(2 t + 2 i + 2 q - 3)!) ((t + i + q - 2)!/(t + i - 1)!) + ((2 q - j - 1)/(2 t + 2 i + 2 q - 1)) ((2 q - j - 2)!/(2^(q - j - 1) (j - 1)! (q - j - 1)!)) 2^(q - 1) ((2 t + 2 i + j - 1)/(2 t + 2 i + 2 q - 3)!) ((t + i + q - 2)!/(t + i - 1)!))]

Fig. 4 Probability distribution of Y_i

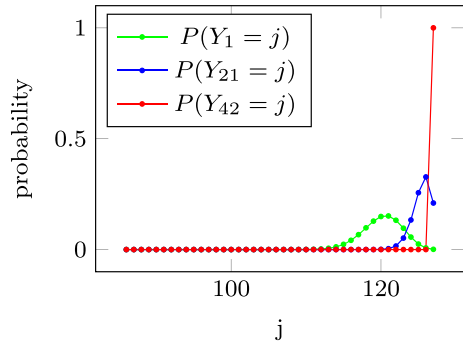


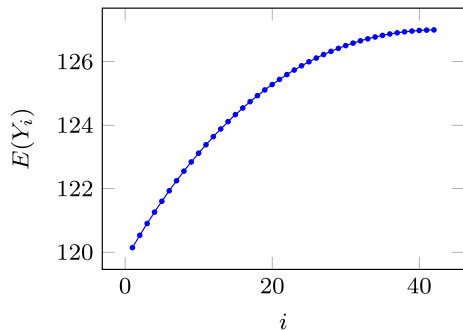
Figure 4 shows the probability distribution for Y_1 , Y_{21} and Y_{42} where $k = 7$. Y_{42} corresponds to the insertion of b_{i_k} which is the first element of the batch and thus is always inserted into $2^k - 1$ elements, in this case 127. Y_1 corresponds to the insertion of $b_{i_{k-1}+1}$ (the last element of the batch). In addition to those three probability distributions Fig 5 shows the mean of all Y_i for $k = 7$. Looking at the values from above, for $i = 42$ the mean (non-surprisingly) is 127 and for $i = 1$ it is close to 120. In between we have what appears to be an exponential curve.

3.1 Binary Insertion and Different Decision Trees

The Binary Insertion step is an important part of MergeInsertion. In the average case many elements are inserted in less than $2^k - 1$ (which is the worst case). This leads to ambiguous decision trees where at some positions inserting an element requires only $k - 1$ instead of k comparisons. Since not all positions are equally likely (positions on the left have a slightly higher probability, see Fig. 2 for an example), this results in different average insertion costs. We compare four different strategies all satisfying that the corresponding decision trees have their leaves distributed across at most two layers. For an example with five elements see Fig. 6.

First there are the center-left and center-right strategies (the standard options for binary insertion): they compare the element to be inserted with the middle element, rounding down(up) in case of an odd number. The left

Fig. 5 Mean of Y_i for different i . $k = 7$



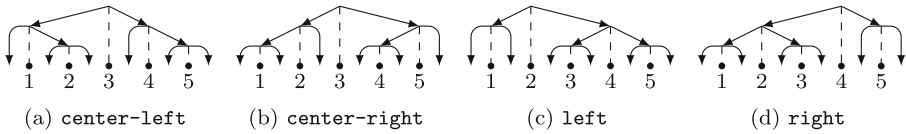


Fig. 6 Different strategies for binary insertion

strategy chooses the element to compare with in a way such that the positions where only $k - 1$ comparisons are required are at the very left. The right strategy is similar, here the positions where one can insert with just $k - 1$ comparisons are at the right. To summarize, the element to compare with is

$$\begin{aligned} \left\lfloor \frac{n+1}{2} \right\rfloor & \quad \text{strategy center-left} \\ \left\lceil \frac{n+1}{2} \right\rceil & \quad \text{strategy center-right} \\ \max\{n - 2^k + 1, 2^{k-1}\} & \quad \text{strategy left} \\ \min\{2^k, n - 2^{k-1} + 1\} & \quad \text{strategy right} \end{aligned}$$

where $k = \lfloor \log n \rfloor$. Notice that the left strategy is also used in [7], where it is called *right-hand-binary-search*. Figure 7 shows experimental results comparing the different strategies for binary insertion regarding their effect on the average-case of MergeInsertion. As we can see the left strategy performs the best, closely followed by center-left and center-right. right performs the worst. The left strategy performing best is no surprise since the probability that an element is inserted into one of the left positions is higher than it being inserted to the right. Therefore, in all further experiments we use the left strategy.

Notice that the oscillating behavior in Fig. 7 can be explained as follows: Binary insertion is most efficient when inserting into a list of length approximately a power of two because this allows for a (almost) completely balanced decision tree. The

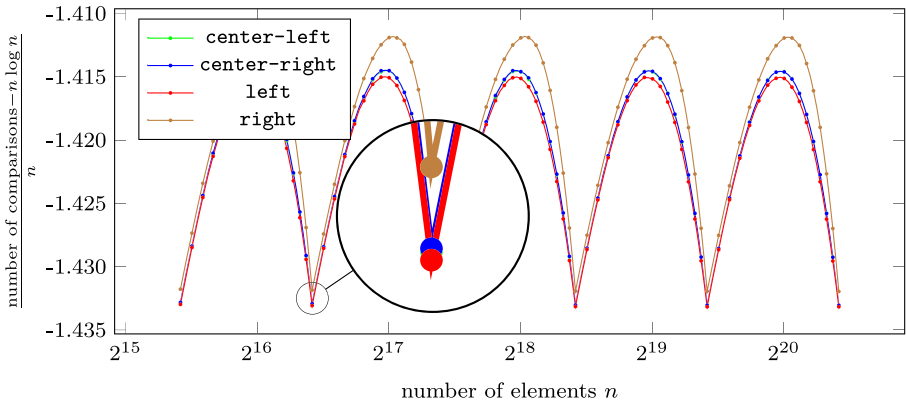


Fig. 7 Experimental results on the effect of different strategies for binary insertion on the number of comparisons

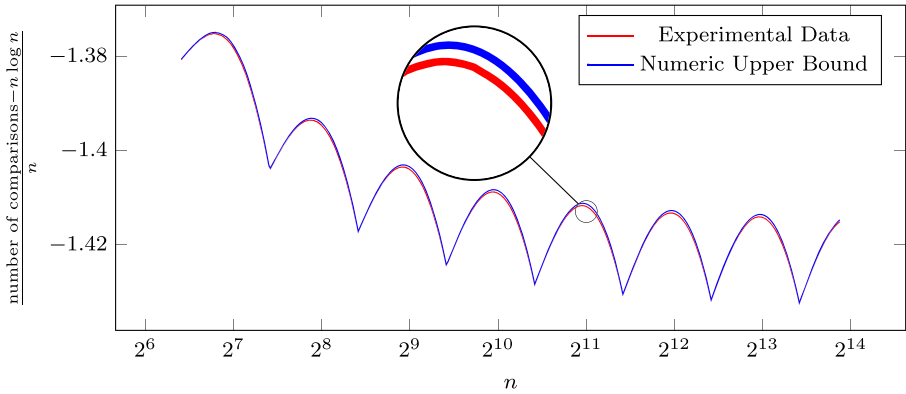


Fig. 8 Comparing our upper bound with experimental data on the number of comparisons required by MergeInsertion

points where MergeInsertion reaches its optimum are instances where the elements of the last batch are inserted into a list of length approximately a power of two (for all other batches this holds in any case).

