

The Lower Bound on Comparisons Comes From the Number of Answers, Not From the Input

— For $n = 8$ the bound is $\log_2 8! = 15.2992$, rounded up to **16**, and running all **40320** permutations gives a worst case of **17** for both binary insertion and merge sort and **28** for insertion sort — the separator is whether one asks about the count on one input or the count that suffices for every input — [Paper 843]

Yuuki YAMAGISHI (山岸 勇輝)

Yamagishi Complexity Torus Lab, Japan

September 2026

YOI: <https://www.yamagishiyuuki.com/yoi/843>

— *From thinking to syncing.* —

Abstract

When something is called “fast”, the comparison being made is often not the one intended. This paper counts where the lower bound comes from. No new theorem or law is claimed. First, the bound is the logarithm of the number of answers: for $n = 8$ it is $\log_2 8! = 15.2992$, rounded up to **16** (Section 2). Second, real procedures come close to it: binary insertion and merge sort both take **17** in the worst case at $n = 8$, one above the bound (Section 2). Third, others stay far from it: insertion sort takes **28**, which is 1.75 times the bound (Section 2). Fourth, the averages do not fall below it either: binary insertion averages **15.5905**, above $\log_2 8! = 15.2992$ (Section 2). Fifth, and this is the core. On one input the bound can be beaten: eight already-sorted items take insertion sort **7** comparisons, far below **16** (Section 3). Sixth, the separator is whether one asks about the count on one input or the count that suffices for every input: the bound protects only the second (Section 4). **Keywords:** comparison sort, lower bound, decision tree, worst and average case, exhaustive enumeration, model.

Scope of this paper (scope note): No new theorem or law is claimed: that comparison sorting needs $\log_2 n!$ comparisons (a decision tree needs $n!$ leaves), that merge sort and binary insertion sort stay at that order, and that the same bound holds on average, are all known. Implementation speed is not discussed: memory traffic, constant factors and parallelism are not the subject; only comparisons are counted. No optimal count is claimed: the exact minimum number of comparisons for each n was not computed; what was counted is three procedures. Non-comparison sorting is not treated: counting sort, radix sort and other methods that use the values themselves are not entered. The size is not generalised: $n = 9$ and beyond were not counted. Relation to earlier papers: Paper 759 showed that with the same number of comparisons the number of slow-memory reads is set by the tree's branching. This paper stands beside it: there the same count could cost differently, here the floor of the count itself. Paper 300 showed that whether two things share a root is decidable: the count on one input and the count that suffices for every input are distinct roots by that criterion. What is added is running three procedures over all permutations from $n = 4$ to 8 and laying out worst and average counts, measuring that the gap to the bound is one comparison at $n = 8$, measuring the 7 comparisons on sorted input and restating what the bound is about, and placing the separator on one input versus every input.

1. Notation and the Question

Setting: n items are put in increasing order using nothing but pairwise comparisons. Only the comparisons are counted.

$$\text{number of answers} = n!, \quad \text{lower bound on comparisons} = \lceil \log_2 n! \rceil$$

One comparison narrows the answers by at most half, so reaching one of $n!$ takes at least this many. Nothing about the input appears on the right.

Question: what exactly is this a bound on?

2. The Bound and the Real Counts

Computation 2.1: from $n = 4$ to 8, all permutations were run through three procedures.

n	$\log_2 n!$	rounded up	insertion (worst)	binary insertion (worst)	merge (worst)
4	4.5850	5	6	5	5
5	6.9069	7	10	8	8
6	9.4919	10	15	11	11
7	12.2992	13	21	14	14
8	15.2992	16	28	17	17

Observation 2.2

no procedure fell below the bound. At all five sizes the worst case is at least the rounded-up bound: unavoidable once a comparison narrows the answers only by half.

Observation 2.3

some procedures are near and some are far. At $n = 8$ binary insertion and merge take 17, one above the bound of 16, while insertion sort takes 28, which is 1.75 times it: under one bound, the quality of a procedure shows directly.

