Elementary Yet Precise Worst-case Analysis of MergeSort

A manuscript (MS) intended for future publication[☆]

MAREK A. SUCHENEK

California State University Dominguez Hills, Department of Computer Science, 1000 E. Victoria St., Carson, CA 90747, USA, Suchenek@csudh.edu

Abstract

This paper offers two elementary yet precise derivations of an exact formula

$$W(n) = \sum_{i=1}^{n} \lceil \lg i \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$$

for the maximum number $W(n)$ of comparisons of keys performed by `MergeSort` on an n-element array. The first of the two, due to its structural regularity, is well worth carefully studying in its own right.

Close smooth bounds on $W(n)$ are derived. It seems interesting that $W(n)$ is linear between the points $n = 2^{\lfloor \lg n \rfloor}$ and it linearly interpolates its own lower bound $n \lg n - n + 1$ between these points.

Keywords: MergeSort, sorting, worst case.

2010 MSC: 68W40 Analysis of algorithms

ACM Computing Classification

Theory of computation: Design and analysis of algorithms: Data structures design and analysis: Sorting and searching
Mathematics of computing: Discrete mathematics: Graph theory: Trees
Mathematics of computing: Continuous mathematics: Calculus
ACM classes: F.2.2; G.2.0; G.2.1; G.2.2

Contents

1. Introduction

`MergeSort` is one of the fundamental sorting algorithms that is being taught in undergraduate Computer Science curricula across the U.S. and elsewhere. Its worst-case performance, measured by the number of comparisons of keys performed while sorting them, is optimal for the class of algorithms that sort inductively[1] by comparisons of keys.[2] Historically, it[3] was the first sorting algorithm to run in $O(n \lg n)$ time[4].

So it seems only fitting to provide an exact formula for `MergeSort`'s worst-case performance *and* derive it precisely. Unfortunately, many otherwise decent texts offer unnecessarily imprecise[5] variants of it, and some with quite convoluted, incomplete, or incorrect proofs. Due to these imperfections, the fact that the worst-case performance of `MergeSort` is the same as that of another benchmark sorting algorithm, the *binary insertion sort* of [5], has remained unnoticed[6].

In this paper, I present two elementary yet precise and complete derivations of an exact formula

$$W(n) = \sum_{i=1}^{n} \lceil \lg i \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$$

for the maximum number $W(n)$ [7] of comparisons of keys performed by `MergeSort` on an n-element array. The first of the two, due to its structural regularity, is well worth carefully studying in its own right.

---

[1] Inductive sorting of n keys sorts a set of $n-1$ of those keys first, and then "sorts-in" the remaining n-th key.

[2] In its standard form analyzed in this paper, `MergeSort` is not an inductive sorting algorithm. However, its worst-case performance, measured by the number of comparisons of keys performed while sorting them, is equal to the worst-case performance of the *binary insertion sort* first described by Steinhaus in [5] that is worst-case optimal in the class of inductive sorting algorithms that sort by comparisons of keys; see [3] page 186.

[3] A bottom-up version of it that was invented by John Neumann.

[4] In the worst case.

[5] Notable exceptions in this category are [2] and [4] that derive almost exact formulas, but see Section 8 page 29 for a brief critique of the results and their proofs offered there.

[6] Even in [3].

[7] Elementary derivation of an exact formula for the *best*-case performance $B(n)$ of `MergeSort`, measured by the number of comparisons of keys performed while sorting them, has been done in [7]; see Section 9 page 32 of this paper.

Unlike some other basic sorting algorithms[8] that run in $O(n \lg n)$ time, `MergeSort` exhibits a remarkably regular[9] worst-case behavior, the elegant simplicity of which has been mostly lost on its rough analyses. In particular, $W(n)$ is linear[10] between the points $n = 2^{\lfloor \lg n \rfloor}$ and it linearly interpolates its own lower bound $n \lg n - n + 1$ [11] between these points.

2. Some Math prerequisites

$\mathbb{R}$ is the set of all reals. $\mathbb{Z}$ is the set of all integers. $\mathbb{N}$ is the set of all non-negative integers. The functions *floor*[12] and *ceiling* are defined, respectively, for every $x \in \mathbb{R}$ as follows:

$$\lfloor x \rfloor = \max\{k \in \mathbb{Z} \mid k \leq x\} \tag{1}$$

and

$$\lceil x \rceil = \min\{k \in \mathbb{Z} \mid x \leq k\}. \tag{2}$$

One can verify that for every $x \in \mathbb{R}$ and $n \in \mathbb{Z}$,

$$x \geq n \text{ if, and only if, } \lfloor x \rfloor \geq n, \tag{3}$$

$$x \leq n \text{ if, and only if, } \lceil x \rceil \leq n, \tag{4}$$

$$x > n \text{ if, and only if, } \lceil x \rceil > n, \tag{5}$$

$$\lceil \frac{n}{2} \rceil = \lfloor \frac{n+1}{2} \rfloor, \tag{6}$$

and

$$\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil = n. \tag{7}$$

Indeed, equalities (3) and (4) are direct consequences of definitions (1) and (2), respectively. Equality (5) is the contrapositive of (4). Equalities (6) and (7) can be verified by inspection for odd and even values of n.

---

[8] For instance, `Heapsort`; see [6] for a complete analysis of its worst-case behavior.

[9] As revealed by Theorem 5.2, page 18.

[10] See Figure 6 page 19.

[11] Given by the left-hand side of the inequality (28) page 18.

[12] Java applies *floor* to the result of evaluation of any expression of type `int` if that expression evaluates to a non-negative value; otherwise, it applies *ceiling*. In particular, if `n` is of type `int` and is non-negative then Java expression `n/2` is of type `int` and evaluates to $\lfloor \frac{n}{2} \rfloor$.

Here is a clever derivation of a well-known[13] closed-form formula for $\sum_{i=1}^{n}\lceil \lg i\rceil$. It proves insightful in my worst-case analysis of `MergeSort` as its right-hand side will occur on page 9 in the fundamental equality (12) and serve as an instrument to derive the respective exact formula for `MergeSort`'s worst-case behavior.

Lemma 2.1. *For every integer* $n \geq 1$,

$$\sum_{i=1}^{n}\lceil \lg i\rceil = \sum_{y=0}^{\lceil \lg n\rceil -1}(n-2^y). \tag{8}$$

Proof. The equality (8) may be neatly derived by inverting the function $y = \lceil \lg i\rceil$ (the result is a relation and *not* a function) and changing the control variable in the summation of the left-hand side of (8) from i to y, as it has been illustrated on Figure 1. One can notice that the shaded area is given by each side of (8); the left-hand side by adding the vertical rectangles and the right-hand side by adding the horizontal ones. □

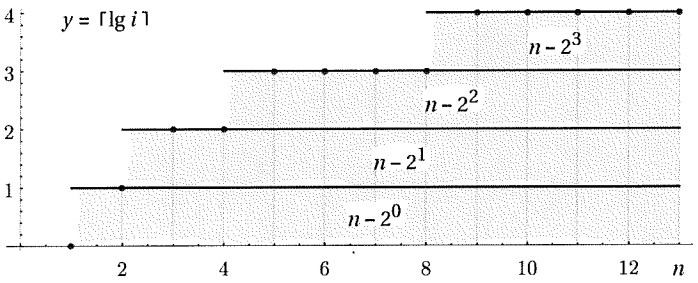

Figure 1: Computation of $\sum_{i=1}^{n}\lceil \lg i\rceil$ as $\sum_{y=0}^{\lceil \lg n\rceil -1}(n-2^y)$ for $n = 13$.

Corollary 2.2. *For every integer* $n \geq 1$,

$$\sum_{i=1}^{n}\lceil \lg i\rceil = n\lceil \lg n\rceil - 2^{\lceil \lg n\rceil}+1. \tag{9}$$

[13] See [3].

Proof. Observation that

$$\sum_{y=0}^{\lceil \lg n \rceil - 1} (n - 2^y) = \sum_{y=0}^{\lceil \lg n \rceil - 1} n - \sum_{y=0}^{\lceil \lg n \rceil - 1} 2^y = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1,$$

by virtue of (8), completes the proof. □

A 2-tree is a finite binary whose every non-leaf has two children.

Theorem 2.3. *Every non-empty 2-tree with m non-leaves has $m+1$ leaves.*

Proof. Every non-empty finite tree with k nodes has $k-1$ edges. Clearly, there are $2m$ edges in a 2-tree T with m non-leaves. Therefore, there are $2m+1$ nodes in T, $2m+1-m = m+1$ of which must be leaves. □

3. `MergeSort` and its worst-case behavior $W(n)$

A call to `MergeSort` inherits an n-element array `A` of integers and sorts it non-decreasingly, following the steps described below.

Algorithm `MergeSort` 3.1. *To sort an n-element array `A` do:*

1. *If $n \leq 1$ then return `A` to the caller,*
2. *If $n \geq 2$ then*
   (a) *pass the first $\lfloor \frac{n}{2} \rfloor$ elements of `A` to a recursive call to `MergeSort`,*
   (b) *pass the last $\lceil \frac{n}{2} \rceil$ elements of `A` to another recursive call to `MergeSort`,*
   (c) *linearly merge, by means of a call to `Merge`, the non-decreasingly sorted arrays that were returned from those calls onto one non-decreasingly sorted array `A'`,*
   (d) *return `A'` to the caller.*

A Java code of `Merge` is shown on the Figure 2.[14]

A typical measure of the running time of `MergeSort` is the number of *comparisons of keys*, which for brevity I call *comps*, that it performs while sorting array `A`.