4 Improved Upper Bounds for MergeInsertion

4.1 Numeric Upper Bound

The goal of this section is to combine the probability given by Theorem 2 that an element $b_{t_{k-1}+i}$ is inserted into j elements with an upper bound for the number of comparisons required for binary insertion.

By [4], the number of comparisons required for binary insertion when inserting into $m - 1$ elements is $T_{\text{InsAvg}}(m) = \lceil \log m \rceil + 1 - \frac{2^{\lceil \log m \rceil}}{m}$. While only being exact in case of a uniform distribution, this formula acts as an upper bound in our case, where the probability is monotonically decreasing with the index.

This leads to an upper bound for the cost of inserting $b_{t_{k-1}+i}$ of $T_{\text{Ins}}(i, k) = \sum_j P(Y_i = j) \cdot T_{\text{InsAvg}}(j + 1)$. From this we calculated an upper bound for MergeInsertion. Figure 8 compares those results with experimental data on the number of comparisons required by MergeInsertion. We observe that the bound is very accurate.

4.2 Computing the Exact Number of Comparisons

In this section we explore how to numerically calculate the exact number of comparisons required in the average case. The most straightforward way of doing this is to compute the external path length of the decision tree (sum of lengths of all paths from the root to leaves) and dividing by the number of leaves ($n!$ when sorting n elements), which unfortunately is only feasible for very small n . Instead we use (1), which describes the number of comparisons. The only unknown in that formula

is $G(n)$, the number of comparisons required in the insertion step of the algorithm. Since the insertion step of MergeInsertion works by inserting elements in batches, we write $G(n)$ as the sum of the cost of those batches:

$$G(n) = \left(\sum_{1 < k \leq k_n} \text{Cost}(t_{k-1}, t_k) \right) + \text{Cost}(t_{k_n}, n) \text{ where } t_{k_n} \leq n < t_{k_n+1}.$$

Algorithm 1 Computation of $\text{Cost}(s, e)$.

```

1: procedure  $\text{Cost}(s, e)$ 
2:    $r \leftarrow e - s$                                 ▷ next element to be inserted is  $b_r$ 
3:    $q_1 \leftarrow 2s$                                 ▷ number of elements on the main chain that are  $< a_{s+1}$ 
4:    $q_2, \dots, q_r \leftarrow 0$                     ▷  $q_i$  is the number of elements between  $a_{s+i-1}$  and  $a_{s+i}$ 
5:    $(p, l) \leftarrow \text{COSTINSERT}(r, q_1, \dots, q_r)$ 
6:   return  $\frac{p}{l}$ 
7: end procedure
8:
9: procedure  $\text{COSTINSERT}(r, q_1, \dots, q_r)$ 
10:  if  $r = 0$  then
11:    return  $(0, 1)$                                 ▷ We reached a leaf
12:  end if
13:   $elements \leftarrow r - 1 + \sum q_i$                 ▷ number of elements  $b_r$  is inserted into
14:   $k \leftarrow \lceil \log(elements + 1) \rceil$ 
15:   $cheap\_insertions \leftarrow 2^k - elements - 1$ 
16:   $p \leftarrow 0$                                     ▷ external path length
17:   $l \leftarrow 0$                                     ▷ number of leaves
18:   $index \leftarrow 0$                                 ▷ We iterate over all indices where  $b_r$  can be inserted
19:  for all  $0 < i \leq r$  do
20:     $(p_c, l_c) \leftarrow \text{COSTINSERT}(r - 1, q_1, \dots, q_{i-1}, q_i + 1, q_{i+1}, \dots, q_{r-1})$ 
21:    repeat  $q_i + 1$  times                            ▷  $q_i + 1$  positions between  $a_{s+i-1}$  and  $a_{s+i}$ 
22:      if  $index < cheap\_insertions$  then
23:         $p \leftarrow p + p_c + (k - 1) \cdot l_c$ 
24:      else
25:         $p \leftarrow p + p_c + k \cdot l_c$ 
26:      end if
27:       $l \leftarrow l + l_c$ 
28:       $index \leftarrow index + 1$ 
29:    end
30:  end for
31:  return  $(p, l)$ 
32: end procedure

```

Here $\text{Cost}(s, e)$ is the cost of inserting one batch of elements starting from b_{s+1} up to b_e . The idea for computing $\text{Cost}(s, e)$ is to calculate the external path length of the decision tree corresponding to the insertion of that batch of elements and then dividing by the number of leaves.

Fig. 9 Computed values of $F(n) \cdot n!$

$n =$	1	2	3	4	5	6	7	8
$F(n) \cdot n! =$	0	2	16	112	832	6912	62784	623232
$n =$	9	10	11					
$F(n) \cdot n! =$	6743808	79292160	1013736960					
$n =$	12	13						
$F(n) \cdot n! =$	13921182720	20448999360						
$n =$	14	15						
$F(n) \cdot n! =$	3199119114240	53153472153600						

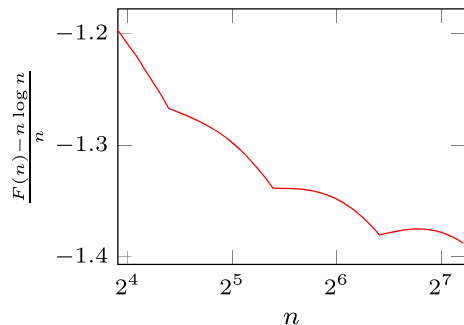
As this is still not feasible, we apply the following optimization: We collapse “identical” branches of the decision tree. E.g. whether b_e is inserted between x_1 and x_2 or between x_2 and x_3 does not influence the number of comparisons required to insert the subsequent elements. So we can neglect that difference. However, if b_e is inserted between a_{e-1} and a_e , then the next element (and all thereafter) is inserted into one less element. So this is a difference we need to acknowledge. Same if an element is inserted between any a_i and a_{i+1} . By the time we insert b_i the element inserted between a_i and a_{i+1} is known to be larger than b_i and thus is no longer part of the main chain, resulting in b_i being inserted into one element less. In conclusion that means that our algorithm needs to keep track of the elements inserted between any a_i and a_{i+1} as well as those inserted at any position before a_{s+1} as two branches of the decision tree that differ in any of these cannot be collapsed. Algorithm 1 shows how this is implemented.

For $n \in \{1, \dots, 15\}$ the computed values are shown in Fig. 9, for larger n Fig. 10 shows the values we computed. The complete data set is provided in the file `exact.txt` in [13]. Our results match up with the values for $n \in \{1, \dots, 8\}$ calculated in [8]. Note that for these values the chosen insertion strategy does not affect the average case (we use the `left` strategy).