Computation 2.4: the averages were also counted.

n	$\log_2 n!$	insertion (avg)	binary insertion (avg)	merge (avg)
6	9.4919	11.0500	9.7333	9.8333
7	12.2992	14.9071	12.5905	12.7333
8	15.2992	19.2821	15.5905	15.7333

Observation 2.5

the averages stay above too. Binary insertion averages 15.5905 at $n = 8$, above 15.2992, because the narrowing that must be done is the same on average.

3. On One Input, the Bound Can Be Beaten

Computation 3.1: already-sorted items were fed to insertion sort.

n	comparisons	rounded-up bound
6	5	10
8	7	16

Observation 3.2

this is the core. The bound was beaten.Eight items in 7 comparisons, less than half the bound of 16: no contradiction: the bound is about the count that suffices for every input, not the count on one.

Observation 3.3

the same procedure takes 28 at worst and 7 at best.A spread of 4 times: “fast” fixes no meaning until it says on which inputs.

Observation 3.4

the bound never looks at the input.Only n enters $\lceil \log_2 n! \rceil$; neither the values nor their order do, so “it is fast on inputs like this” has no effect on the formula at all.

4. The Separator

Observation 4.1

the separator is whether one asks about the count on one input or the count that suffices for every input.The first can be made as small as one likes; the second cannot go below $\log_2 n!$: both are called “the number of comparisons” and they are different quantities.

Observation 4.2

placed beside Paper 759.759 showed that with the same number of comparisons the number of slow-memory reads is set by the tree's branching. There the same count could cost differently; here the floor of the count itself. The same quantity is pinned from above and from below. Paper 300 showed that whether two things share a root is decidable: the two counts are distinct roots by that criterion.

Observation 4.3

what is not said. Implementation speed, the exact minimum for each n and non-comparison sorting are not treated. What is shown is that the bound comes from the number of answers, not from the input.

5. Census and Closing

1. the bound is $\log_2 n!$ (Section 1): **no input enters it.**
2. **15.2992** at $n = 8$ (Computation 2.1): **rounded up to 16.**
3. **binary insertion and merge: 17** (Computation 2.1): **one above.**
4. **insertion: 28** (Computation 2.1): **1.75 times the bound.**
5. **binary insertion averages 15.5905** (Computation 2.4): **above on average too.**
6. **sorted eight: 7 comparisons** (Computation 3.1): **the core.**
7. **less than half the bound** (Computation 3.1): **no contradiction.**
8. **28 against 7** (Observation 3.3): **a spread of four.**
9. **only n enters** (Observation 3.4): **values and order do not.**
10. **one input, or every input** (Observation 4.1): **the separator; Papers 759, 300.**

Closing: The lower bound on comparisons comes from the number of answers, not from the input. At $n = 8$ the bound is $\log_2 8! = 15.2992$, rounded up to 16; binary insertion and merge come within one comparison at 17, and insertion sort takes 28, which is 1.75 times it, because one comparison narrows the answers by at most half. Yet eight already-sorted items take insertion sort 7 comparisons, less than half the bound of 16, because the bound is about the count that suffices for every input, not the count on one. The same procedure spreads by a factor of four, 28 against 7, so “fast” fixes no meaning until it says on which inputs. Placed among the earlier papers: beside Paper 759’s “the same count can cost differently”, pinning the same quantity from below, and treating the two counts as

distinct roots by the criterion of Paper 300. To be honest: only three procedures up to $n = 8$ were counted, and the exact minimum for each n was not computed.

References

1. Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley.
 2. Yamagishi, Y. (2026). *The Three Hundredth Census: Whether Two Things Share a Root Is Decidable* [Paper 300]. YOI: <https://www.yamagishiyuuki.com/yoi/300>; DOI: 10.5281/zenodo.22171914
 3. Yamagishi, Y. (2026). *Comparing Equally Often, the Number of Slow-Memory Reads Is Set by the Tree's Branching* [Paper 759]. YOI: <https://www.yamagishiyuuki.com/yoi/759>; DOI: 10.5281/zenodo.23015187
-

— *From thinking to syncing.* —

Yuuki YAMAGISHI (山岸 勇輝)

© 2026 Yamagishi Complexity Torus Lab, Japan · Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0)