[14] A Java code of `MergeSort` is shown in Appendix A Figure A.8 page 32.

```
70
71      private static int[] Merge(int[] A, int[] B)
72              // Merges two sorted arrays into one sorted array
73  {
74      int[] C = new int[A.length + B.length];
75      int indexA = 0, indexB = 0, indexC = 0;
76      while ((indexA < A.length) && (indexB < B.length))
77      {
78          if (A[indexA] < B[indexB] & Bcnt.incr()) // move A[indexA] to C
79              C[indexC++] = A[indexA++];
80          else // move B[indexA] to B
81              C[indexC++] = B[indexB++];
82      }
83      // copy the remaining part of A or B to C
84      if (indexA < A.length) // copy the remaining part of A
85          for (int i = indexA; i < A.length; i++)
86              C[indexC++] = A[i];
87      else // copy the remaining part of B
88          for (int i = indexB; i < B.length; i++)
89              C[indexC++] = B[i];
90  //    cnt2.incr();
91      return C;
92  }
93
```

Figure 2: A Java code of `Merge`, based on a pseudo-code from [1]. Calls to `Boolean` method `Bcnt.incr()` count the number of comps for the purpose of experimental verification of the worst-case analysis of `MergeSort`.

Definition 3.2. *The worst-case running time*

$$W(n)$$

of `MergeSort` *is defined as the maximum number of comps it performs while sorting an array of* n *distinct*[15] *elements.*

Clearly, if $n = 0$ then $W(n) = 0$. From this point on, I am going to assume that $n \geq 1$.[16]

Since no comps are performed outside `Merge`, $W(n)$ can be computed as the sum of numbers of comps performed by all calls to `Merge` during the execution of `MergeSort`. Moreover, the number of comps performed by `Merge` on two sorted list is maximal if, and only if, the last elements of those lists end up being compared to one another. This, of course, happens if, and only if, the two largest elements between those lists do not belong to the same list. In such a case, the number of comps performed by `Merge` on two sorted lists of sizes k and m, respectively, is equal to $k+m-1$ because each element

[15]This assumption is superfluous for the purpose of worst-case analysis as the mere presence of duplicates does not force `MergeSort` to perform more comps.

[16]This assumption turns out handy while using expression $\lg n$.

of those list, except for the largest one, loses one comparison to an element from another list before it is moved to the merged list (list C in the code of Figure 2). The above observations allow for a straightforward construction of a recursive program (its Java code is shown in the Appendix B) that unsorts given sorted array by breaking it on two sorted subarrays so that none contain two largest elements between them.[17]

This way we proved the following well-known fact:

Theorem 3.3. *The maximum number of comps performed by* Merge *on two sorted list of total number n of elements is $n-1$.*

Moreover, if the difference between the lengths of merged list is not larger than 1 then no algorithm that merges sorted lists by means of comps beats Merge in the worst case, that is, has a lower than $n-1$ maximum number of comps.[18]

4. An easy yet precise derivation of $W(n)$

MergeSort is a recursive algorithm. If $n \geq 2$ then it spurs a cascade of two or more recursive calls to itself. A rudimentary analysis of the respective recursion tree T_n, shown on Figure 3, yields a neat derivation of the exact formula for the maximum number $W(n)$ of comps that MergeSort performs on an n-element array.

The idea behind the derivation is strikingly simple. It is based on the observation[19] that for every $k \in \mathbb{N}$, the maximum number C_k of comps performed at each level[20] k of T_n is given by this neat formula:[21]

$$C_k = \max\{n - 2^k, 0\}. \tag{10}$$

[17]Here is an inductive construction of a worst-case permutation of $\{1,...,n\}$ for any $n \geq 1$. 1-element worst-case permutation of $\{1\}$ is a 1-element list $\langle 1\rangle$. For $n > 1$, in order to construct a worst-case permutation of $\{1,...,n\}$, when for all $m < n$ the worst-case permutations of $\{1,...,m\}$ have already been constructed, use the worst-case permutation $\langle a_1,...,a_{\lfloor \frac{n}{2} \rfloor}\rangle$ and the worst-case permutation $\langle b_1,...,b_{\lceil \frac{n}{2} \rceil}\rangle$ and construct from them the permutation $\langle 2a_1,...,2a_{\lfloor \frac{n}{2} \rfloor}, 2b_1-1,...,2b_{\lceil \frac{n}{2} \rceil}-1\rangle$ of $\{1,...,n\}$. A routine induction argument shows the correctness of the above construction.

[18]Proof in [3], Sec. 5.3.2 page 197.

[19]Which I will prove later as Theorem 4.6 on page 14.

[20]Empty or not.

[21]It is a simplification of formulas used in derivation presented in [2] and discussed in Section 8 page 29; in particular, it does not refer to the depth h of the decision tree T_n.

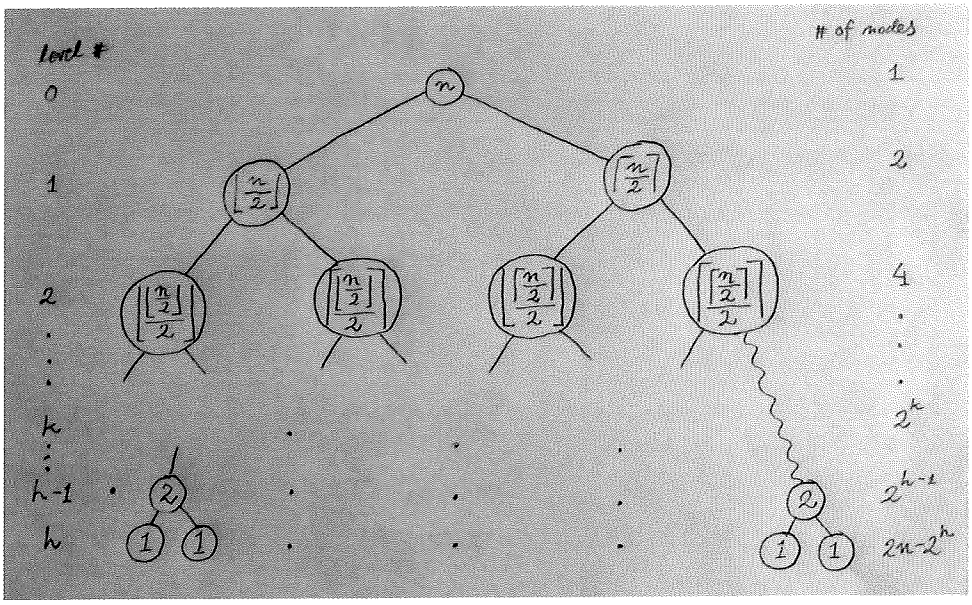

Figure 3: A sketch of the recursion 2-tree T_n for `MergeSort` for a sufficiently large n, with level numbers shown on the left and the numbers of nodes in the respective level shown on the right. The nodes correspond to calls to `MergeSort` and show sizes of (sub)arrays passed to those calls. The last non-empty level is h; it only contains nodes with value 1. (Level $h-1$ *may* also contain nodes with value 1 - not shown on the sketch - somewhere to the left of the last node with value 2.) The empty levels (all those numbered $> h$) are not shown. The root corresponds to the original call to `MergeSort`. If a call that is represented by a node p executes further recursive calls to `MergeSort` then these calls are represented by the children of p; otherwise p is a leaf. The wavy line $\sim\sim\sim$ represents a path in T_n.

Since $n - 2^k > 0$ if, and only if, $n > 2^k$ if, and only if, $\lg n > k$ if, and only if, by virtue of (5), $\lceil \lg n \rceil > k$ if, and only if, $\lceil \lg n \rceil - 1 \geq k$, that is,

$$n - 2^k > 0 \text{ if, and only if, } \lceil \lg n \rceil - 1 \geq k, \tag{11}$$

the Corollary 2.2 will allow me to conclude from (10) and (11) the main result of this paper:

$$W(n) = \sum_{k \in \mathbb{N}} C_k = \sum_{k=0}^{\lceil \lg n \rceil - 1} (n - 2^k) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1 = \sum_{i=1}^{n} \lceil \lg i \rceil. \tag{12}$$

A glimpse at Figure 1 page 5 reveals a remarkable structural regularity of the above derivation; a common part $\sum_{k=0}^{\lceil \lg n \rceil - 1}(n - 2^k)$ occurs both in that

Figure and in the equality (12), thus explaining how the sum $\sum_{i=1}^{n} \lceil \lg i \rceil$, quite obviously[22] the worst-case number of comps for the *binary insertion sort*, ended up as a the `MergeSort`'s worst-case running time $W(n)$. In terms of the graph shown on Figure 1, the worst-case formula $\sum_{i=1}^{n} \lceil \lg i \rceil$ for *binary insertion sort* ads the comps needed for consecutive binary insertions as areas of the corresponding vertical rectangles on that graph, while the worst-case formula $\sum_{k=0}^{\lceil \lg n \rceil - 1} (n - 2^k)$ for `MergeSort` adds them level-by-level of the recursive tree T_n.

The rest of this Section contains all[23] the missing details. Naturally, their only purpose is to prove the equality (10) for all $k \in \mathbb{N}$, as the rest, shown in (12), easily follows from it. The following observation is one of the three key steps towards this objective: since the number of comps that `Merge` needs in order to merge two sorted sub-arrays into one of size m is $m - 1$ in the worst case, the total number of comps performed in the worst case at any level of the recursion tree T_n is equal to the sum of sizes of all sub-arrays at that level minus 1 the number of nodes at that level. The second key step is to prove that at any level k, if $n > 2^k$ then the sum of the said sizes is n and the number of nodes at that level is 2^k, thus making $C_k = n - 2^k$. And the third step is to prove that if $n \leq 2^k$ then all the sizes of the sub-arrays, if any, at the level k are 1, thus making $C_k = 0$. The last two steps, once completed, will yield (10).[24]

Here are the details.