4.3 Improved theoretical upper bounds

In this section we improve upon the upper bound from [3, 4] leading to the following result:

Fig. 10 Computed values of $F(n)$



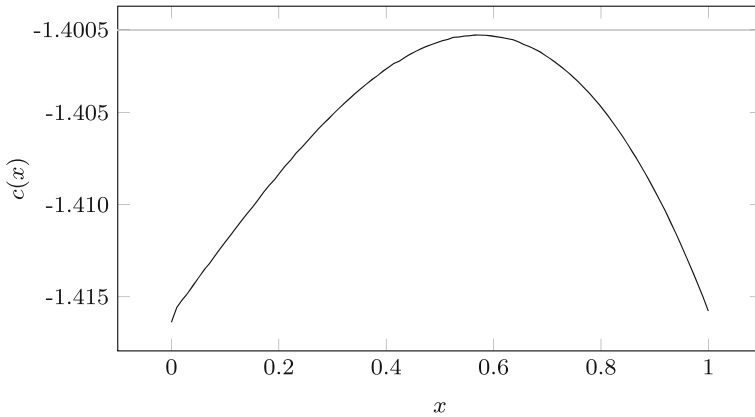


Fig. 11 Plot of $c(x)$

Theorem 3 *The number of comparisons required in the average case of MergeInsertion is at most $n \log n + c(x_n) \cdot n \pm \mathcal{O}(\log^2 n)$ where x_n is the fractional part of $\log(3n)$, i. e., the unique value in $[0, 1)$ such that $n = 2^{k-\log 3+x_n}$ for some $k \in \mathbb{Z}$ and $c : [0, 1) \rightarrow \mathbb{R}$ is given by the following formula:*

$$c(x) = -(3 - \log 3) + (2 - x - 2^{1-x}) - (1 - 2^{-x}) \left(\frac{3}{2^x + 1} - 1 \right) - \frac{2^{\log 3 - x}}{2292} \leq -1.4005$$

Hence we have obtained a new upper bound for the average case of MergeInsertion which is $n \log n - 1.4005n + \mathcal{O}(\log^2 n)$. A visual representation of $c(x)$ is provided in Fig. 11. The worst case is near $x = 0.6$ (i. e., n roughly a power of two) where $c(x)$ is just slightly smaller than -1.4005 . Figure 12 compares our upper bound with experimental data on the number of comparisons – we see that there is still room to improve the upper bound.

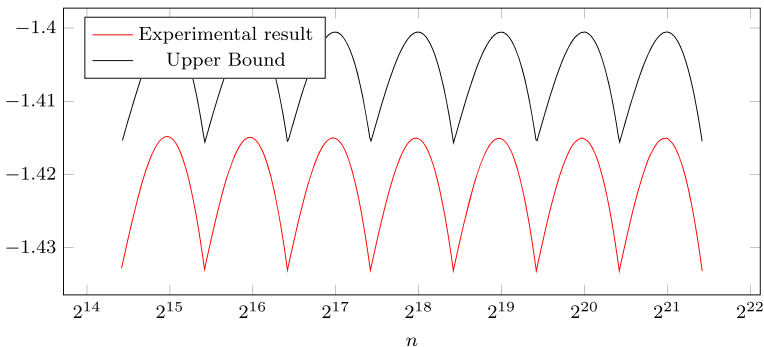


Fig. 12 Comparing the upper bound from Theorem 3 with experimental data

Proof The proof of Theorem 3 analyzes the insertion of one batch of elements more carefully than in [4] (where simply the worst case is used as an upper bound of the average case for all but the last batches). The exact probability that $b_{t_{k-1}+i}$ is inserted into j elements is given by Theorem 2. We are especially interested in the case of $b_{t_{k-1}+u}$ where $u = \lfloor \frac{t_k - t_{k-1}}{2} \rfloor$, because, if we know $P(Y_u < m)$, then we can use that for all $q < u$ we have $P(Y_q < m) \geq P(Y_u < m)$ by Lemma 1. With this we can obtain an upper bound on the average number of comparisons to insert $b_{t_{k-1}+i}$.

However, Theorem 2 is hard to work with, so we approximate it with a binomial distribution. For a given k let $d = t_k - t_{k-1}$ be the number of elements that are inserted as part of the batch. This configuration is illustrated in Fig. 13. Remember $u = \frac{t_k - t_{k-1}}{2} = \frac{d}{2}$. To calculate into how many elements $b_{t_{k-1}+u} = b_{t_{k-1} + \frac{d}{2}}$ is inserted, we ask how many elements out of $b_{t_{k-1} + \lfloor \frac{3}{4}d \rfloor}$ to b_{t_k} (marked as section B in Fig. 13) are inserted between $a_{t_{k-1} + \frac{d}{2} + 1}$ and $a_{t_{k-1} + \lfloor \frac{3}{4}d \rfloor - 1}$ (marked as section A).

The rationale is that for each element from section B that is inserted into section A, $b_{t_{k-1}+u}$ is inserted into one less element. As a lower bound for the probability that an element from section B is inserted into one of the positions in section A we use the probability that b_{t_k} is inserted between $a_{t_{k-1}}$ and a_{t_k} , which is $\frac{1}{2t_{k-1}}$.

That is because, if we assume that all b_i with $i < t_k$ are inserted before inserting b_{t_k} , then b_{t_k} is inserted into $2t_k - 2$ elements, so the probability for each position is $\frac{1}{2t_k - 1}$. Since none of the b_i with $i < t_k$ can be inserted between $a_{t_{k-1}}$ and a_{t_k} because they are all smaller than $a_{t_{k-1}}$, the probability that b_{t_k} is inserted between $a_{t_{k-1}}$ and a_{t_k} does not change when we insert it first as the algorithm demands.

To calculate the probability that an element $b_{t_k - q}$ with $q > 0$ is inserted into the rightmost position we assume that all b_i with $i < t_k - q$ are inserted before inserting $b_{t_k - q}$. Then $b_{t_k - q}$ is inserted into at most $2t_k - q - 2$ elements, i. e., the elements x_1 to $x_{2t_{k-1}}$, $a_{t_{k-1}+1}$ to $a_{t_k - q - 1}$, $b_{t_{k-1}+1}$ to $b_{t_k - q - 1}$ and at most q elements out of $b_{t_k - q + 1}$ to b_{t_k} .

Hence the probability for each position is greater than $\frac{1}{2t_k - q - 1}$ which is greater than $\frac{1}{2t_{k-1}}$. Since none of the b_i with $i < t_k - q$ can be inserted to the right of $a_{t_k - q - 1}$, the probability that $b_{t_k - q}$ is inserted into any of the positions between $a_{t_k - q - 1}$ and $a_{t_k - q}$ remains unchanged when inserting the elements in the correct order.

The probability that an element is inserted at a specific position is monotonically decreasing with the index. This is because if an element b_i is inserted to the left of an element a_{i-h} then b_{i-h} is inserted into one more element than it would be if b_i had

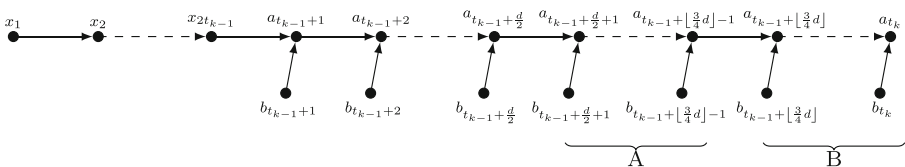


Fig. 13 Configuration where one batch is to be inserted

been inserted to the right of a_{i-h} . As a result any position further to the left is more likely than the right-most position, so we can use that as a lower bound.

There are $\lfloor \frac{d}{4} \rfloor - 1$ elements in section A, i. e., there are at least $\lfloor \frac{d}{4} \rfloor$ positions where an element can be inserted. Hence the probability that an element from section B is inserted into section A is at least $\frac{\lfloor \frac{d}{4} \rfloor}{2t_k-1}$ and consequently the probability that it is not inserted before $b_{t_{k-1}+u}$ is at least $\frac{\lfloor \frac{d}{4} \rfloor}{2t_k-1}$. That is because all positions part of section A are after $a_{t_{k-1}+u}$.

Section B contains $\lceil \frac{d}{2} \rceil$ elements. Using that and substituting $u = \frac{d}{2}$ we obtain the binomial distribution with the parameters $n_B = \lceil \frac{u}{2} \rceil$ and $p_B = \frac{\lfloor \frac{d}{4} \rfloor}{2t_k-1}$. As a result we have

$$p(j) = \binom{\lceil \frac{u}{2} \rceil}{q} \left(\frac{\lfloor \frac{u}{2} \rfloor}{2t_k-1} \right)^q \left(\frac{2t_k-1 - \lfloor \frac{u}{2} \rfloor}{2t_k-1} \right)^{\lceil \frac{u}{2} \rceil - q} \tag{5}$$

with $q = 2^k - 1 - j$, that by construction fulfills the property given in (6) for all j_0 :

$$\sum_{j=0}^{j_0} p(j) \leq \sum_{j=0}^{j_0} P(Y_u = j) = P(Y_u \leq j_0) \tag{6}$$

Figure 14 compares our approximation $p(j)$ with the real distribution $P(Y_u = j)$. We observe that the maximum of our approximation is further to the right than the one of the real distribution.

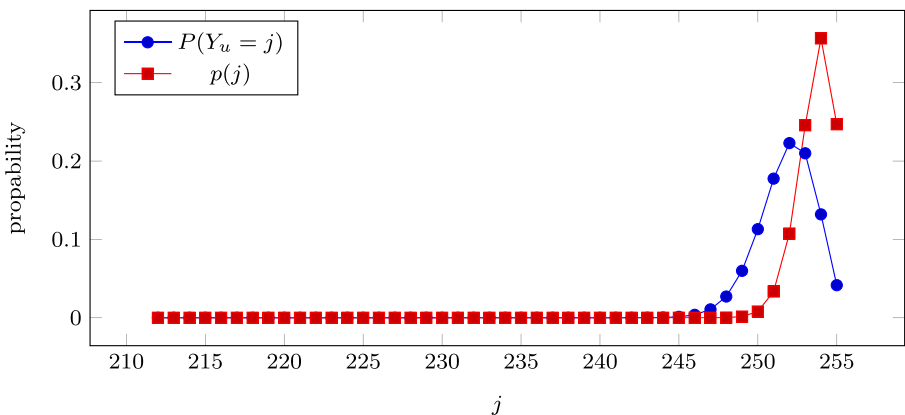


Fig. 14 Difference between the real distribution and our approximation for $k = 8$ and $u = 43$

By using the approximation $P(Y_u = j) \approx p(j)$ we can calculate a lower bound for the median of $Y_{\frac{t_k - t_{k-1}}{2}}$

$$\begin{aligned}
 & 2^k - 1 - \lfloor n_B \cdot p_B \rfloor \\
 = & 2^k - 1 - \left\lfloor \left\lceil \frac{t_k - t_{k-1}}{4} \right\rceil \frac{\lfloor \frac{t_k - t_{k-1}}{4} \rfloor}{2t_k - 1} \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{k-2} + (-1)^k - 3}{3} + \frac{1}{2}(-1)^k + \frac{1}{2} \right) \left(\frac{2^{k-2} + (-1)^k - 3}{3} + \frac{1}{2}(-1)^k - \frac{1}{2} \right) \frac{1}{2t_k - 1} \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{k-2}}{3} + \frac{1}{6}(-1)^k + \frac{1}{2} \right) \left(\frac{2^{k-2}}{3} + \frac{1}{6}(-1)^k - \frac{1}{2} \right) \frac{1}{2t_k - 1} \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{2k-4}}{9} + \frac{2^{k-2}}{9}(-1)^k + \frac{1}{36} - \frac{1}{4} \right) \frac{1}{2t_k - 1} \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{2k-4}}{9} + \frac{2^{k-2}}{9}(-1)^k + \frac{1}{36} - \frac{1}{4} \right) \left(\frac{1}{2^{\frac{2k+1}{3} + (-1)^k} - 1} \right) \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{2k-4}}{9} + \frac{2^{k-2}}{9}(-1)^k + \frac{1}{36} - \frac{1}{4} \right) \left(\frac{1}{2^{\frac{2k+1}{3}} \pm \mathcal{O}(2^{-k})} \right) \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \left(\frac{2^{2k-4}}{9} + \frac{2^{k-2}}{9}(-1)^k + \frac{1}{36} - \frac{1}{4} \right) \frac{1}{2^{\frac{2k+1}{3}} \pm \mathcal{O}(1)} \right\rfloor \\
 = & 2^k - 1 - \left\lfloor \frac{2^{k-6}}{3} + \frac{1}{3}(-1)^k \pm \mathcal{O}(1) \right\rfloor \\
 = & 2^k - 1 - \frac{2^{k-6}}{3} \pm \mathcal{O}(1)
 \end{aligned}$$

This tells us that with a probability $\geq 50\%$, $b_{t_{k-1}+u}$ is inserted into $2^k - 1 - \frac{2^{k-6}}{3} \pm \mathcal{O}(1)$ or less elements. In conclusion all b_i with $i \leq u = \frac{t_k - t_{k-1}}{2}$ are inserted into less than $2^k - 1 - \frac{2^{k-6}}{3} \pm \mathcal{O}(1)$ elements with a probability $\geq 50\%$.

Using that result we can calculate a better upper bound for the average case performance of the entire algorithm.

According to Knuth [8] in its worst case MergeInsertion requires $F_{wc}(n) = n \log n - (3 - \log 3)n + n(y + 1 - 2^y) + \mathcal{O}(\log n)$ comparisons where $y = y(n) = \lceil \log(3n/4) \rceil - \log(3n/4) \in [0, 1)$.

We calculate the number of comparisons $F_{av}(n)$ required in the average case in a similar fashion to [4]. Recall (1) which states that the number of comparisons required by MergeInsertion is given by $F_{av}(n) = \lfloor \frac{n}{2} \rfloor + F_{av}(\lfloor \frac{n}{2} \rfloor) + G_{av}(\lceil \frac{n}{2} \rceil)$ and we have an analogous formula for $F_{wc}(n)$. Here $G_{av}(m)$ corresponds to the work done in the insertion step and can be bounded as

$$G_{av}(m) \leq (k_m - \alpha_m)(m - t_{k_m-1}) + \sum_{1 \leq k < k_m} (k - \beta_k)(t_k - t_{k-1}) \tag{7}$$

where $t_{k_{m-1}} \leq m < t_{k_m}$ and $\alpha_m, \beta \in [0, 1]$. Inserting an element b_i with $t_{k_{i-1}} < i \leq t_{k_i}$ requires at most k_i comparisons. However, since we are looking at the average case we need to consider that in some cases b_i can be inserted using just $k_i - 1$ comparisons. This is reflected by α_m and β_k , the first of which has already been studied by [4].

To estimate the cost of an insertion we use the formula $T_{\text{InsAvg}}(m) = \lceil \log m \rceil + 1 - \frac{2^{\lceil \log m \rceil}}{m}$ by [4]. Technically this formula is only correct if the probability of an element being inserted is the same for each position. This is not the case with MergeInsertion. Instead the probability is monotonically decreasing with the index. Binary insertion can be implemented to take advantage of this property, as explained in Section 3.1, in which case $T_{\text{InsAvg}}(m)$ acts as an upper bound on the cost of an insertion.