The nodes in the recursion tree T_n (visualized on Figure 3) correspond to calls to `MergeSort` and show sizes of (sub)arrays passed to those calls. The root corresponds to the original call to `MergeSort`. If a call that is represented by a node p executes further recursive calls to `MergeSort` then

[22] It helps to remember that, for any positive integer i, $\lfloor \lg i \rfloor + 1 = \lceil \lg(i+1) \rceil$ in order to see that binary insertion of an i-th key into the already sorted $i - 1$ keys takes $\lceil \lg i \rceil$ comps in the worst case.

[23] Some of them could have been omitted for the sake of brevity; virtually all could have been replaced with brief intuitive sketches.

[24] A "clever" way to accomplish the same would be to get rid of the condition $n > 2^k$ in the second step and to replace the last step with an intuitively appealing observation that C_k cannot be negative, trying to conclude (10) from that fact; this, however, would be an invalid inference.

these calls (two, to be exact) are represented by the children of p; otherwise p is a leaf. Thus, T_n is a 2-tree[25].

The levels in tree T_n are enumerated from 0 to ∞. h is the number of the last non-empty level of the tree, or - in other words - the depth of T_n. On Figure 3, the level numbers, up to h, are shown on the left side of the tree. The root is at the level 0, its children (if any) are at level 1, its grand children (if any) are at level 2, its great grand children (if any; not shown on the sketch) are at level 3, and so on. Clearly, since every call to `MergeSort` on a sub-array of size ≥ 2 executes two further recursive calls to `MergeSort` and the calls to a sub-array of size 1 execute none, only the nodes that show value 1 are the leaves and all other nodes have 2 children each. Thus, since all nodes in the last non-empty level h are leaves, they all show value 1. And since the original input array gets split, eventually, onto n 1-element sub-arrays, the number of all leaves in T_n is n. (This, however, does not mean that the last level h necessarily contains all the leaves of T_n.)

I will use the following *sequence notation*:

$$\langle a_i \rangle_{i=n}^{m} = \begin{cases} \langle\rangle \text{ if } n > m \\ \\ \langle a_n, ..., a_m \rangle \text{ otherwise,} \end{cases} \tag{13}$$

and denote by

$$\langle a_i \mid \varphi(i) \rangle_{i=n}^{m} \tag{14}$$

the subsequence of $\langle a_i \rangle_{i=n}^{m}$ consisting of all those, and only those, elements a_i of $\langle a_i \rangle_{i=n}^{m}$ for which the condition $\varphi(i)$ is true.

The following Lemma characterizes exactly what values are shown at the nodes in the level k of the recursion tree T_n. It allows for *brute-force*, straightforward derivation of the formula (10).

The Main Lemma 4.1. *The sizes of sub-arrays at any level* [26] k *of the recursion tree* T_n *are given by this sequence*

$$\sigma_k = \langle \lfloor \frac{n+i}{2^k} \rfloor \mid n + (i \bmod 2^{k-1}) \geq 2^k \rangle_{i=0}^{2^{k-1}}, \quad [27] \tag{15}$$

[25] A binary tree whose every non-leaf has exactly 2 children.

[26] Empty or not.

[27] The condition $n+(i \mod 2^{k-1}) \geq 2^k$ in the sequence (15) is the continuation condition for the recursion within the `MergeSort`. It assures that the parent subarray that was split and sent to the corresponding recursive call had at least two elements.

not necessarily in this order.

An example of the recursion tree T_n is shown on Figure 4.

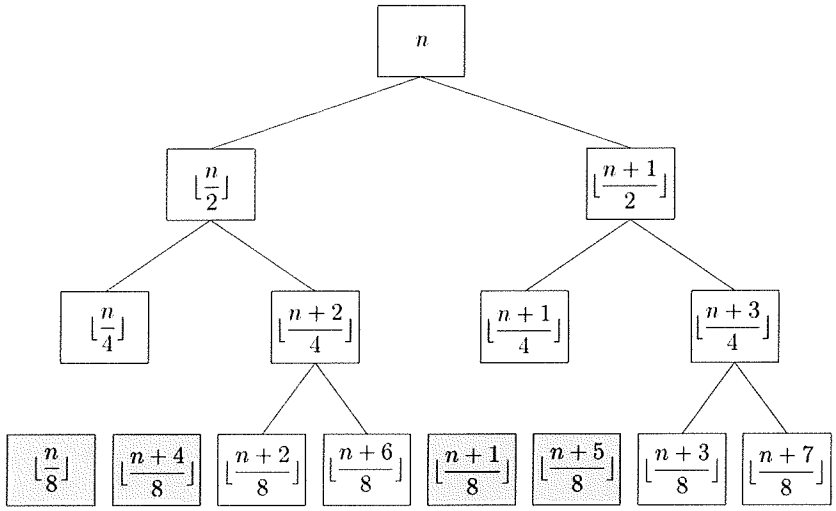

Figure 4: A complete example of the recursion tree T_6 ($n = 6$) derived from the sketch on Figure 3, with the equality (6) applied recursively to the nodes of the tree. Nodes shaded gray are missing in the last nonempty level of T_6 (level 3) because the condition $n + (i \bmod 2^{k-1}) \geq 2^k$ in (15) is false for those missing nodes.

Proof. By induction on k.
Basis step. $k = 0$.

$$\sigma_0 = \langle \lfloor \frac{n+i}{2^0} \rfloor \mid n + (i \bmod 2^{0-1}) \geq 2^0 \rangle_{i=0}^{2^0-1} = \langle \lfloor \frac{n}{1} \rfloor \mid n \geq 1 \rangle = \langle n \rangle.^{28}$$

Since the level 0 contains one node that shows value n, this completes the basis step.
Inductive step. Assume that the sizes of sub-arrays at level k are given by the sequence σ_k of the equality (15) for some $k \geq 0$ (the *inductive hypothesis*).

[28] Recall the assumption $n \geq 1$ of page 7.

Prove that the sizes of sub-arrays at level $k+1$ are given by this sequence

$$\sigma_{k+1} = \langle \lfloor \frac{n+i}{2^{k+1}} \rfloor \mid n + (i \bmod 2^k) \geq 2^{k+1} \rangle_{i=0}^{2^{k+1}-1}. \tag{16}$$

From the inductive hypothesis and the description of `MergeSort` on page 6, Algorithm 3.1 , step 2, we conclude that since any node at the level k in the recursion tree T_n has children if, and only if, it shows value $\lfloor \frac{n+i}{2^k} \rfloor \geq 2$, the sizes of sub-arrays at level $k+1$ are given by this sequence:

$$\langle \lfloor \frac{\lfloor \frac{n+i}{2^k} \rfloor}{2} \rfloor, \lfloor \frac{\lfloor \frac{n+i}{2^k} \rfloor + 1}{2} \rfloor \mid \lfloor \frac{n+i}{2^k} \rfloor \geq 2 \wedge n + (i \bmod 2^{k-1}) \geq 2^k \rangle_{i=0}^{2^k-1} =$$

[by $\lfloor \frac{\lfloor \frac{n+i}{2^k} \rfloor}{2} \rfloor = \lfloor \frac{n+i}{2^{k+1}} \rfloor$, $\lfloor \frac{\lfloor \frac{n+i}{2^k} \rfloor + 1}{2} \rfloor = \lfloor \frac{\lfloor \frac{n+i+2^k}{2^k} \rfloor}{2} \rfloor = \lfloor \frac{n+i+2^k}{2^{k+1}} \rfloor$, and $\lfloor \frac{n+i}{2^k} \rfloor \geq 2 \equiv$ (by (3)) $n + i \geq 2^{k+1} \equiv n + i - 2^k \geq 2^k \Rightarrow n + (i \bmod 2^{k-1}) \geq 2^k$ for any $0 \leq i < 2^k$]

$$= \langle \lfloor \frac{n+i}{2^{k+1}} \rfloor, \lfloor \frac{n+i+2^k}{2^{k+1}} \rfloor \mid n + i \geq 2^{k+1} \rangle_{i=0}^{2^k-1},$$

which [since $0 \leq i < 2^k$ implies $(i+2^k) \bmod 2^k = i$] is equal to, not necessarily in this order, the sequence σ_{k+1} of the equality (16). This completes the inductive step. □

I will use the following immediate consequences of the Lemma 4.1.

Lemma 4.2. *For every k with $n \geq 2^k$, the sizes of sub-arrays at level k of the recursion tree T_n are given by this sequence*

$$\sigma_k = \langle \lfloor \frac{n+i}{2^k} \rfloor \rangle_{i=0}^{2^k-1}, \tag{17}$$

not necessarily in this order.