Using our result from above that on average $\frac{1}{4}$ of the elements are inserted in less than $2^k - 1 - \frac{2^{k-4}}{9} \pm \mathcal{O}(1)$ elements we can calculate β_k as the difference of the cost of an insertion in the worst-case (k comparisons) and in the average case:

$$\begin{aligned} \beta_k &\geq k - \left(\frac{3}{4} T_{\text{InsAvg}}(2^k) + \frac{1}{4} T_{\text{InsAvg}} \left(2^k - \frac{2^{k-6}}{3} \pm \mathcal{O}(1) \right) \right) \\ &= k - \left(\frac{3}{4} \left(k + 1 - \frac{2^k}{2^k} \right) + \frac{1}{4} \left(k + 1 - \frac{2^k}{2^k - \frac{2^{k-6}}{3} \pm \mathcal{O}(1)} \right) \right) \\ &= -1 + \frac{3}{4} + \frac{1}{4} \cdot \frac{1}{1 - \frac{1}{1 - \frac{2^{-6}}{3}} \pm \mathcal{O}(2^{-k})} \\ &= -\frac{1}{4} + \frac{1}{4} \cdot \frac{1}{1 - \frac{1}{192}} \pm \mathcal{O}(2^{-k}) \\ &= -\frac{1}{4} + \frac{1}{4} \cdot \frac{1}{\frac{191}{192}} \pm \mathcal{O}(2^{-k}) \\ &= -\frac{1}{4} + \frac{1}{4} \cdot \frac{192}{191} \pm \mathcal{O}(2^{-k}) \\ &= \frac{1}{764} \pm \mathcal{O}(2^{-k}) \end{aligned}$$

Combining this with (7) we calculate the difference between worst-case and average-case as

$$\begin{aligned} G_{\text{wc}}(m) - G_{\text{av}}(m) &\geq k_m(m - t_{k_{m-1}}) + \sum_{1 \leq k < k_m} k(t_k - t_{k-1}) \\ &\quad - (k_m - \alpha_m)(m - t_{k_{m-1}}) - \sum_{1 \leq k < k_m} (k - \beta_k)(t_k - t_{k-1}) \\ &= \alpha_m(m - t_{k_{m-1}}) + \sum_{1 \leq k < k_m} \beta_k(t_k - t_{k-1}) \\ &\geq \alpha_m(m - t_{k_{m-1}}) + \sum_{1 \leq k < k_m} \left(\frac{1}{764} \pm \mathcal{O}(2^{-k}) \right) (t_k - t_{k-1}) \end{aligned}$$

$$\begin{aligned}
 &= \alpha_m(m - t_{k_m-1}) + \frac{1}{764}(t_{k_m-1} - t_1) \pm \mathcal{O}(\log m) \\
 &= \alpha_m(m - t_{k_m-1}) + \frac{1}{764}t_{k_m-1} \pm \mathcal{O}(\log m) \\
 &= \alpha_m(m - t_{k_m-1}) + \frac{1}{764} \frac{2^k + (-1)^{k_m-1}}{3} \pm \mathcal{O}(\log m) \\
 &= \alpha_m(m - t_{k_m-1}) + \frac{1}{764} \frac{2^{k_m}}{3} \pm \mathcal{O}(\log m) \tag{8}
 \end{aligned}$$

By writing m as $m = 2^{l_m - \log 3 + x}$ with $x \in [0, 1)$ we get $l_m = \lfloor \log 3m \rfloor$. To approximate k_m with l_m we need to show that $k_m \geq l_m$. Recall that $t_{k_m-1} \leq m < t_{k_m}$. For all $t_{k_m-1} < m < t_{k_m}$ we have

$$\frac{2^{k_m} + (-1)^{k_m-1}}{3} < m < \frac{2^{k_m+1} + (-1)^{k_m}}{3}.$$

Since $m \in \mathbb{N}$ and $t_k \in \mathbb{N}$ adding/subtracting $\frac{1}{3}$ does not alter the relation, we obtain

$$\frac{2^{k_m}}{3} < m < \frac{2^{k_m+1}}{3},$$

which resolves to

$$k_m < \log 3m < k_m + 1.$$

Thus, $k_m = \lfloor \log 3m \rfloor = l_m$. For $m = t_{k_m-1}$ we get

$$\begin{aligned}
 \frac{2^{k_m} + (-1)^{k_m-1}}{3} &= m \\
 \iff 2^{k_m} &= 3m + (-1)^{k_m} \\
 \iff k_m &= \log \left(3m + (-1)^{k_m} \right).
 \end{aligned}$$

If $k_m = \log (3m + 1)$, that resolves to $k_m = \log (3m + 1) > \log (3m) > \lfloor \log 3m \rfloor = l_m$. If instead $k_m = \log (3m - 1)$, using $k_m \in \mathbb{N}$, we have $k_m = \lfloor \log (3m - 1) \rfloor$ and for all $m \geq 1$ this is equal to $\lfloor \log 3m \rfloor = l_m$. Hence, in all cases $l_m \leq k_m$ holds. Therefore, we can replace k_m with l_m in (8):

$$G_{wc}(m) - G_{av}(m) \geq \alpha_m(m - t_{k_m-1}) + \frac{1}{764} \frac{2^{l_m}}{3} \pm \mathcal{O}(\log m)$$

From [4] we know that the $\alpha_m(m - t_{k_m-1})$ term can be approximated with $(m - 2^{l_m - \log 3}) \left(\frac{2^{l_m}}{m + 2^{l_m - \log 3}} - 1 \right)$. Hence,

$$\begin{aligned}
 G_{wc}(m) - G_{av}(m) &\geq \left(m - 2^{l_m - \log 3} \right) \left(\frac{2^{l_m}}{m + 2^{l_m - \log 3}} - 1 \right) + \frac{1}{764} \frac{2^{l_m}}{3} \pm \mathcal{O}(\log m).
 \end{aligned}$$

Now we calculate

$$\begin{aligned}
 S(n) &= F_{wc}(m) - F_{av}(m) \\
 &= \lfloor \frac{n}{2} \rfloor + F_{wc}(\lfloor \frac{n}{2} \rfloor) + G_{wc}(\lceil \frac{n}{2} \rceil) - \lfloor \frac{n}{2} \rfloor - F_{av}(\lfloor \frac{n}{2} \rfloor) - G_{av}(\lceil \frac{n}{2} \rceil) \\
 &= S(\lfloor \frac{n}{2} \rfloor) + G_{wc}(\lceil \frac{n}{2} \rceil) - G_{av}(\lceil \frac{n}{2} \rceil) \\
 &\geq S(\lfloor \frac{n}{2} \rfloor) + (m - 2^{l_m - \log 3}) \left(\frac{2^{l_m}}{m + 2^{l_m - \log 3}} - 1 \right) + \frac{1}{764} \frac{2^{l_m}}{3} \pm \mathcal{O}(\log m).
 \end{aligned}
 \tag{9}$$

We split $S(n)$ into $S_\alpha(n) + S_\beta(n)$ with

$$\begin{aligned}
 S_\alpha(n) &\geq S_\alpha(\lfloor \frac{n}{2} \rfloor) + (m - 2^{l_m - \log 3}) \left(\frac{2^{l_m}}{m + 2^{l_m - \log 3}} - 1 \right), \\
 S_\beta(n) &\geq S_\beta(\lfloor \frac{n}{2} \rfloor) + \frac{1}{764} \frac{2^{l_m}}{3} \pm \mathcal{O}(\log m).
 \end{aligned}$$

From [4] we know $S_\alpha(n) \geq (n - 2^{l_n - \log 3}) \left(\frac{2^{l_n}}{n + 2^{l_n - \log 3}} - 1 \right) + \mathcal{O}(1)$. For $S_\beta(n)$ we obtain

$$S_\beta(n) \geq \sum_{i=1}^{l_n - 1} \frac{2^i}{764 \cdot 3} \pm \mathcal{O}(\log 2^i) = \frac{2^{l_n}}{2292} \pm \mathcal{O}(\log^2 n).$$