Proof. If $n \geq 2^k$ then for any $i \geq 0$,

$$n + (i \bmod 2^{k-1}) \geq n \geq 2^k.$$

Thus the condition $n+(i \bmod 2^{k-1}) \geq 2^k$ in the sequence σ_k of (15) is satisfied for all i and (17) holds. □

Lemma 4.3. *Every level k with $n \geq 2^k$ of the recursion tree T_n contains 2^k nodes.*

Proof. The length of the sequence σ_k of (17) is 2^k, from which observation the thesis of this Lemma follows. □

Lemma 4.4. *For every k with $n \leq 2^k$, the sizes of sub-arrays at level k of the recursion tree T_n are given by the empty sequence $\sigma_k = \langle\rangle$ or by this sequence*

$$\sigma_k = \langle 1, ..., 1 \rangle. \tag{18}$$

Proof. If $n \leq 2^k$ then for any i with $0 \leq i \leq 2^k - 1$,

$$\lfloor \frac{n+i}{2^k} \rfloor \leq \lfloor \frac{2^k + 2^k - 1}{2^k} \rfloor \leq \lfloor 2 - \frac{1}{2^k} \rfloor = 1. \tag{19}$$

If a node with value $\lfloor \frac{n+i}{2^k} \rfloor$ occurs at the level k of T_n then from the condition $n+(i \bmod 2^{k-1}) \geq 2^k$ in (15) we conclude that $n+i \geq n+(i \bmod 2^{k-1}) \geq 2^k$ and $\lfloor \frac{n+i}{2^k} \rfloor \geq 1$. Thus, by (19), $\lfloor \frac{n+i}{2^k} \rfloor = 1$ so if σ_k is non-empty then it is equal to $\langle 1, ..., 1 \rangle$. □

Here is a known technical result that indicates sizes of elements of the "even" partition of an n-element set onto m sets.[29] For $m = 2^k$, it states intuitively obvious fact that the sum of sizes of sub-arrays at each full level k [30] of the recursion tree T_n is n. Its special case for $m = 2$, namely $\lfloor \frac{n}{2} \rfloor + \lfloor \frac{n+1}{2} \rfloor = n$, is a direct consequence of the equalities (6) and (7) page 4.

Theorem 4.5. *For every natural number n and every positive natural number m,*

$$\sum_{i=0}^{m-1} \lfloor \frac{n+i}{m} \rfloor = n. \tag{20}$$

Proof. See, for instance, Theorem Appendix B.0.5 in [7]. □

At this point, the derivation of (10) page 8 becomes an easy exercise.

Theorem 4.6. *The maximum number C_k of comps performed at each level k of the recursion tree T_n is given by the formula* (10).

---

[29] The sum of the differences between the sizes those m sets is minimal; the inequality (46) page 25 applies.

[30] A level k that contains 2^k nodes.

Proof. Recall, that by the Theorem 3.3, the maximum number of comps performed at node p that shows a value of m is $m-1$. Thus if $n > 2^k$ then, by the Lemmata 4.2 and 4.3,

$$C_k = \sum_{i=0}^{2^k-1}(\lfloor\frac{n+i}{2^k}\rfloor - 1) = \sum_{i=0}^{2^k-1}\lfloor\frac{n+i}{2^k}\rfloor - 2^k =$$

[by (20), put $m = 2^k$]

$$= n - 2^k.$$

If $n \leq 2^k$ then $C_k = 0$ if level k is empty or, by Lemma 4.4, is

$$C_k = \sum(1-1) = 0.$$

Thus $C_k = 0$ in this case.

Hence, (10) holds. □

This concludes the derivation of the main result (12) stated on page 9. This way I have proved the following.

The Main Theorem 4.7. *The number $W(n)$ of comparisons of keys that* `MergeSort` *performs in the worst case while sorting an n-element array is*

$$W(n) = \sum_{i=1}^{n}\lceil\lg i\rceil = n\lceil\lg n\rceil - 2^{\lceil\lg n\rceil} + 1. \tag{21}$$

Proof follows from the above derivation. □

From that we can conclude a usual rough characterization of $W(n)$:

$$W(n) \leq n(\lg n + 1) - 2^{\lg n} + 1 = n\lg n + n - n + 1 = n\lg n + 1$$

and

$$W(n) \geq n\lg n - 2^{\lg n+1} + 1 = n\lg n - 2n + 1.$$

Therefore,

$$W(n) \in \Theta(n\log n).$$

$\sum_{i=1}^{n}\lceil\lg i\rceil$ allows to conclude that $W(n)$ is exactly equal[31] to the number of comparisons of keys that the *binary insertion sort*, considered by H. Steinhaus in [5] and analyzed in [3], performs in the worst case. Since the *binary*

[31] [3] contains no mention of that fact.

insertion sort is known to be worst-case optimal[32] in the class of algorithms that perform incremental sorting, `MergeSort` is worst-case optimal in that class[33], too.

5. Close smooth bounds on $W(n)$

Our formula for $W(n)$ contains a function ceiling that is harder to analyze than arithmetic functions and their inverses. In this subsection, I will derive close lower and upper bounds on $W(n)$ that are expressible by simple arithmetic formulas. I will show that these bounds are the closest to $W(n)$ in the class of functions of the form $n \lg n + cn + 1$, where c is a real constant.

Using the function ε (analyzed briefly in [3] and [6]), a form of which is shown on Figure 5, given by:

$$\varepsilon = 1 + \theta - 2^{\theta} \text{ and } \theta = \lceil \lg\, n \rceil - \lg\, n, \tag{22}$$

one can conclude[34] that, for every $n > 0$,

$$n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} = n(\lg n + \varepsilon - 1), \tag{23}$$

which yields

$$W(n) = n(\lg n + \varepsilon - 1) + 1 = n \lg n + (\varepsilon - 1)n + 1. \tag{24}$$

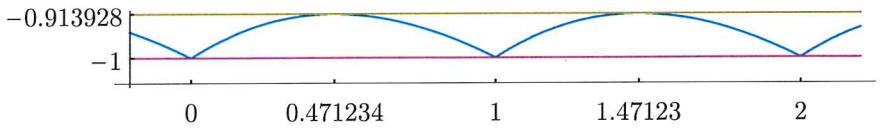

Figure 5: Graph of $\varepsilon - 1$ as a function of $\lg n$.

[32] With respect to the number of comparisons of keys performed.
[33] Although it is not a member of that class.
[34] See [6], Thm. 12.2 p. 94 for a proof.

Property 5.1. *Function ε given by* (22) *is a continuous function of n on the set of reals > 0. It assumes the minimum 0 for every $n = 2^{\lfloor \lg n \rfloor}$ and the maximum*

$$\delta = 1 - \lg e + \lg \lg e \approx 0.0860713320559342,^{35} \tag{25}$$

for every

$$n = 2^{\lfloor \ln n + \lg \lg e \rfloor} \ln 2 \tag{26}$$

and only such n. The function ε restricted to integers never reaches the value δ. However, δ is the supremum of ε restricted to integers.

Proof. Let's consider ε a function of $y = \lg n$, given on $\mathbb{R}$ by

$$\varepsilon(y) = 1 + (\lceil y \rceil - y) - 2^{\lceil y \rceil - y}.$$

Since $\varepsilon(y)$ is a periodic function with period 1, it suffices to prove the desired properties of it on the closed interval $[0, 1]$. Continuity of $\varepsilon(y)$ on the open interval $(0, 1)$ is obvious. Continuity of $\varepsilon(y)$ at $y = 0$ and $y = 1$ follows from the fact $\lim_{y \to 0^+} \varepsilon(y) = 0 = \lim_{y \to 1^-} \varepsilon(y)$. The derivative $\varepsilon'(y) = 2^{1-y} \ln 2 - 1$ of $\varepsilon(y)$ exists on $(0, 1)$ and is equal to 0 at one point $y_0 = 1 - \lg \lg e$ [36] $\approx 0.4712336270551024$ only. Since $\varepsilon(0) = 0 = \varepsilon(1)$ and $\varepsilon(y_0) = 1 - \lg e + \lg \lg e$ [37] $\approx 0.0860713320559342$, the minimum of $\varepsilon(y)$ on $\mathbb{R}$ is 0 and is assumed for all integer y, and only such y, and its maximum on $\mathbb{R}$ is δ and is assumed for $y = k - \lg \lg e$, where k is any integer, that is, for

$$y = \lfloor y + \lg \lg e \rfloor - \lg \lg e, \tag{27}$$

and only such y.

Let's go back to the function ε of $n = 2^y$. Clearly, it is continuous, since $\varepsilon(y)$ is, and has the minimum 0 assumed for every $n = 2^{\lfloor \lg n \rfloor}$ and the maximum δ assumed, by virtue of (27), for $n = 2^{\lfloor y + \lg \lg e \rfloor - \lg \lg e} = \frac{2^{\lfloor \ln n + \lg \lg e \rfloor}}{2^{\lg \lg e}}$ $= 2^{\lfloor \ln n + \lg \lg e \rfloor} \ln 2$, which yields (26). Since $\ln 2$ is an irrational number[38], so

---

[35] The constant $1 - \lg e + \lg \lg e$ has been known as the *Erdös constant* δ. Erdös used it around 1955 in order to establish an asymptotic upper bound for the number $M(k)$ of different numbers in a multiplication table of size $k \times k$ by means of the following limit:

$$\lim_{k \to \infty} \frac{\ln \frac{k \times k}{M(k)}}{\ln \ln (k \times k)} = \delta.$$

[36] Proof by direct inspection.

[37] By direct computation.

[38] Here, I only use the fact that $\ln 2$ does not have finite binary representation.

is every n that satisfies (26). In particular, no such n is an integer. Thus the function ε restricted to integers never reaches the value δ. However, one can easily show that $\lim_{k\to\infty}\varepsilon(\lfloor 2^k \ln 2\rfloor) = \delta$,[39] which makes δ the *supremum* of ε restricted to integers. □

Characterization (24) and Property 5.1 yield close smooth bounds of $W(n)$. They are both of the form $n\lg n + cn + 1$ and they sandwich tightly $W(n)$ between each other. If one sees $W(n)$ as an infinite polygon[40], its lower bound circumscribes it and its upper bound inscribes it.

Theorem 5.2. *$W(n)$ is a continuous concave function, linear between the points $n = 2^{\lfloor \lg n\rfloor}$, that for every $n > 0$ satisfies this inequality:*

$$n\lg n - n + 1 \le W(n) \le n\lg n - (1-\delta)n + 1 < n\lg n - 0.913n + 1, \quad (28)$$

with the left $\le$ becoming $=$ for every $n = 2^{\lfloor \lg n\rfloor}$ and the right $\le$ becoming $=$ for every $n = 2^{\lfloor \lg n + \lg\lg e\rfloor}\ln 2$, and only for such n. Moreover, the graph of $W(n)$ is tangent to the graph of $n\lg n - (1-\delta)n + 1$ at the points $n = 2^{\lfloor \lg n + \lg\lg e\rfloor}\ln 2$, and only at such points.

Proof. Continuousness of $W(n)$ follows from (24) and continuousness of ε ascertained by the Property 5.1. That it is linear between the points $n = 2^{\lfloor \lg n\rfloor}$ and concave, follows directly from (21). Inequality (28) follows from (24) and the miminum and maximum of ε established in the Property 5.1, as does the rest of Theorem 5.2. □

The bounds given by (28) are really close[41] to the exact value of $W(n)$, as it is shown on Figure 6 page 19. The exact value $n\lceil \lg n\rceil - 2^{\lceil \lg n\rceil} + 1$ is a continuous function (if n is interpreted as a real variable) despite that it incorporates discontinuous function *ceiling*.

---

[39] Indeed, $0 \le 2^k \ln 2 - \lfloor 2^k \ln 2\rfloor \le 1$, while $\lim_{k\to\infty} 2^k \ln 2 = \infty$ and ε is a continuous function of n on the set of reals > 0 differentiable on its domain minus the countable set of $n = 2^{\lfloor \lg n\rfloor}$ and $\lim_{2^{\lfloor \lg n\rfloor}\ne n\to\infty}\varepsilon'(n) = 0$, so that the $\lim_{k\to\infty}(\varepsilon(2^k \ln 2) - \varepsilon(\lfloor 2^k \ln 2\rfloor)) = 0$. Thus $\lim_{k\to\infty}\varepsilon(\lfloor 2^k \ln 2\rfloor) = \lim_{k\to\infty}\varepsilon(2^k \ln 2) = \delta$.

[40] Which it is.

[41] The distance between them is less than $\delta n \approx 0.0860713320559342n$ for any positive integer n.

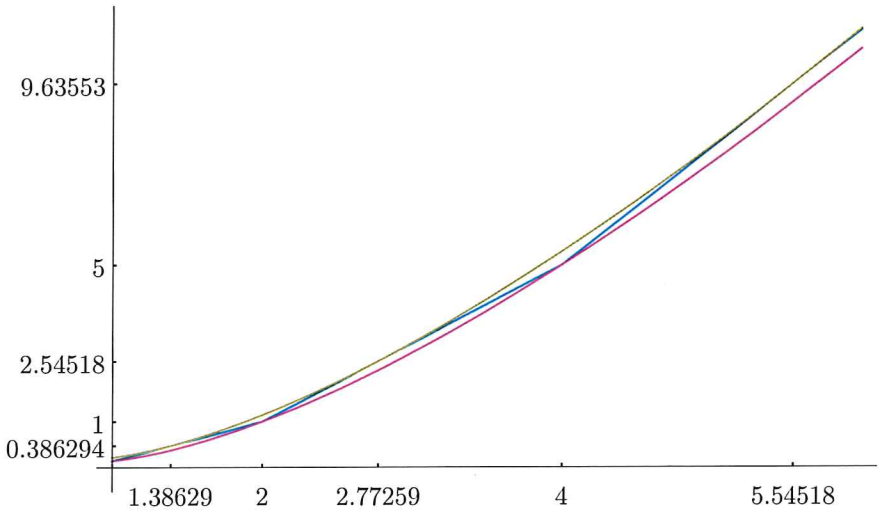

Figure 6: $W(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$ (the middle line) and its bounds $n \lg n - n + 1$ and $n \lg n - (1-\delta)n + 1 \approx n \lg n - 0.913n + 1$, all three treated as functions of a positive real variable n, plotted for $n \in [1, 6]$. $W(n)$ is linear between the points $n = 2^{\lfloor \lg n \rfloor}$ and it linearly interpolates its lower bound $n \lg n - n + 1$ between these points. Its upper bound $n \lg n - (1-\delta)n + 1$ inscribes it and is tangent to it at the points $n = 2^{\lfloor \lg n + \lg \lg e \rfloor} \ln 2$.

Note 5.3. *It seems interesting that* $W(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$ *(whether* n *is interpreted as a real variable or an integer variable) is linear between points* $n = 2^{\lfloor \lg n \rfloor}$ *and linearly interpolates its own lower bound* $n \lg n - n + 1$ *between these points.*

For n restricted to positive integers, the inequality (28) can be slightly enhanced by replacing the $\leq$ symbol with $<$, with the following result.

Theorem 5.4. $1 - \delta$ *is the greatest constant* c *such that for every integer* $n \geq 1$,

$$W(n) < n \lg n - cn + 1. \tag{29}$$

Proof. Proof follows from the fact, stated by Property 5.1, that δ is a supremum of ε restricted to integers. □

Theorem 5.4 can be reformulated as follows.

Corollary 5.5.

$$\inf\{c \in \mathbb{R} \mid \forall n \in \mathbb{N} \setminus \{0\}, W(n) < n \lg n - cn + 1\} = 1 - \delta. \tag{30}$$

Proof. Proof follows directly from Theorem 5.4 and the definition of *infimum.* □

No upper bound of $W(n)$ that has a form $n \lg n - cn + 1$ can coincide with $W(n)$ at any integer n, as the following fact ascertains.

Corollary 5.6. *There is no constant c such that for every integer $n \geq 1$,*

$$W(n) \leq n \lg n - cn + 1 \tag{31}$$

and for some integer $n \geq 1$,

$$W(n) = \lg n - cn + 1. \tag{32}$$

Proof. If c satisfies (31) then, by Theorem 5.4,

$$c \leq 1 - \delta, \tag{33}$$

for otherwise a constant

$$d = \frac{c + 1 - \delta}{2}$$

would satisfy, for every integer $n \geq 1$,

$$c > d > 1 - \delta \text{ and } W(n) < n \lg n - dn + 1,$$

which would contradict Theorem 5.4. From (29) and (33) we infer the negation of (32). This conclusion completes the proof. □

In particular[42],

$$\inf\{c \in \mathbb{R} \mid \forall n \in \mathbb{N} \setminus \{0\}, W(n) \leq n \lg n - cn + 1\} = 1 - \delta. \tag{34}$$

Moreover, we can conclude from Theorem 5.4 the following fact.

Corollary 5.7. *$1 - \delta$ is the greatest constant c such that for every integer $n \geq 1$,*

$$W(n) \leq \lceil n \lg n - cn \rceil. \tag{35}$$

[42]Note the $\leq$ symbol in (34).

Proof. By the equivalence (5) page 4,

$$W(n) < n\lg n - cn + 1 \text{ iff } W(n) < \lceil n\lg n - cn + 1\rceil = \lceil n\lg n - cn\rceil + 1.$$

Hence,

$$W(n) < n\lg n - cn + 1 \text{ iff } W(n) \leq \lceil n\lg n - cn\rceil.$$

Thus Theorem 5.4 implies Corollary 5.7. □

Since for any integer $n \geq 1$, $W(n)$ is integer, the lower bound given by (28) yields, by virtue of the equivalence (4) page 4,

$$W(n) \geq \lceil n\lg n - n + 1\rceil = \lceil n\lg n\rceil - n + 1. \tag{36}$$

Combining (35) and (36) yields, for every integer $n \geq 1$,

$$\lceil n\lg n\rceil - n + 1 \leq W(n) \leq \lceil n\lg n - (1-\delta)n\rceil, \tag{37}$$

and

$$\lceil n\lg n\rceil - n + 1 \leq W(n) \leq \lceil n\lg n - 0.913n\rceil. \tag{38}$$

By virtue of Corollary 5.7, for some integers $n \geq 1$,[43]

$$W(n) > \lceil n\lg n - 0.914n\rceil. \tag{39}$$

Although the bounds given by (38) [44] are tighter than those given by (28), they nevertheless involve the discontinuous *ceiling* function, so that they may not be as easy to visualize or analyze as some differentiable functions, thus losing their advantage over the precise formula $W(n) = n\lceil\lg n\rceil - 2^{\lceil\lg n\rceil} + 1$. Therefore, the bounds given by (28) appear to have an analytic advantage over those given by (38).

6. Other properties of the recursion tree T_n

This sections contains some well-known auxiliary facts that I didn't need for the derivation of the exact formula for $W(n)$ but am going to derive from the Lemma 4.1 for the sake of a thoroughness of my analysis of the decision tree T_n.

[43] For instance, for $n = 11$.

[44] Almost the same bounds were given in [2]; see Section 8 for more details on this.

Theorem 6.1. *The depth h of the recursion tree T_n is*

$$h = \lceil \lg n \rceil. \tag{40}$$

Proof. As I have mentioned above, h is the level number of the last non-empty level of T_n, or - in other words - the largest integer k for which the sequence (15) is non-empty. For that to happen, it is necessary and sufficient that h is the largest integer k that satisfies at least one condition in the sequence (15), that is, h is the largest integer that satisfies

$$(\exists i \leq 2^h - 1)(n + (i \bmod 2^{h-1}) \geq 2^h). \tag{41}$$

Since $i \bmod 2^{h-1}$ is maximal if $i = 2^h - 1$, in which case it has a value of $2^{h-1} - 1$, formula (41) is equivalent to

$$n + 2^{h-1} - 1 \geq 2^h,$$

or

$$n - 1 \geq 2^h - 2^{h-1},$$

or

$$n > 2^{h-1},$$

or

$$\lg n > h - 1,$$

or, by (5),

$$\lceil \lg n \rceil > h - 1,$$

or

$$\lceil \lg n \rceil \geq h. \tag{42}$$

Thus, since h is the largest integer that satisfies (42), the equality (40) is true. □

Note 6.2. *Theorem 6.1 allows for quick derivation of fairly close upper bound on the number of comps performed by* `MergeSort` *on an n-element array. Since at each level of T_n less than n comparisons are performed by* `Merge` *and at level h no comps are performed, and there are $h = \lceil \lg n \rceil$ levels below level h, the total number of comps is not larger than*

$$(n-1)h = (n-1)(\lceil \lg n \rceil) < (n-1)(\lg n + 1) \in O(n \log n). \tag{43}$$

A *cut* of a tree T_n is a set Γ of nodes of T such that every branch[45] in T_n has exactly one element in Γ.