We can represent n as $2^{k - \log 3 + x_n}$ with $x_n \in [0, 1)$. This leads to

$$\begin{aligned}
 \frac{S(n)}{n} &= \frac{S_\alpha(n) + S_\beta(n)}{n} \\
 &\geq \frac{2^{k - \log 3 + x_n} - 2^{k - \log 3}}{2^{k - \log 3 + x_n}} \left(\frac{2^k}{2^{k - \log 3 + x_n} + 2^{k - \log 3}} - 1 \right) + \frac{2^k}{2292 \cdot 2^{k - \log 3 + x_n}} \pm \mathcal{O}\left(\frac{\log^2 n}{n}\right) \\
 &= (1 - 2^{-x_n}) \left(\frac{3}{2^{x_n} + 1} - 1 \right) + \frac{2^{\log 3 - x_n}}{2292} \pm \mathcal{O}\left(\frac{\log^2 n}{n}\right).
 \end{aligned}$$

Finally, we can combine our bound for $S(n)$ with Knuth’s bound for $F_{wc}(n)$ in order to obtain a bound on $F_{av}(n)$:

$$\begin{aligned}
 F_{av}(n) &= F_{wc}(n) - S(n) \\
 &\leq n \log n - (3 - \log 3)n + n(y + 1 - 2^y) \\
 &\quad - \left((1 - 2^{-x_n}) \left(\frac{3}{2^{x_n} + 1} - 1 \right) + \frac{2^{\log 3 - x_n}}{2292} \right) \cdot n \pm \mathcal{O}(\log^2 n) \\
 &= n \log n + c(x_n) \cdot n \pm \mathcal{O}(\log^2 n)
 \end{aligned}$$

where $y = 1 - x_n$ and

$$c(x_n) = -(3 - \log 3) + (y + 1 - 2^y) - (1 - 2^{-x_n}) \left(\frac{3}{2^{x_n} + 1} - 1 \right) - \frac{2^{\log 3 - x_n}}{2292}.$$

Thus, we have concluded the proof of Theorem 3. □

5 Experiments

In this section we discuss our experiments, which consist of two parts: first, we evaluate how increasing t_k by some constant factor can reduce the number of comparisons, then we examine how the combination with the (1,2)-Insertion algorithm as proposed in [7] improves MergeInsertion.

All experiments use the `left` strategy for binary insertion (see Section 3.1). The number of comparisons has been averaged over 10 to 10000 runs, depending on the size of the input.

5.1 Implementing MergeInsertion

We implemented MergeInsertion using a tree based data structure, similar to the Rope data structure [1] used in text processing, resulting in a comparably “fast” implementation. The pseudo code is provided in Algorithm 2. For the purpose of our implementation we assume that each element is unique. This condition is easy to fulfill for synthetic test data. We now go over some of the key challenges when implementing MergeInsertion.

1. MergeInsertion requires elements to be inserted into arbitrary positions. When using a simple array to store the elements this operation requires moving $\mathcal{O}(n)$ elements. Since MergeInsertion inserts each element exactly once this results in a complexity of $\mathcal{O}(n^2)$. To avoid this we store the elements in a custom data structure inspired by the Rope data structure [1] used in text processing. Being based on a tree it offers $\mathcal{O}(\log n)$ performance for lookup, insertion and deletion operations, thus putting our algorithm in $\mathcal{O}(n \log^2 n)$.
2. In the second step of the algorithm we need to rename the b_i after the recursive call. Our chosen solution is to store which a_i corresponds to which b_i in a hash map (line 11) before the recursive call and use the information to reorder the b_i afterwards (line 13). The disadvantage of this solution is that it requires each element to be unique and the hash map might introduce additional comparisons (which we do not count as such in our experiments).

An alternative would be to have the recursive call generate the permutation it applies to the larger elements and then apply that to the smaller ones. That is a cleaner solution as it does not require the elements to be unique and it avoids potentially introducing additional comparisons. It is also potentially faster, though not by much. Since we only are interested in the number of comparisons on random permutations, we chose to use a hash map as that solution is easier to implement.

3. In the insertion step we need to know into how many elements a specific b_i is inserted. For b_{t_k} this is $2^k - 1$ elements. However, for other elements that number can be smaller depending on where the previous elements have been inserted. To account for that, we create the variable u in line 21. It holds the position of the a_i corresponding to the element b_i that is inserted next. Thus b_i is inserted into

$u - 1$ elements (since $b_i < a_i$). After the insertion of b_i , we decrease u in line 25 until it matches the position of a_{i-1} so that the next element b_{i-1} can be inserted into $u - 1$ elements again (with the new value for u).

Algorithm 2 MergeInsertion.

```

1: procedure MERGEINSERTION( $d$  : array of  $n$  elements)
2:   Step 1: Pairwise comparison
3:   for all  $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$  do                                 $\triangleright$  Split into larger and smaller half
4:      $a_i \leftarrow \max \left\{ d_i, a_{i+\lfloor \frac{n}{2} \rfloor} \right\}$ 
5:      $b_i \leftarrow \min \left\{ d_i, a_{i+\lfloor \frac{n}{2} \rfloor} \right\}$ 
6:   end for
7:   if  $n \bmod 2 = 1$  then
8:      $b_{\lfloor \frac{n}{2} \rfloor} \leftarrow d_n$ 
9:   end if
10:  Step 2: Recursion and Renaming
11:   $m \leftarrow \{(a_i, b_i) \mid 1 \leq i \leq \lfloor \frac{n}{2} \rfloor\}$                                  $\triangleright$  Store mapping
12:   $a \leftarrow \text{MERGEINSERTION}(a)$ 
13:  for all  $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$  do                                 $\triangleright$  Permute smaller half
14:     $b_i \leftarrow e$  where  $(a_i, e) \in m$ 
15:  end for
16:  Step 3: Insertion
17:   $d \leftarrow b_1, a_1, \dots, a_{\lfloor \frac{n}{2} \rfloor}$ 
18:   $k \leftarrow 2$ 
19:  while  $t_{k-1} < \lceil \frac{n}{2} \rceil$  do
20:     $m \leftarrow \min \left\{ t_k, \lceil \frac{n}{2} \rceil \right\}$                                  $\triangleright$  first element of the batch
21:     $u \leftarrow t_{k-1} + m$                                              $\triangleright$  position of  $a_m$  in  $d$ 
22:    for  $i$  in  $m$  down to  $t_{k-1} + 1$  do
23:       $d \leftarrow \text{BINARYINSERTION}(b_i, d_1, \dots, d_{u-1}), d_u, \dots, d_{2m+t_{k-1}-i}$ 
24:      while  $d_u \neq a_{i-1}$  do                                         $\triangleright$  adjust  $u$ 
25:         $u \leftarrow u - 1$ 
26:      end while
27:    end for
28:     $k \leftarrow k + 1$ 
29:  end while
30:  return  $d$ 
31: end procedure

```

Notice that this step also makes use of the requirement that each element is unique. Also, at this point we have to be aware that testing whether the element at position u is a_{i-1} might introduce additional comparisons to the algorithm. This is acceptable because we do not count these comparisons. Also these are not necessary. We could keep track of the positions of the elements a_i ; however, we chose not to, in order to keep the implementation simple.