Theorem 6.3. *The sum of values shown at the elements of any cut of T_n is n.*

Proof. The thesis follows from the equality (7) and the fact that for every tree T of height not larger than ω [46] whose nodes are labeled with numbers in such a way that the label of any parent is the sum of labels of its children, the sum of labels of the elements of any cut of T is equal to the label of the root of T. □

Theorem 6.4. *The number of leaves in the recursion tree T_n is n.*

Proof. Since all values shown at nodes of T_n are positive integers and for every $k \geq 2$, $\lfloor \frac{k}{2} \rfloor \leq \lceil \frac{k}{2} \rceil < k$, every path in T_n is finite. Thus the set of leaves of T_n is a cut of T_n. Since each leaf of T_n shows value 1, the number of leaves in T_n is equal to the sum of values shown at the leaves of T_n. Application of Theorem 6.3 completes the proof. □

The following corollary provides some statistics about recursive calls to `MergeSort`.

Corollary 6.5. *For every integer $n > 0$,*

(i) *T_n has $2n - 1$ nodes.*

(ii) *The number or recursive calls spurred by* `MergeSort` *on any n-element array is $2(n - 1)$.*

(iii) *The sum S_n of all values shown in the recursion tree T_n on Figure 3 is equal to:*

$$S_n = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 2n = n(\lg n + \varepsilon + 1). \tag{44}$$

(iv) *The average size A_n of array passed to any recursive call to* `MergeSort` *while sorting an n-element array is:*

$$A_n = \frac{1}{2}(1 + \frac{1}{n-1})(\lg n + \varepsilon) \approx \frac{1}{2}(\lg n + \varepsilon). \tag{45}$$

[45] A maximal path.

[46] The least infinite ordinal.

Proof. (i) By virtue of Theorems 2.3, page 6, and 6.4, T_n is a 2-tree with n leaves and it has $n-1$ non-leaves and a total of $2n-1$ nodes.
(ii) By virtue of (i), the number or recursive calls spurred by `MergeSort` on any n-element array is $2(n-1)$.
(iii) By virtue of (i) and Theorems 4.6, page 14, and 6.4, the sum S_n of all values shown in the recursion tree T_n on Figure 3 is equal to:

$$S_n = \sum_{k=0}^{\infty} C_k + 2n - 1 = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 2n.$$

Applying (23) page 16 to the above yields:

$$S_n = n(\lg n + \varepsilon - 1) + 2n = n(\lg n + \varepsilon + 1).$$

(iv) By virtue of (iii), the average size A_n of array passed to any call to `MergeSort` while sorting an n-element array is, by virtue of Theorems 2.3 and 6.4, excluding the size n of the array passed to the original call to `MergeSort` and applying (ii), is:

$$A_n = \frac{n(\lg n + \varepsilon + 1) - n}{2(n-1)} = \frac{1}{2}(1 + \frac{1}{n-1})(\lg n + \varepsilon) \approx \frac{1}{2}(\lg n + \varepsilon).$$

□

Here is a very insightful property. It states that `MergeSort` is splitting its input array fairly evenly[47] so that at any level of the recursive tree, the difference between the lengths of the longest sub-array and the shortest sub-array is ≤ 1. This fact is the root cause of good worst-case performance of `MergeSort`.

Property 6.6. *The difference between values shown by any two nodes in the same level of of the recursion tree T_n is ≤ 1.*

Proof. [48] Clearly, by virtue of the Lemma 4.1, the value $\lfloor \frac{n+i}{2^k} \rfloor$ shown at any node of level k of T_n must satisfy

$$\lfloor \frac{n}{2^k} \rfloor \leq \lfloor \frac{n+i}{2^k} \rfloor \leq \lfloor \frac{n+2^k-1}{2^k} \rfloor.$$

[47] The sizes of the sub-arrays passed to recursive calls at any non-empty level k of the decision tree T_n above the last non-empty level h are the same as the sizes of the elements of the maximally even partition of an n-element set onto 2^k subsets.

[48] A direct proof of the the Property 6.6 without any reference to the Lemma 4.1 has been archived in [7], Property Appendix A.0.1. page 33.

Since $\lfloor\frac{n+2^k-1}{2^k}\rfloor = \lfloor\frac{n-1}{2^k}\rfloor + 1 \leq \lfloor\frac{n}{2^k}\rfloor + 1$,

$$\lfloor\frac{n}{2^k}\rfloor \leq \lfloor\frac{n+i}{2^k}\rfloor \leq \lfloor\frac{n}{2^k}\rfloor + 1 \tag{46}$$

and the thesis of the Property follows. □

Property 6.6 has this important consequence that `Merge` is, by virtue of the observation on page 8 after the Theorem 3.3 page 8, worst-case comparison-optimal while merging any two sub-arrays of the same level of the recursion tree. Thus the worst-case of `MergeSort` cannot be improved just by replacing `Merge` with some tricky merging `X` as long as `X` merges by means of comparisons of keys.

Corollary 6.7. *Replacing* `Merge` *with any other method that merges sorted arrays by means of comps will not improve the worst-case performance of* `MergeSort` *measured with the number of comps while sorting an array.*

Proof. Proof follows from the above observation. □

Since a parent must show a larger value than any of its children, the Property 6.6 has also the following consequence.

Corollary 6.8. *The leaves in the recursion tree T_n can only reside at the last two non-empty levels of T_n.*

Proof. Proof follows from the Property 6.6 as the above observation indicates. □

As a result, one can conclude[49] that the recursion tree T_n has the miminum *internal* and *external path length*s among all binary trees on $2n-1$ nodes.

Since all nodes at the level h of the recursion tree T_n are leaves and show value 1, no node at level $h-1$ can show a value > 2. Indeed, level $h-1$ may only contain leaves, that show value 1, and parents of nodes of level h that show value $1+1=2$. This observation and the previous result allow for easy characterization of contents of the last two non-empty levels of tree T_n.

[49] Cf. [3], Sec. 5.3.1 Ex. 20 page 195.

Corollary 6.9. *For every $n \geq 2$:*

(i) *there are $2^h - n$ leaves, all showing value 1, at the level $h-1$,*
(ii) *there are $n - 2^{h-1}$ non-leaves, all showing value 2, at the level $h-1$, and*
(iii) *there are $2n - 2^h$ [50] nodes, all leaves showing value 1, at the level h*

of the recursion tree T_n, where h is the depth[51] of T_n.

Proof. Let x be the number of leaves at the level $h-1$ and x be the number of leaves at the level h. So far, by Corollary 6.8, we know that only levels $h-1$ and h can contain leaves. From this we infer that all levels from 0 to $h-1$ are maximal. In particular, level $h-1$ has 2^{h-1} nodes. Therefore, there are $2^{h-1} - x$ non-leaves at level $h-1$. The above observations allow yield the following equations:

$$x + y = n, \tag{47}$$

and

$$x + 2(2^{h-1} - x) = n. \tag{48}$$

Indeed, (47) is true by virtue of the Theorem 6.4. Since the level $h-1$ is maximal, it is a cut of T_n. Hence, (48) is true by virtue of the Theorem 6.3. Solving (47) and (48) for x and y yields:

$$x = 2^h - n,$$

$$2^{h-1} - x = 2^{h-1} - (2^h - n) = n - 2^{h-1},$$

and

$$y = 2n - 2^h.$$

The above, together with the observation that all non-leaves et level $h-1$ show value 2, complete the proof of (i), (ii), and (iii). □

[50] This value shows in the lower right corner of Figure 3 page 9 of a sketch of the recursion tree T_n; it was not need needed for the derivation of the main result (21) page 15, included for the sake of completeness only.

[51] The level number of the last non-empty level of T_n.

7. A derivation of $W(n)$ without references to the recursion tree

In order to formally prove Theorem 4.7 without any reference to the recursion tree, I use here the well-known[52] recurrence relation

$$W(n) = W(\lfloor \frac{n}{2} \rfloor) + W(\lceil \frac{n}{2} \rceil) + n - 1 \text{ if } n \geq 2 \tag{49}$$

$$W(1) = 0 \tag{50}$$

that easily follows from the description (Algorithm 3.1 page 6) of `MergeSort`, steps 2a, 2c and Theorem 3.3. I am going to prove, by direct inspection, that the function $W(n)$ defined by (21) satisfies equations (49) and (50).

By (21),

$$W(1) = 1\lceil \lg 1 \rceil - 2^{\lceil \lg 1 \rceil} + 1 = 0 - 1 + 1 = 0.$$

Thus $W(n)$ satisfies (50).

Let $n \geq 2$. I am going to show that

$$\underbrace{n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1}_{W(n)} =$$

$$= \underbrace{\lfloor \frac{n}{2} \rfloor \lceil \lg \lfloor \frac{n}{2} \rfloor \rceil - 2^{\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil} + 1}_{W(\lfloor \frac{n}{2} \rfloor)} + \underbrace{\lceil \frac{n}{2} \rceil \lceil \lg \lceil \frac{n}{2} \rceil \rceil - 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil} + 1}_{W(\lceil \frac{n}{2} \rceil)} + n - 1. \tag{51}$$

This will prove (49).

I consider two cases.

Case 1: $n \neq 2^{\lfloor \lg n \rfloor} + 1$. In such a case,

$$\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil = \lceil \lg \lceil \frac{n}{2} \rceil \rceil. \tag{52}$$

Indeed, (49) is equivalent to $2^{\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil} = 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil}$, which means that the smallest power of 2 that is not less than $\lfloor \frac{n}{2} \rfloor$ (the left-hand side of the preceding equation) is equal to the smallest power of 2 that is not less than $\lceil \frac{n}{2} \rceil$ (the

[52] For instance, derived in [1] and [2].

right-hand side of that equation). They may be different if, and only if, $\lfloor \frac{n}{2} \rfloor = 2^k$ and $\lceil \frac{n}{2} \rceil = 2^k + 1$, for some integer $k \geq 0$, that is, iff $n = 2^{k+1} + 1$.

Substituting (52) to the right-hand side of (51) we get

$$\underbrace{\lfloor \frac{n}{2} \rfloor \lceil \lg \lceil \frac{n}{2} \rceil \rceil}_{\text{group}} - 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil} + 1 + \underbrace{\lceil \frac{n}{2} \rceil \lceil \lg \lceil \frac{n}{2} \rceil \rceil}_{\text{together}} - 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil} + 1 + n - 1 =$$

[by $\lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil = n$]

$$\underbrace{n \lceil \lg \lceil \frac{n}{2} \rceil \rceil + n}_{\text{factor}} - 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil + 1} + 1 = n(\lceil \lg \lceil \frac{n}{2} \rceil \rceil + 1) - 2^{\lceil \lg \lceil \frac{n}{2} \rceil \rceil + 1} + 1 =$$

[by $\lceil \lg \lceil \frac{n}{2} \rceil \rceil = \lceil \lg \frac{n}{2} \rceil$ (left as an exercise for the reader) and $\lceil x \rceil + 1 = \lceil x + 1 \rceil$]

$$= n \underbrace{\lceil \lg \frac{n}{2} + 1 \rceil}_{\lg n} - 2^{\overbrace{\lceil \lg \frac{n}{2} + 1 \rceil}^{\lg n}} + 1 = n \lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1.$$

Thus (51) is satisfied in this case.

Case 2: $n = 2^{\lfloor \lg n \rfloor} + 1$. In such a case,

$$\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil = \lceil \lg \lceil \frac{n}{2} \rceil \rceil - 1. \tag{53}$$

Indeed, since as we argued in *Case 1*, $n = 2^{\lfloor \lg n \rfloor} + 1$ implies $\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil \neq \lceil \lg \lceil \frac{n}{2} \rceil \rceil$. Therefore $\lceil \lg \lfloor \frac{n}{2} \rfloor \rceil < \lceil \lg \lceil \frac{n}{2} \rceil \rceil \leq \lceil \lg \lfloor \frac{n}{2} \rfloor \rceil + 1$, so that (53) must hold.

Also,

$$\lfloor \frac{n}{2} \rfloor = \lfloor \frac{2^{\lfloor \lg n \rfloor} + 1}{2} \rfloor = \lfloor 2^{\lfloor \lg n \rfloor - 1} + \frac{1}{2} \rfloor = 2^{\lfloor \lg n \rfloor - 1} =$$

[because n is not a power of 2 so that $\lg n$ is not integer and $\lfloor \lg n \rfloor = \lceil \lg n \rceil - 1$]

$$= 2^{\lceil \lg n \rceil - 2} = 2^{\lceil \lg \frac{n}{2} \rceil - 1}.$$

Thus

$$\lfloor \frac{n}{2} \rfloor = 2^{\lceil \lg \frac{n}{2} \rceil - 1}. \tag{54}$$

Substituting (53) to the right-hand side of (51) we get

$$\underbrace{\lfloor\frac{n}{2}\rfloor(\lceil\lg\lceil\frac{n}{2}\rceil\rceil}_{\text{group}}-1)-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil-1}+1+\underbrace{\lceil\frac{n}{2}\rceil\lceil\lg\lceil\frac{n}{2}\rceil\rceil}_{\text{together}}-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil}+1+n-1=$$

[by $\lfloor\frac{n}{2}\rfloor+\lceil\frac{n}{2}\rceil=n$]

$$=\underbrace{n\lceil\lg\lceil\frac{n}{2}\rceil\rceil+n}_{\text{factor}}-\lfloor\frac{n}{2}\rfloor-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil-1}-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil}+1=$$

[by (54)]

$$=n(\lceil\lg\lceil\frac{n}{2}\rceil\rceil+1)-2^{\lceil\lg\frac{n}{2}\rceil-1}-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil-1}-2^{\lceil\lg\lceil\frac{n}{2}\rceil\rceil}+1=$$

[by $\lceil\lg\lceil\frac{n}{2}\rceil\rceil=\lceil\lg\frac{n}{2}\rceil$ (left as an exercise for the reader) and $\lceil x\rceil+1=\lceil x+1\rceil$]

$$=n\lceil\lg\frac{n}{2}+1\rceil-2^{\lceil\lg\frac{n}{2}\rceil-1}-2^{\lceil\lg\frac{n}{2}\rceil-1}-2^{\lceil\lg\frac{n}{2}\rceil}+1=$$

$$=n\lceil\lg n\rceil-2^{\lceil\lg\frac{n}{2}\rceil}-2^{\lceil\lg\frac{n}{2}\rceil}+1=n\lceil\lg n\rceil-2^{\lceil\lg\frac{n}{2}\rceil+1}+1=n\lceil\lg n\rceil-2^{\lceil\lg n\rceil}+1.$$

Thus (51) is satisfied in this case.

Since there are no more cases, this - by virtue of Corollary 2.2 page 5 - competes the proof of Theorem 4.7.

8. Other work

Although some variants of parts of the formula (21) appear to have been known for quite some time now, even otherwise precise texts offer derivations that leave room for improvement. For instance, the recurrence relation for `MergeSort` analyzed in [4] asserts that the least number of comparisons of keys performed outside the recursive calls, if any, that suffice to sort an array of size n is n rather than $n-1$. This seemingly inconsequential variation results in a solution $W(n)=\sum_{i=1}^{n-1}(\lfloor\lg i\rfloor+2)$ [53] on page 2, Exercise 1.4, rather than the correct formula (12) $W(n)=\sum_{i=1}^{n}\lceil\lg i\rceil$ derived in this paper. (Also, the relevant derivations presented in [4], although quite clever, are not nearly

[53] I saw $W(n)=\sum_{i=1}^{n-1}\lfloor\lg i\rfloor$ on slides that accompany [4].

as precise and elementary as those presented in this paper.) As a result, the fact that `MergeSort` performs exactly the same number of comparisons of keys as does another classic, *binary insertion sort*, considered by H. Steinhaus and analyzed in [3], remains unnoticed.

Pages 176 – 177 of [2] contain an early sketch of proof of

$$W(n) = nh - 2^h + 1, \tag{55}$$

where h is the depth of the recursion tree T_n, with remarkably close[54] bounds given by (56) page 31. It is similar[55] to a simpler derivation based on the equality (10), presented in this paper in Section 4 and outlined in (12) page 9 (except for the $\sum_{i=1}^{n} \lceil \lg i \rceil$ part), which it predates by several years.

The [2]'s version of the decision tree T_n (Figure 4.14 page 177 of [2], shown here on Figure 7) was a re-use of a decision tree for the special case of $n = 2^{\lfloor \lg n \rfloor}$, with an ambiguous, if at all correct[56], comment in the caption that "[w]henever a node size parameter is odd, the left child size parameter is rounded up[57] and the right child size is rounded down[58]." The proof of the fact, needed for the derivation in [2], that T_n had no leaves outside its last two levels (Corollary 6.8 page 25, not needed for the derivation presented in Section 4) was waved with a claim "[w]e can[59] determine that [...]"

---

[54] Although not 100 percent correct.

[55] The idea behind the sketch of the derivation in [2] was based on an observation that

$$W(n) = \sum_{i=0}^{h-2} (n - 2^i) + \frac{n - B}{2},$$

where B was the number of leaves at the level $h - 1$ of the decision tree T_n; it was sketchily derived from the recursion tree shown on Figure 7 and properties stated in the Corollary 6.9 page 26 (with only a sketch of proof in [2]) not needed for the derivation presented in Section 4.

[56] It may be interpreted as to imply that for any level k, all the left-child sizes at level k are the same and all the right-child sizes at level k are the same, neither of which is a valid statement.

[57] Should be: *down*, according to (49) page 27.

[58] Should be: *up*, according to (49) page 27.

[59] This I do not doubt.

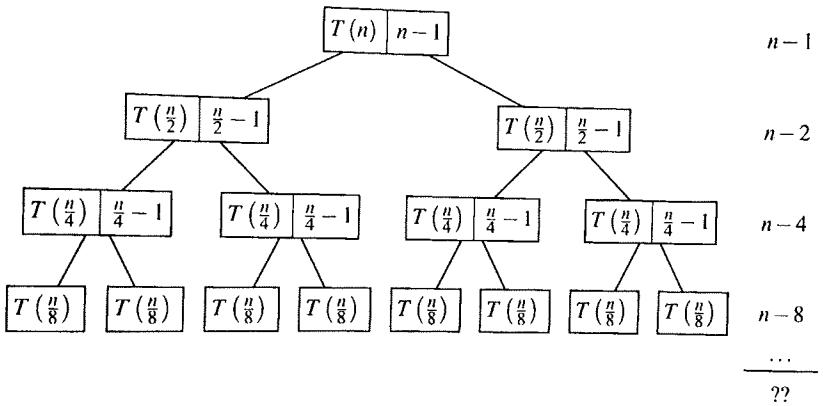

Figure 4.14 Recursion tree for Mergesort. Wherever a node size parameter is odd, the left child size is rounded up and the right child size is rounded down.