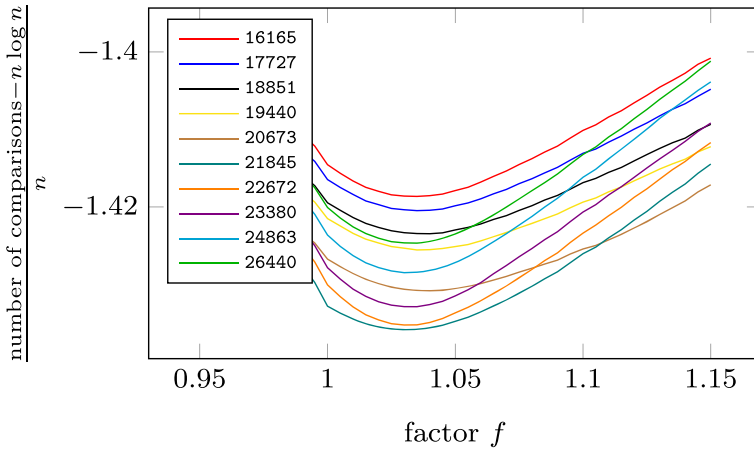


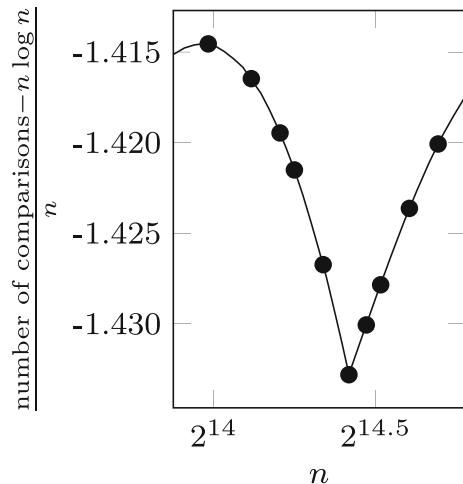
Fig. 15 Effects of replacing t_k with \hat{t}_k

5.2 Increasing t_k by a Constant Factor

In this section we modify MergeInsertion by replacing t_k with $\hat{t}_k = \lfloor f \cdot t_k \rfloor$ – apart from that the algorithm is the same. Originally, the numbers t_k have been chosen such that each element b_i with $t_{k-1} < i \leq t_k$ is inserted into at most $2^k - 1$ elements (which is optimal for the worst case). As we have seen in previous sections, many elements are inserted into slightly less than $2^k - 1$ elements. The idea behind increasing t_k by a constant factor f is to allow more elements to be inserted into close to $2^k - 1$ elements.

Figure 15 shows how different factors f affect the number of comparisons required by MergeInsertion. The different lines represent different input lengths. For instance, $n = 21845$ is an input size for which MergeInsertion works best. An

Fig. 16 n used in Fig. 15



overview of the different input lengths and how original MergeInsertion performs for these can be seen in Fig. 16. The chosen values are assumed to be representative for the entire algorithm. We observe that for all shown input lengths, multiplying t_k by a factor f between 1.02 and 1.05, leads to an improvement.

Figure 17 compares different factors from 1.02 to 1.05. The factor 1.0 (i. e., the original algorithm) is included as a reference. We observe that all the other factors lead to a considerable improvement compared to 1.0. The difference between the factors in the chosen range is rather small. However, 1.03 appears to be best out of the tested values. At $n \approx 2^k/3$ the difference to the information-theoretic lower bound is reduced to $0.007n$, improving upon the original algorithm, which has a difference of $0.01n$ to the optimum.

Another observation we make from Fig. 17 is that the plot periodically repeats itself with each power of two. Thus, we conclude that replacing t_k with $\hat{t}_k = \lfloor f \cdot t_k \rfloor$ with $f \in [1.02, 1.05]$ reduces the number of comparisons required per element by some constant.

5.3 Merging

In this section we validate whether splitting the input into two parts, sorting them separately and then merging the sorted parts leads to an improvement in the average case of the algorithm. The idea was presented first in [9]. It relies on the fact that there are specific points $u_k = \lfloor (\frac{4}{3}) 2^k \rfloor$ where MergeInsertion reaches its optimum. The input is split into two parts with size m_1 and m_2 such that $m_1 = \max \{u_k \mid u_k \leq n\}$ and $m_2 = n - m_1$. As a result we have $m_1 > m_2$ and m_1 is of a size for with MergeInsertion is optimal.

For merging the two lists we use the Hwang-Lin Algorithm [6], for it is a simple general purpose merging algorithm. It merges two lists by comparing the first element of the small list with the 2^r -th element of the large list, where $r = \lfloor \log \frac{m_1}{m_2} \rfloor$. If it is larger, then the 2^r first elements of the large list are removed and appended to the result, and the process is repeated. If it is smaller, then it is inserted into the $2^r - 1$

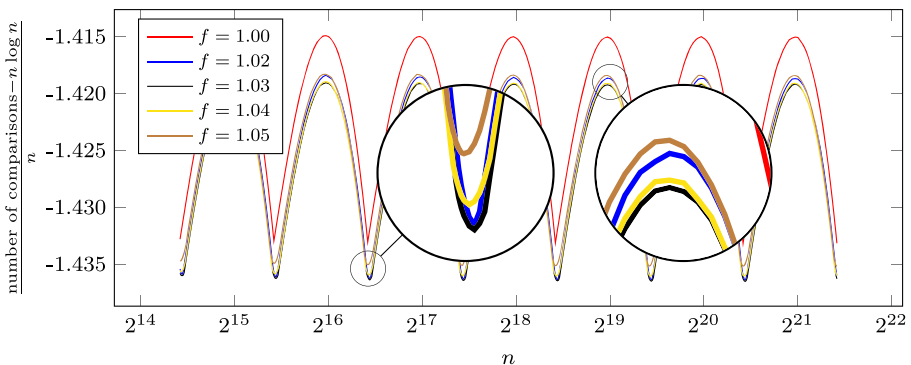


Fig. 17 Comparison of different factors f for \hat{t}_k

first elements of the large list using binary insertion and then all elements up to the freshly inserted one are removed from the large list and appended to the result. This is repeated until one of the lists is exhausted.

To be precise, we actually use a variant of the Hwang-Lin Algorithm called Static Hwang-Lin Algorithm presented in [9]. The difference is that $r = \lfloor \log \frac{m_1}{m_2} \rfloor$ is only calculated once (hence “static”) instead of every iteration. This leads to slightly improved performance in cases where $\frac{m_1}{m_2}$ is close to a power of 2.

Both [9] and [2] present more advanced merging algorithms which possibly could yield better results than the ones we have obtained. Since our primary aim in this section is to establish experimentally that merging also helps in the average case, we do not bother to implement these more complicated algorithms. Even without using those advanced merging algorithms our experiments show that this technique does indeed improve upon MergeInsertion.

The results of our experiments are presented in Fig. 18. Here, $F(n)$ denotes the number of comparisons required by MergeInsertion in the average-case and $T(n)$ is the number of comparisons required when splitting the list into two parts sorting them separately and merging them afterwards as explained above. As we can see $T(n)$ beats $F(n)$ for some ranges of n , but for other values of n , pure MergeInsertion is better. Also observe that the points where $T(n)$ is best are not those where m_2 is optimal for MergeInsertion. Instead this happens where m_2 is a bit smaller than what would be optimal for MergeInsertion. From this we conclude that $T(n)$ is defined mostly by cost of merging m_1 and m_2 and that the cost of MergeInsertion for m_2 plays only a minor role.

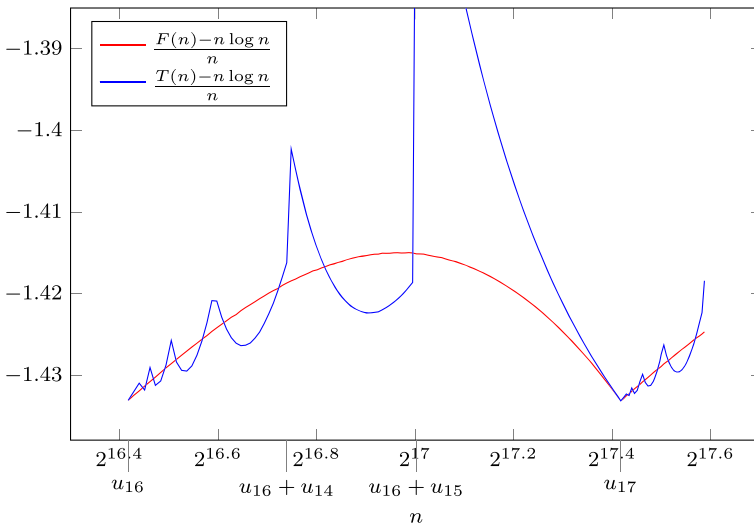


Fig. 18 Experimental results comparing MergeInsertion (red) to the combination (blue) with the Hwang-Lin merging algorithm

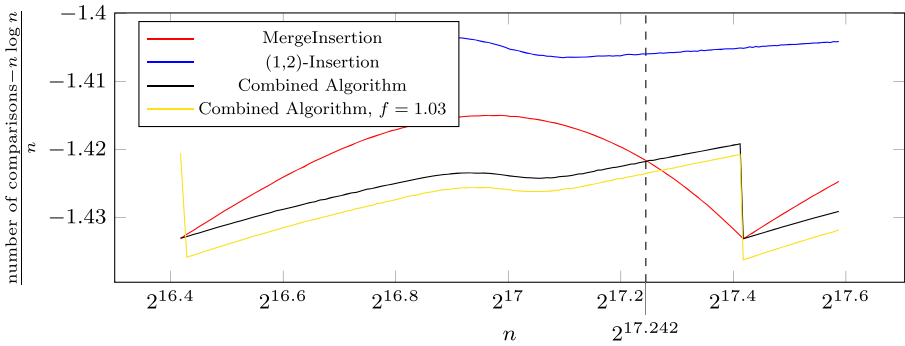


Fig. 19 Experimental results comparing MergeInsertion, (1,2)-Insertion and the combined algorithm

5.4 Combination with (1,2)-Insertion

(1,2)-Insertion is a sorting algorithm presented in [7]. It works by inserting either a single element or two elements at once into an already sorted list. On its own, (1,2)-Insertion is worse than MergeInsertion; however, a combination of the two improves upon both algorithms. The combined algorithm works by sorting $m = \max\{u_k \mid u_k \leq n\}$ elements with MergeInsertion; then the remaining elements are inserted using (1,2)-Insertion.

Let $u_k = \lfloor \left(\frac{4}{3}\right)^k \rfloor$ denote a point where MergeInsertion is optimal. In Fig. 19 we can see that at the point u_k MergeInsertion and the combined algorithm perform the same. However, in the values following u_k the combined algorithm surpasses MergeInsertion until at one point close to the next optimum MergeInsertion is better once again. In their paper Iwama and Teruyama calculated that for $0.638 < \frac{n}{2^{\lceil \log n \rceil}} \leq \frac{2}{3}$ MergeInsertion is better than the combined algorithm. The fraction $\frac{2}{3}$ corresponds to the point where MergeInsertion is optimal. They derived the constant 0.638 from their theoretical analysis using the upper bound for MergeInsertion from [3]. Comparing this to our experimental results we observe that the range where MergeInsertion is better than the combined algorithm starts at $n \approx 2^{17.242}$. This yields $\frac{2^{17.242}}{2^{18}} = 2^{17.242-18} = 2^{-0.758} \approx 0.591$. Hence the range where MergeInsertion is better than the combined algorithm is $0.591 \leq \frac{n}{2^{\lceil \log n \rceil}} \leq \frac{2}{3}$, which is slightly larger than the theoretical analysis suggested. Also shown in Fig. 19 is the number of comparisons of the combined algorithm where we additionally apply our suggestion of replacing t_k by $\hat{t}_k = \lfloor f \cdot t_k \rfloor$ with $f = 1.03$. This leads to an additional improvement and comes even closer to the lower bound of $\log(n!)$.

6 Conclusion and Outlook

We improved the previous upper bound of $n \log n - 1.3999n + o(n)$ to $n \log n - 1.4005n + o(n)$ for the average number of comparisons of MergeInsertion. However,

there still is a gap between the number of comparisons required by MergeInsertion and this upper bound.

In Section 4.3 we used a binomial distribution to approximate the probability of an element being inserted into a specific number of elements during the insertion step. However, the difference between our approximation and the actual probability distribution is rather large. Finding an approximation which reduces that gap while still being simple to analyze with respect to its mean would facilitate further improvements to the upper bound.

Our suggestion of increasing t_k by a constant factor f reduced the number of comparisons required per element by some constant. However, we do not have a proof for this. Thus, future research could try to determine the optimal value for the factor f as well as to study how this suggestion affects the worst-case.

Funding Information Open Access funding provided by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Boehm, H.J., Atkinson, R., Plass, M.: Ropes: an alternative to strings. *Softw. Pract. Exper.* **25**(12), 1315–1330 (1995)
2. Bui, T., Thanh, M.: Significant improvements to the Ford-Johnson algorithm for sorting. *BIT Numer. Math.* **25**(1), 70–75 (1985)
3. Edelkamp, S., Weiß, A.: QuickXsort: Efficient Sorting with $n \log n - 1.399n + o(n)$ Comparisons on Average. In: *CSR 2014 Proc.*, pp. 139–152 (2014)
4. Edelkamp, S., Weiß, A., Wild, S.: Quickxsort: A, fast sorting scheme in theory and practice. *Algorithmica* **82**(3), 509–588 (2020). <https://doi.org/10.1007/s00453-019-00634-0>
5. Ford, L.R., Johnson, S.M.: A tournament problem. *Am. Math. Mon.* **66**(5), 387–389 (1959)
6. Hwang, F.K., Lin, S.: A simple algorithm for merging two disjoint linearly ordered sets. *SIAM J. Comput.* **1**(1), 31–39 (1972)
7. Iwama, K., Teruyama, J.: Improved Average Complexity for Comparison-Based Sorting. In: *Workshop on Algorithms and Data Structures*, pp. 485–496. Springer, New York (2017)
8. Knuth, D.E.: *The art of computer programming, volume 3: (2nd edn.) sorting and searching*. Addison wesley longman, redwood city (1998)
9. Manacher, G.K.: The Ford-Johnson sorting algorithm is not optimal. *J. ACM* **26**(3), 441–456 (1979)
10. Peczarski, M.: New results in minimum-comparison sorting. *Algorithmica* **40**(2), 133–145 (2004)
11. Peczarski, M.: The Ford-Johnson algorithm still unbeaten for less than 47 elements. *Inf. Process. Lett.* **101**(3), 126–128 (2007)
12. Reinhardt, K.: Sorting In-Place with a Worst Case Complexity of $N \log N - 1.3N + O(\log n)$ Comparisons and $\epsilon N \log N + O(1)$ Transports. In: *Algorithms and Computation, ISAAC '92, Proc.*, pp. 489–498 (1992)
13. Stober, F.: Source code and generated data <https://github.com/CodeCrafter47/merge-insertion> (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.