Figure 7: A snapshot from [2], page 177, showing a decision tree for `MergeSort`. ***Note***: This picture is copyrighted by *Addison Wesley Longman* (2000). It was reproduced here from [2] for *criticism* and *comment* purposes only, and not for any other purpose, as prescribed by U.S. Code Tittle 17 Chapter 1 para 107 that established the "fair use" exception of copyrighted material.

Although h was claimed in [2] to be equal to $\lceil \lg(n+1) \rceil$ [60] (and not to the correct $\lceil \lg n \rceil$ given by the equality (40) page 22, a fact not needed for the derivation presented in Section 4), somehow the mostly correct conclusion[61] was inferred from it, however, with no details offered - except for a mention that a function α that satisfies $h = \lg n + \lg \alpha$, similar to function ε shown on Figure 5 page 16, was used. It stated that (Theorem 4.6, page 177, in [2]):

$$\lceil n \lg n - n + 1 \rceil \leq W(n) \leq \lceil n \lg n - 0.914n \rceil. \tag{56}$$

It follows from (39) page 21 that the constant 0.914 that appears in (56) is incorrect. It was a rounding error[62], I suppose, that produced a false upper bound[63].

[60] Which claim must have produced an incorrect formula $n\lceil \lg(n+1) \rceil - 2^{\lceil \lg(n+1) \rceil} + 1$ for $W(n)$ and precluded concluding the neat characterization $W(n) = \sum_{i=1}^{n} \lceil \lg i \rceil$.

[61] Almost identical with (38) page 21, except for the constant 0.914.

[62] Of $1 - \delta$, where δ is given by (25) page 17.

[63] For instance, if $n = 11$ then `MergeSort` performs 29 comparisons of keys while the

9. Best-case analysis of MergeSort

It turns out that derivation of minimum number $B(n)$ of comps performed by MergeSort on an n-element array is a bit more tricky. A formula

$$\frac{n}{2}(\lfloor \lg n \rfloor + 1) - \sum_{k=0}^{\lfloor \lg n \rfloor} 2^k \, Zigzag\,(\frac{n}{2^{k+1}}), \tag{57}$$

where

$$Zigzag(x) = \min(x - \lfloor x \rfloor, \lceil x \rceil - x),$$

has been derived and thoroughly analyzed in [7]. It has been also demonstrated in [7] that there is no closed-form formula for $B(n)$.

Incidentally, as it was pointed out in [7] , $B(n)$ is equal to the sum $A(n, 2)$ of bits in binary representations of all integers $< n$.

Appendix A. A Java code of MergeSort

Figure A.8 shows a Java code of MergeSort.

```
93
94        public static int[] Sort(int[] A, int lo, int hi)
95    // input constrain: lo <= hi
96        {
97            if ((hi - lo) == 0)
98            {
99    //            return ({A[lo]});
100               int[] C = {A[lo]};
101               return C;
102           }
103           int splitPoint = (lo + hi)/2; // floor of ...
104           return Merge(Sort(A, lo, splitPoint),
105                   Sort(A, splitPoint + 1, hi));
106       }
107
```

Figure A.8: A Java code of MergeSort. A code of Merge is shown on Figure 2.

value of the upper bound $\lceil n \lg n - .914n \rceil$ given in [2], Theorem 4.6. p. 177, is 28; this is a significant error as 28 or less comps while sorting any 11-element array beats the *binary insertion sort* that requires $\sum_{i=1}^{11} \lceil \lg i \rceil = 29$ comps in the worst case.

Appendix B. Generating worst-case arrays for MergeSort

Figure B.9 shows a self-explanatory Java code of recursive method unSort that given a sorted array A reshuffles it, in a way resembling InsertionSort[64], onto a worst-case array for MergeSort.

```
115       public static void unSort(int[] A, int lo, int hi)
116   // turns A onto a worst-case array
117   // input constrain: lo <= hi and A is sorted
118       {
119           if ((hi - lo) <= 1) return; // already worst
120           int splitPoint = (lo + hi)/2; // floor of ...
121           int max = A[hi]; // will be sent to left array
122           for (int i = hi; i > splitPoint; i--)
123               A[i] = A[i-1]; //A[hi] is now 2nd largest
124           A[splitPoint] = max;
125           unSort(A, lo, splitPoint); // still sorted
126           unSort(A, splitPoint + 1, hi); // still sorted
127       }
```

Figure B.9: A Java code of unSort that, given a sorted array A, reshuffles it onto a worst-case array for MergeSort. Its structure mimics the Java code of MergeSort shown on Figure A.8.

For instance, it produced this array of integers between 1 and 500:

1, 500, 2, 3, 4, 7, 5, 6, 8, 15, 9, 10, 11, 14, 12, 13, 16, 31, 17, 18, 19, 22, 20, 21, 23, 30, 24, 25, 26, 29, 27, 28, 32, 62, 33, 34, 35, 38, 36, 37, 39, 46, 40, 41, 42, 45, 43, 44, 47, 61, 48, 49, 50, 53, 51, 52, 54, 60, 55, 56, 57, 59, 58, 63, 124, 64, 65, 66, 69, 67, 68, 70, 77, 71, 72, 73, 76, 74, 75, 78, 92, 79, 80, 81, 84, 82, 83, 85, 91, 86, 87, 88, 90, 89, 93, 123, 94, 95, 96, 99, 97, 98, 100, 107, 101, 102, 103, 106, 104, 105, 108, 122, 109, 110, 111, 114, 112, 113, 115, 121, 116, 117, 118, 120, 119, 125, 249, 126, 127, 128, 131, 129, 130, 132, 139, 133, 134, 135, 138, 136, 137, 140, 155, 141, 142, 143, 146, 144, 145, 147, 154, 148, 149, 150, 153, 151, 152, 156, 186, 157, 158, 159, 162, 160, 161, 163, 170, 164, 165, 166, 169, 167, 168, 171, 185, 172, 173, 174, 177, 175, 176, 178, 184, 179, 180, 181, 183, 182, 187, 248, 188, 189, 190, 193, 191, 192, 194, 201, 195, 196, 197, 200, 198, 199, 202, 216, 203, 204, 205, 208, 206, 207, 209, 215, 210, 211,

[64] Although not with InsertionSort's sluggishness; the number of moves of keys it performs is only slightly more than the *minimum* number (57) of comps performed by MergeSort on any n-element array.

212, 214, 213, 217, 247, 218, 219, 220, 223, 221, 222, 224, 231, 225, 226, 227, 230, 228, 229, 232, 246, 233, 234, 235, 238, 236, 237, 239, 245, 240, 241, 242, 244, 243, 250, 499, 251, 252, 253, 256, 254, 255, 257, 264, 258, 259, 260, 263, 261, 262, 265, 280, 266, 267, 268, 271, 269, 270, 272, 279, 273, 274, 275, 278, 276, 277, 281, 311, 282, 283, 284, 287, 285, 286, 288, 295, 289, 290, 291, 294, 292, 293, 296, 310, 297, 298, 299, 302, 300, 301, 303, 309, 304, 305, 306, 308, 307, 312, 373, 313, 314, 315, 318, 316, 317, 319, 326, 320, 321, 322, 325, 323, 324, 327, 341, 328, 329, 330, 333, 331, 332, 334, 340, 335, 336, 337, 339, 338, 342, 372, 343, 344, 345, 348, 346, 347, 349, 356, 350, 351, 352, 355, 353, 354, 357, 371, 358, 359, 360, 363, 361, 362, 364, 370, 365, 366, 367, 369, 368, 374, 498, 375, 376, 377, 380, 378, 379, 381, 388, 382, 383, 384, 387, 385, 386, 389, 404, 390, 391, 392, 395, 393, 394, 396, 403, 397, 398, 399, 402, 400, 401, 405, 435, 406, 407, 408, 411, 409, 410, 412, 419, 413, 414, 415, 418, 416, 417, 420, 434, 421, 422, 423, 426, 424, 425, 427, 433, 428, 429, 430, 432, 431, 436, 497, 437, 438, 439, 442, 440, 441, 443, 450, 444, 445, 446, 449, 447, 448, 451, 465, 452, 453, 454, 457, 455, 456, 458, 464, 459, 460, 461, 463, 462, 466, 496, 467, 468, 469, 472, 470, 471, 473, 480, 474, 475, 476, 479, 477, 478, 481, 495, 482, 483, 484, 487, 485, 486, 488, 494, 489, 490, 491, 493, 492.

It took my `MergeSort` 3,989 comps to sort it. Of course,

$$500\lceil \lg 500 \rceil - 2^{\lceil \lg 500 \rceil} + 1 = 4,500 - 512 + 1 = 3,989.$$

References

[1] S. BAASE, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley Publishing, 2nd ed., 1991.

[2] S. BAASE AND A. VAN GELDER, *Computer Algorithms; Introduction to Design & Analysis*, Asddison Wesley, 3rd ed., 2000.

[3] D. E. KNUTH, *The Art of Computer Programming*, vol. 3, Addison-Wesley Publishing, 2nd ed., 1997.

[4] R. SEDGEWICK AND P. FLAJOLET, *An Introduction to the Analysis of Algorithms*, Pearson, 2013.

[5] H. STEINHAUS, *Mathematical Snapshots*, Oxford University Press, 1950.

[6] M. A. SUCHENEK, *A complete worst-case analysis of heapsort with experimental verifcation of its results (MS)*, http://arxiv.org/abs/1504.01459, (2015).

[7] ——, *Best-case analysis of MergeSort with an application to the sum of digits problem (MS)*, http://arxiv.org/abs/1607.04604, (2016).